МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ МЕНЕДЖМЕНТУ

Кафедра економічної кібернетики,
комп'ютерних наук та інформаційних технологій

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Методичні рекомендації до виконання лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
ОПП «Комп'ютерні науки» спеціальності F3 (122) «Комп'ютерні науки»
денної форми здобуття вищої освіти

Миколаїв – 2025

УДК 004.42
А45

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету (протокол №1 від 28 серпня 2025 року)

Укладачі:

О. Ю. Пархоменко – к.ф.-м.н., доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

С. І. Тищенко – в.о. завідувача кафедри, к.п.н., доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

С. І. Ємельянов – PhD, старший викладач кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. О. Жебко – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. Є. Богатєнкова – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету.

Рецензенти:

В.В.Базаренко – заступник начальника Миколаївської обласної військової адміністрації з питань цифрового розвитку, цифрових трансформацій і цифровізації (CDTO);

Д.Л.Кошкін – к.т.н., дцент, доцент кафедри кафедри електроенергетики, електротехніки та електромеханіки Миколаївського національного аграрного університету.

А45 Алгоритмізація та програмування : методичні рекомендації до виконання лабораторних робіт для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності F3 (122) «Комп'ютерні науки» денної форми здобуття вищої освіти / уклад. О. Ю. Пархоменко, С. І. Тищенко, С. І. Ємельянов, О. О. Жебко, О. Є. Богатєнкова. Миколаїв : МНАУ, 2025. 435 с.

УДК 004.42

ЗМІСТ

ПЕРЕДМОВА

Дисципліна «Алгоритмізація та програмування» є фундаментальною у підготовці сучасного ІТ-фахівця. Вона не лише навчає синтаксису конкретної мови, а й формує алгоритмічне мислення – здатність розбивати складні проблеми на прості кроки та знаходити найбільш ефективні шляхи їх розв'язання. Ці методичні рекомендації, що складаються з 19 лабораторних робіт, пропонують структурований шлях від перших рядків коду до розробки комплексних графічних додатків на мові Python.

Python обрано не випадково. Ця мова поєднує в собі лаконічність, потужність та величезну кількість бібліотек, що робить її ідеальним інструментом як для навчання, так і для професійної розробки в галузях від веб-технологій до аналізу даних.

Структура посібника та етапи навчання

Методичні рекомендації побудовано за принципом «від простого до складного». На першому етапі студенти знайомляться з базовим синтаксисом: типами даних, змінними та арифметичними операторами. Особлива увага приділяється культурі написання коду, зокрема використанню зрозумілих назв змінних та сучасним способам форматування виводу через f-рядки.

Наступний логічний блок охоплює керуючі конструкції: розгалуження та цикли. Студенти вчаться будувати складні логічні умови та реалізовувати ітераційні процеси, що є основою будь-якого алгоритму. Лабораторна робота №5 вводить поняття вкладених структур, що відкриває шлях до обробки матриць та складних графічних шаблонів.

Робота з даними та функціональний підхід

Значна частина курсу присвячена структурам даних. Студенти опановують роботу з рядками як незмінними послідовностями символів. Вивчення списків, кортежів, словників та множин дозволяє зрозуміти, як ефективно організовувати інформацію та виконувати операції з пошуку та фільтрації з мінімальною часовою складністю.

Перехід до функцій ознаменовує новий рівень майстерності – декомпозицію. Студенти вчаться розбивати великі завдання на ізольовані модулі, що підлягають повторному використанню. Розширені можливості функцій, такі як рекурсія та функції вищого порядку (map, filter, reduce), дозволяють писати код у функціональному стилі.

Модульний підхід, вивчений у 11-й роботі, вчить створювати власні бібліотеки та працювати з файловими системами.

Алгоритми, структури даних та ООП

Розуміння алгоритмічної складності (Big O notation) та реалізація класичних структур, таких як стеки та черги, є критичними для підготовки інженера. Студенти порівнюють різні алгоритми сортування та пошуку, оцінюючи їхню ефективність на практиці.

Об'єктно-орієнтоване програмування (ООП) представлено як потужна парадигма для моделювання реальних сутностей. Через концепції інкапсуляції, наслідування та поліморфізму студенти вчаться створювати гнучкий та масштабований код. Використання спеціальних («магічних») методів та властивостей (properties) дозволяє створювати «пітонічні» класи, що органічно вписуються в екосистему мови.

Візуалізація, інтерфейси та фінальний проект

Завершальні роботи спрямовані на практичне застосування знань. Візуалізація даних за допомогою бібліотек Turtle та Matplotlib допомагає перетворювати цифри на зрозумілі графічні образи. Створення графічного інтерфейсу користувача (GUI) за допомогою Tkinter та CustomTkinter надає програмам завершеного, професійного вигляду.

Кульмінацією курсу є **Фінальний проект**. Це підсумкова робота, де студент має самостійно пройти всі етапи розробки програмного продукту: від проектування інтерфейсу до обробки винятків та документації.

Методичні особливості

Кожна лабораторна робота містить:

- чітко сформульовану мету та завдання;
- короткі теоретичні відомості з прикладами коду;
- завдання трьох рівнів складності: легкі (для засвоєння синтаксису), середні (для логічного мислення) та складні (для творчого підходу);
- аналіз типових помилок та поради щодо налагодження;
- питання для самоперевірки, що стимулюють глибше розуміння матеріалу.

Окремо заохочується використання інструментів штучного інтелекту не для копіювання, а для аналізу, оптимізації та кращого розуміння власного коду.

Ці методичні рекомендації стануть надійним путівником для кожного, хто прагне не просто вивчити Python, а стати справжнім майстром алгоритмізації.

Бажаємо успіхів у навчанні та натхнення у створенні ваших власних програмних шедеврів!

ЛАБОРАТОРНА РОБОТА №1
Вступ до Python:
перша програма, змінні та типи даних

1. Мета

Засвоєння базових принципів написання програм на Python. Студенти навчаться створювати прості програми, що виводять інформацію на екран, правильно оголошувати та ініціалізувати змінні, працювати з основними типами даних (цілі та дробові числа, рядки, логічні значення), а також виконувати перевірку та перетворення типів за допомогою вбудованих функцій.

2. Завдання

1. Оволодіти написанням першої програми та використанням коментарів.
2. Навчитися оголошувати змінні, присвоювати їм значення різних типів.
3. Зрозуміти відмінності між типами даних int, float, str, bool, NoneType.
4. Опанувати роботу з вбудованими функціями для перевірки (type()) та перетворення (int(), float(), str(), bool()) типів даних.
5. Розробити програми, які комбінують різні типи даних для Рішення простих завдань.

3. Короткі теоретичні відомості

3.1. Перша програма та виведення тексту

Програма на Python починається з інструкцій, які виконуються по порядку. Найпростіша дія – виведення тексту (рядка) на екран за допомогою функції print().

```
    # Це рядковий коментар. Він починається з символу # і ігнорується інтерпретатором.
    print("Привіт, світ!")    # Функція print виводить текст у консоль
```

3.2. Змінні

Змінна – це ім’я, пов’язане з певним значенням, яке зберігається в пам’яті комп’ютера. В Python не потрібно заздалегідь оголошувати тип змінної. Тип визначається автоматично відповідно до присвоєного

значення (динамічна типізація). Для присвоєння значення використовується оператор =.

```
# Оголошення та ініціалізація (присвоєння початкового значення) змінних
message = "Це текстовий рядок" # Змінна message тепер має тип "рядок"
number = 42                    # Змінна number тепер має тип "ціле число"
```

3.3. Основні типи даних

Python підтримує кілька вбудованих типів даних. Основні з них:

Тип даних	Назва	Опис	Приклад
int	Ціле число	Число без дробової частини	-5, 0, 100
float	Число з плаваючою точкою	Число з дробовою частиною	3.14, -0.001, 2.0
str	Рядок (String)	Текст, послідовність символів у лапках	"Привіт", ‘100’
bool	Булевий (Boolean)	Логічне значення (Істина/Хиба)	True, False
NoneType	Пусте значення	Спеціальне значення, що позначає відсутність значення	None

Приклад роботи з різними типами:

```
student_name = "Анна"    # str
age = 19                 # int
average_score = 84.5     # float
is_student = True        # bool
graduated = None         # NoneType - поки що не визначено

print(student_name, age, average_score, is_student, graduated)
```

3.4. Перевірка типів та приведення (перетворення) типів

Часто потрібно дізнатися тип значення змінної або перетворити одне значення в інший тип.

- **Функція** type() повертає тип об’єкта.

```
x = 10
print(type(x))   # Виведе: <class 'int'>
```

- **Функції перетворення типів:** int(), float(), str(), bool().
  - int() – перетворює значення на ціле число (відкидає дробову частину).
  - float() – перетворює значення на число з плаваючою точкою.
  - str() – перетворює значення на текстовий рядок.
  - bool() – перетворює значення на логічне (True/False).

```
    # Приведення типів
    a = "123"
    b = int(a)          # b = 123 (ціле число)
    c = float(b)        # c = 123.0 (число з плаваючою точкою)
    d = str(c)          # d = "123.0" (рядок знову)

# Увага: перетворення має сенс не завжди
# int("10.5")
# Помилка! Крапка заважає безпосередньому перетворенню в int
int(float("10.5"))
# Працює: спочатку в float (10.5), потім в int (10)
```

3.5. Комбінування типів даних та операції

При роботі з різними типами важливо пам'ятати про можливі операції. Деякі операції автоматично змінюють тип результату.

```
# Рядки можна "додавати" (конкатенувати) і "множити"
greeting = "Привіт, " + student_name  # "Привіт, Анна"
line = "-" * 20                       # "--------------------"

# Числа можна додавати, віднімати тощо
sum = age + 5                         # 24

# Помилка при невідповідності типів:
# result = "Мій вік: " + age  # TypeError! Не можна додавати рядок і число
result = "Мій вік: " + str(age)  # Правильно: "Мій вік: 19"
```

Ці знання є фундаментом для написання будь-якої програми на Python. У ході виконання завдань ви закріпите розуміння цих концепцій на практиці.

4. Методичні рекомендації

Нижче наведено розв'язання типових задач, які демонструють основні навички, необхідні для виконання лабораторної роботи. Кожен приклад включає пояснення логіки розв'язку та коментарі щодо застосування функцій та операцій.

Задача 1. Обчислення площі прямокутника

Напишіть програму, яка запитує у користувача довжину та ширину прямокутника (як цілі або дробові числа), обчислює його площу та виводить результат у зрозумілому форматі.

```
# Запит введення від користувача.
# Функція input() завжди повертає рядок (str).
length_str = input("Введіть довжину прямокутника: ")
width_str = input("Введіть ширину прямокутника: ")

# Перетворення введених рядків у числа з плаваючою точкою.
# Використовуємо float(), щоб обробляти як цілі, так і дробові числа.
length = float(length_str)
width = float(width_str)

# Обчислення площі.
area = length * width

# Форматування результату за допомогою f-рядка (f-string).
# :.2f - означає виведення числа з двома знаками після коми.
print(f"Площа прямокутника з довжиною {length} та шириною {width} дорівнює {area:.2f}")
```

Приклад вхідних та вихідних даних:

Вхід: 5, 3 → Вихід:
Площа прямокутника з довжиною 5.0 та шириною 3.0 дорівнює 15.00

Вхід: 4.2, 7.1 → Вихід:
Площа прямокутника з довжиною 4.2 та шириною 7.1 дорівнює 29.82

Вхід: 10, 2.5 → Вихід:
Площа прямокутника з довжиною 10.0 та шириною 2.5 дорівнює 25.00

Коментарі:

– Демонструє роботу з функціями input(), float() та print().

– Важливо перетворювати введення від користувача (str) в потрібний числовий тип (float), інакше операція множення призведе до помилки.

– F-рядки (f"...") є сучасним та зручним способом форматування виводу.

Задача 2. Формування персонального привітання.

Створіть програму, яка запитує ім'я та вік користувача, а потім виводить персоналізоване привітання, вказуючи, скільки років буде користувачеві через 5 років.

```
# Отримання даних від користувача
name = input("Як вас звати? ")
age_str = input("Скільки вам років? ")

# Перетворення віку з рядка в ціле число для арифметичних операцій
age_now = int(age_str)

# Обчислення віку у майбутньому
age_future = age_now + 5

# Виведення привітання. Увага: age_future - ціле число (int).
# При конкатенації рядків його потрібно перетворити в str().
print("Привіт, " + name + "!")
print("Зараз вам " + str(age_now) + " років.")
print("Через 5 років вам буде " + str(age_future) + " років.")
```

Приклад вхідних та вихідних даних:

```
Вхід: "Анна", "19" → Вихід:
Привіт, Анна!
Зараз вам 19 років.
Через 5 років вам буде 24 роки.
Вхід: "Олег", "23" → Вихід:
Привіт, Олег!
Зараз вам 23 роки.
Через 5 років вам буде 28 років.
```

Коментарі:

– Демонструє роботу з рядками (str) та цілими числами (int).

– Ключовий момент: змінна age_str типу str перетворюється в int для обчислення, а результат обчислення (age_future) знову перетворюється в str для виведення разом з іншими рядками.

– Показує різні способи формування виводу: конкатенація (+) та виклик print() з кількома аргументами.

Задача 3. Конвертер температури.

Напишіть програму, яка перетворює температуру з градусів Цельсія в градуси Фаренгейта за формулою: F = C * 9/5 + 32.

```
    # Отримання температури в Цельсіях.
    # Можна одразу перетворити результат input() у float.
    celsius = float(input("Введіть температуру в градусах Цельсія:
"))

    # Обчислення температури за Фаренгейтом.
    fahrenheit = celsius * 9 / 5 + 32

    # Виведення результату. Переконаємося, що celsius залишається
float для виводу.
    print(f"{celsius}°C дорівнює {fahrenheit:.1f}°F")
```

Приклад вхідних та вихідних даних:

```
Вхід: 0 → Вихід: 0.0°C дорівнює 32.0°F
Вхід: 36.6 → Вихід: 36.6°C дорівнює 97.9°F
Вхід: -10 → Вихід: -10.0°C дорівнює 14.0°F
```

Коментарі:

– Ілюструє використання арифметичних операцій з типом float.

– Важливо використовувати float() замість int(), щоб програма коректно працювала з дробовими значеннями температури.

– Форматування виводу :.1f забезпечує читабельність результату.

Задача 4. Перевірка коректності введення числа.

Напишіть програму, яка перевіряє, чи може введений користувачем рядок бути коректно перетворений на ціле число. Виведіть відповідне повідомлення та тип початкових даних.

```
    # Отримання вводу від користувача
    user_input = input("Введіть щось: ")

    # Спроба перетворити введення на ціле число
```

```
    try:
        number = int(user_input)
        print(f("Введення 'user_input}' успішно перетворено в ціле число {number}."))
        print(f("Початковий тип даних: {type(user_input)}"))
        print(f("Кінцевий тип даних: {type(number)}"))
    except ValueError:
        # Цей блок виконується, якщо int(user_input) викликає помилку ValueError
        print(f("Помилка! Введений текст 'user_input}' не можна інтерпретувати як ціле число.")
        print(f("Тип введених даних залишився: {type(user_input)}")
```

Приклад вхідних та вихідних даних:

```
Вхід: 42 → Вихід:
Введення '42' успішно перетворено в ціле число 42.
Початковий тип даних: <class 'str'>
Кінцевий тип даних: <class 'int'>

Вхід: 3.14 → Вихід:
Помилка! Введений текст '3.14' не можна інтерпретувати як ціле число.
Тип введених даних залишився: <class 'str'>

Вхід: Привіт → Вихід:
Помилка! Введений текст 'Привіт' не можна інтерпретувати як ціле число.
Тип введених даних залишився: <class 'str'>
```

Коментарі:

- Знайомить з базовою обробкою винятків (try...except), що є важливим для створення стійких програм.
- Наочно показує, що input() завжди повертає str, і не всі рядки можна перетворити на int.
- Конструкція try...except поки що подається як "магічний" код без глибокого пояснення механізму винятків.

Типові помилки і шляхи їх усунення

1. TypeError: can only concatenate str (not "int") to str

Причина. Спроба "додати" рядок та число за допомогою оператора +.

Рішення. Перетворити число в рядок за допомогою str(число) перед конкатенацією або використовувати f-рядки.

2. **ValueError: invalid literal for int() with base 10: '12.5'**

   Причина. Спроба перетворити рядок, що містить дробове число або текст, на ціле число за допомогою int().

   Рішення. Перевірити вхідні дані. Якщо очікується дробове число, використовуйте float(). Для перетворення дробу в ціле: int(float(рядок)).

3. **Неправильний результат обчислень з** float**.**

   Причина. Числа з плаваючою точкою (float) в комп'ютері зберігаються наближено. Наприклад, 0.1 + 0.2 може дати 0.30000000000000004.

   Рішення. Для грошових розрахунків використовуйте тип Decimal (мине пізніше). Для виводу округляйте результат за допомогою round() або форматування (f"{value:.2f}").

4. **Назва змінної не відображає її призначення.**

   Причина. Використання назв типу a, x1, var.

   Рішення. Називайте змінні описово англійською мовою: user_name, total_price, is_valid.

Корисні поради

1. **Назви змінних.** Використовуйте зрозумілі, описові назви англійською мовою. Розділяйте слова в назвах з нижнім підкресленням (наприклад, student_age). Це значно покращить читабельність вашого коду.

2. **Коментарі.** Коментуйте не *що* робить код (це видно з команд), а *навіщо* він це робить, якщо причина не очевидна.

3. **Тестування.** Завжди перевіряйте свою програму на різних вхідних даних: звичайних, крайніх (наприклад, 0, від'ємні числа) та некоректних (текст замість числа).

4. **Крок за кроком.** Не намагайтеся написати всю програму одразу. Напишіть код для отримання вводу, перевірте його. Потім додайте обчислення, знову перевірте. І лише потім – форматування виводу.

5. **Читайте повідомлення про помилки.** Python детально описує, що пішло не так і в якому рядку. Вміння читати Traceback – це 50% успіху в налагодженні.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Форматування привітання

Напишіть програму, яка виводить привітання студенту у форматі: "Вітаю, [Ім'я]! Ласкаво просимо до вивчення Python.". Використайте один виклик print() з конкатенацією рядків.

```
Вхід: "Анна" →
Вихід: Вітаю, Анна! Ласкаво просимо до вивчення Python.
Вхід: "Олег" →
Вихід: Вітаю, Олег! Ласкаво просимо до вивчення Python.
Вхід: "Марія" →
Вихід: Вітаю, Марія! Ласкаво просимо до вивчення Python.
```

Завдання 1.2. Таблиця множення (фрагмент)

Виведіть один рядок таблиці множення числа 7 у форматі "7 × 3 = 21". Використайте параметри sep та end функції print() для отримання результату без пробілів навколо знаків.

```
Вихід: 7×3=21
```

Завдання 1.3. Каталог змінних

Створіть чотири змінні: product_name (назва товару), price (ціна), in_stock (наявність на складі), weight (вага). Присвойте їм значення довільних, але відповідних типів. Виведіть тип кожної змінної.

```
Вихід: <class 'str'>, <class 'float'>, <class 'bool'>, <class 'int'>
```

Завдання 1.4. Аналіз значень None

Створіть змінні: middle_name зі значенням None, age з числовим значенням. Виведіть їхні типи за допомогою type(). Поясніть різницю у виводі.

```
Вхід: None, 25 → Вихід: <class 'NoneType'>, <class 'int'>
```

Завдання 1.5. Обчислення вартості

Обчисліть загальну вартість 5 одиниць товару, ціна якого становить 27.50 грн. Результат округліть до двох знаків після коми за допомогою форматування f-рядком.

```
Вихід: Загальна вартість: 137.50 грн.
```

Завдання 1.6. Формування email

З двох рядкових змінних first_name та last_name створіть третій рядок у форматі електронної адреси: "ім’я.прізвище@university.com". Усі літери мають бути нижнього регістру.

```
Вхід: "Anna", "Kovalenko" → Вихід: anna.kovalenko@university.com
Вхід: "Ivan", "Petrenko" → Вихід: ivan.petrenko@university.com
```

Завдання 1.7. Обрізання рядка

Змінна long_text містить рядок "Це дуже довгий рядок для прикладу". Створіть новий рядок, який містить лише перші 10 символів з long_text та останні 7 символів, з’єднаних разом.

```
Вихід: Це дуже дприкладу
```

Завдання 1.8. Перетворення логічного значення

Створіть змінну is_valid зі значенням True. Виведіть її значення, перетворене на ціле число та на рядок.

```
Вихід: 1, True (спочатку число, потім рядок через кому)
```

Завдання 1.9. Конкатенація чисел у рядок

Дано три цілих числа: a = 12, b = 7, c = 3. Створіть рядок, який є їх послідовністю без пробілів.

```
Вихід: 1273
```

Завдання 1.10. Арифметика з різними типами

Обчисліть вираз: 15 + 2.3 * 4. Виведіть результат та його тип. Потім перетворіть результат на ціле число і виведіть це число.

```
Вихід: 24.2 <class ‘float’>, 24
```

Частина 2. Середні завдання

Завдання 2.1. Конвертер валюти (фіксований курс)

Напишіть програму, яка запитує у користувача суму в гривнях (дробове число) та виводить еквівалент в євро за фіксованим курсом (наприклад, 40.5 грн за 1 євро). Виведіть суму в євро з двома знаками після коми.

```
Вхід: 4050 → Вихід: 100.00 €
Вхід: 202.5 → Вихід: 5.00 €
Вхід: 81 → Вихід: 2.00 €
```

Завдання 2.2. Валідація номера

Користувач вводить "номер квитка" - послідовність з 6 цифр у вигляді рядка. Програма повинна перетворити цей рядок на ціле число, додати до нього число 1000 та вивести новий "номер" вже у вигляді рядка з 7 символів, доповнивши його нулями зліва за потреби.

```
Вхід: "001234" → Вихід: "002234"
Вхід: "999999" → Вихід: "1000999"
Вхід: "000001" → Вихід: "001001"
```

Завдання 2.3. Генератор короткого звіту

Програма отримує три вхідних параметри через input(): назву проєкту (рядок), відсоток виконання (ціле число), бюджет (дробове число). Потрібно сформувати один рядок звіту: "Проєкт ‘[назва]’ виконано на [відсоток]%. Бюджет: [бюджет] грн.". Відсоток має бути цілим числом, бюджет - з двома знаками після коми.

```
Вхід: "Альфа", 75, 123456.7 →
Вихід: Проєкт ‘Альфа’ виконано на 75%. Бюджет: 123456.70 грн.
```

Завдання 2.4. Розрахунок часу

Користувач вводить кількість хвилин (ціле число). Програма перетворює це значення у години та хвилини і виводить у форматі "Годин: [години], Хвилин: [хвилини]". Використайте цілочисельне ділення // та остачу від ділення %.

```
Вхід: 150 → Вихід: Годин: 2, Хвилин: 30
Вхід: 61 → Вихід: Годин: 1, Хвилин: 1
Вхід: 720 → Вихід: Годин: 12, Хвилин: 0
```

Завдання 2.5. Формування наукового позначення

Дано дробове число x = 0.000456. Потрібно отримати його рядкове представлення у вигляді "4.56e-4". Підказка: використайте перетворення в рядок з форматуванням наукового запису :e з двома знаками після коми.

```
Вхід: 0.000456 → Вихід: 4.56e-04
Вхід: 1234567.89 → Вихід: 1.23e+06 (для додаткового тесту)
Вхід: -0.001 → Вихід: -1.00e-03
```

Частина 3. Складні завдання

Завдання 3.1. Калькулятор середнього балу з валідацією

Програма запитує три оцінки студента (від 0 до 100). Оцінки можуть бути введені як цілі або дробові числа. Завдання: перевірити, чи є кожна оцінка коректним числом (не викликає помилку при перетворенні в float). Якщо всі три коректні - обчислити середній бал, інакше - вивести повідомлення "Помилка введення". Для перевірки використайте try-except. Вивести тип кожної введеної оцінки після перетворення.

```
Вхід: 85, 92.5, 78 → Вихід: Середній бал: 85.17 | Типи: float, float, float
Вхід: 100, "95", 80 → Вихід: Помилка введення (лапки позначають, що введено рядок)
Вхід: 75.5, 80, abc → Вихід: Помилка введення
```

Завдання 3.2. Генератор пароля за шаблоном

Користувач вводить три числа: рік народження (4 цифри), місяць (2 цифри), день (2 цифри). Програма має створити "пароль" за шаблоном: "КМ-[останні_2_цифри_року][місяць][день*2]". Усі складові мають бути рядками. Якщо день менше 10, на початку має бути 0. Приклад: для 2002-03-07 → "КМ-020314". Для перетворення використайте str() та зрізи.

```
Вхід: 2001, 12, 25 → Вихід: КМ-011250
Вхід: 1999, 05, 09 → Вихід: КМ-990518
Вхід: 2010, 11, 03 → Вихід: КМ-101106
```

Завдання 3.3. Конвертер фізичних одиниць (комплексний)

Напишіть програму для переведення кілометрів на годину в метри за секунду та навпаки. Користувач вводить значення та одиницю вимірювання ("km/h" або "m/s"). Програма має вивести результат у обох одиницях з точністю до 3 знаків. Формули: m/s = km/h * 1000 / 3600, km/h = m/s * 3600 / 1000. Реалізуйте без умовних операторів, використавши словник (можна як наведений приклад конвертації) або математичну тотожність.

```
Вхід: 90 "km/h" → Вихід: 90.000 km/h = 25.000 m/s
Вхід: 10 "m/s" → Вихід: 10.000 m/s = 36.000 km/h
Вхід: 120 "km/h" → Вихід: 120.000 km/h = 33.333 m/s
```

6. Питання для самоперевірки

1. Яка функція в Python використовується для виведення інформації на екран? Які параметри цієї функції ви знаєте?

2. Що таке змінна в програмуванні? Яким оператором здійснюється присвоєння значення змінній в Python?

3. Наведіть приклади значень для кожного з п’яти основних типів даних: int, float, str, bool, NoneType.

4. Поясніть різницю між динамічною та статичною типізацією. До якого типу належить Python?

5. Чому функція input() завжди повертає значення типу str? Наведіть приклад.

6. Як можна дізнатися тип об’єкта (значення змінної) під час виконання програми?

7. Які функції використовуються для явного перетворення (приведення) типів даних? Наведіть по одному прикладу для int(), float(), str() та bool().

8. Що станеться, якщо спробувати виконати додавання рядка та цілого числа ("Рік: " + 2024)? Як можна виправити цю помилку?

9. Яке значення матиме змінна result після виконання коду: result = bool("False")? Поясніть, чому.

10. Що означає вираз type(None)? Яке практичне застосування у значення None?

11. Що таке f-рядок (f-string) і які його переваги перед конкатенацією за допомогою оператора +?

12. Чому результат виразу 0.1 + 0.2 у Python не дорівнює точно 0.3? Як можна отримати коректний результат для подібних обчислень?

13. Яку операцію виконує символ % при роботі з цілими числами? Наведіть приклад його використання.

14. Що таке PEP 8? Назвіть декілька основних правил цього керівництва зі стилю коду для Python.

15. Яка різниця між операторами / та // при діленні чисел?

16. Що виведе наступний код і чому: print(int(7.8) + float("3.2"))?

17. Як можна отримати довжину рядка? (Підказка: це вбудована функція).

18. Яке значення повертає функція bool() для порожнього рядка "", числа 0 та значення None?

19. Що таке коментар і для чого він використовується? Які способи створення коментарів у Python ви знаєте?

20. Поясніть принцип роботи та призначення конструкції try-except на простому прикладі, пов’язаному з перетворенням типів.

ЛАБОРАТОРНА РОБОТА №2
Оператори та введення/виведення даних

1. Мета

Оволодіти практичними навичками роботи з основними операторами мови Python та функціями введення-виведення даних. Закріпити розуміння пріоритетів операцій, навчитися формувати складні логічні умови, коректно оперувати різними типами даних при отриманні їх від користувача та ефективно форматувати результат виведення для його читабельності.

2. Завдання

1. Закріпити знання з синтаксису та семантики основних операторів мови Python.
2. Навчитися отримувати дані від користувача за допомогою функції input() та конвертувати їх у потрібний тип.
3. Опанувати методи форматування рядків виведення з використанням f-рядків (f-strings) та методу format().
4. Розвинути вміння складати арифметичні та логічні вирази для розв'язання практичних задач.
5. Набути досвіду написання простих, але завершених програм, що взаємодіють з користувачем.

3. Короткі теоретичні відомості

Для роботи з даними в будь-якій мові програмування необхідні оператори та механізми отримання і відображення інформації. У Python ці інструменти є потужними, але простими у використанні.

3.1. Введення даних: функція input()

Функція input() призупиняє виконання програми і чекає, поки користувач введе текст з клавіатури та натисне Enter. Результатом роботи функції завжди є *рядок* (тип str).

```
user_name = input("Як вас звати? ")
print("Привіт,", user_name)
```

Щоб працювати з введеними даними як з числами, необхідно виконати явне перетворення типів (конвертацію) за допомогою функцій int() (ціле число) або float() (число з плаваючою точкою).

```
age_str = input("Скільки вам років? ")
age_int = int(age_str)  # Конвертація рядка в ціле число
next_age = age_int + 1
```

Конвертацію можна виконати відразу:

```
radius = float(input("Введіть радіус кола: "))
area = 3.14159 * radius ** 2
```

3.2. Виведення даних та форматування

Функція print() виводить значення на екран. Для створення зрозумілих повідомлень необхідно об'єднувати дані та текст. Найсучаснішим та рекомендованим способом є **f-рядки** (з Python 3.6+).

```
name = "Анна"
score = 95
# Використання f-рядка для форматування
print(f"Студент {name} отримав {score} балів.")
# Виведення: Студент Анна отримав 95 балів.
```

Всередині фігурних дужок {} можна розміщувати не лише змінні, а й цілі вирази:

```
a = 5
b = 3
print(f"{a} + {b} = {a + b}")  # 5 + 3 = 8
print(f"Середнє: {(a + b) / 2:.2f}")  # Форматування числа: .2f - 2 знаки після коми
```

Альтернативний спосіб – метод format():

```
template = "Координати: X={}, Y={}"
result = template.format(10, 20)
print(result)  # Координати: X=10, Y=20
```

3.3. Основні оператори

Арифметичні оператори використовують для обчислень.

Оператор	Дія	Приклад	Результат
+	Додавання	7 + 3	10
-	Віднімання	7 - 3	4
*	Множення	7 * 3	21
/	Ділення	7 / 3	2.333...
//	Цілочисл. ділення	7 // 3	2
%	Остача від ділення (modulo)	7 % 3	1
**	Піднесення до степеня	2 ** 3	8

Оператори присвоєння використовуються для збереження значення у змінній. Базовий оператор =. Існують також комбіновані оператори.

```
x = 10
x += 5    # Еквівалент x = x + 5, тепер x = 15
x *= 2    # x = x * 2, тепер x = 30
x **= 2   # x = x ** 2, тепер x = 900
```

Оператори порівняння повертають логічне значення True (Істина) або False (Хиба). Вони є основою для прийняття рішень в програмах.

Оператор	Перевірка	Приклад	Результат
==	Дорівнює	5 == 3	False
!=	Не дорівнює	5 != 3	True
>	Більше	5 > 3	True
<	Менше	5 < 3	False
>=	Більше або дорівнює	5 >= 5	True
<=	Менше або дорівнює	5 <= 3	False

Логічні оператори використовуються для об'єднання або інверсії умов.

- and (І): Повертає True, якщо **обидві** умови істинні.
- or (АБО): Повертає True, якщо **хоча б одна** з умов істинна.
- not (НЕ): Інвертує логічне значення (True стає False, і навпаки).

```
temperature = 25
is_weekend = True
# Складена умова: хороша погода для прогулянки
good_for_walk = temperature > 20 and temperature < 30 and
is_weekend
print(good_for_walk)  # True
# Умова для закритого приміщення
stay_inside = temperature <= 0 or temperature >= 35
print(stay_inside)    # False (при temperature=25)
is_rainy = False
print(not is_rainy)   # True - дощ не йде, можна виходити
```

3.4. Пріоритет операцій

Порядок виконання операцій в Python відповідає математичним правилам. Для зміни порядку використовуються круглі дужки ().

1. Дужки ()
2. Піднесення до степеня **

3. Множення *, ділення /, цілочислене ділення //, остача %
4. Додавання +, віднімання -
5. Оператори порівняння (<, <=, >, >=, ==, !=)
6. Логічне not
7. Логічне and
8. Логічне or

Приклад:

```
result = 5 + 2 * 3 ** 2 / (4 - 1)
# 5 + 2 * 9 / 3 = 5 + 18 / 3 = 5 + 6 = 11
print(result)  # 11.0
logical_check = not (5 > 3) and (10 == 20) or (15 >= 10)  # False and False or True = True
print(logical_check)  # True
```

Важливо: завжди використовуйте дужки для явного вказіввання порядку в складних виразах. Це робить код набагато зрозумілішим.

4. Методичні рекомендації

Нижче наведені розв'язки типових задач, які допоможуть зрозуміти застосування вивчених концепцій на практиці. Уважно прочитайте умову, вивчіть код та приклади роботи програми.

Задача 1. Обчислення площі прямокутника

Написати програму, яка запитує у користувача довжину та ширину прямокутника, обчислює його площу та периметр і виводить результати в зручному форматі.

```
# Отримуємо дані від користувача. Функція input() завжди повертає рядок.
length_str = input("Введіть довжину прямокутника (см): ")
width_str = input("Введіть ширину прямокутника (см): ")

# Конвертуємо рядки у числа з плаваючою точкою для математичних обчислень.
length = float(length_str)
width = float(width_str)
# Обчислюємо площу та периметр.
area = length * width
perimeter = 2 * (length + width)
```

```
    # Виводимо результати, використовуючи f-рядки для форматування.
    print(f"Прямокутник зі сторонами {length} см та {width} см:")
    print(f"  Площа: {area} кв.см")
    print(f"  Периметр: {perimeter} см")
```

Приклад вхідних та вихідних даних:

```
Вхід: 10 та 5 → Вихід:
Прямокутник зі сторонами 10.0 см та 5.0 см:
Площа: 50.0 кв.см
Периметр: 30.0 см
Вхід: 7.5 та 3.2 → Вихід:
Прямокутник зі сторонами 7.5 см та 3.2 см:
Площа: 24.0 кв.см
Периметр: 21.4 см
```

Коментарі:

Ця задача демонструє класичний ланцюжок input -> конвертація -> обчислення -> output. Зверніть увагу на використання float() замість int(), щоб програма коректно працювала з дробовими значеннями. F-рядок використано для створення зрозумілого виведення.

Задача 2. Конвертер секунд

Написати програму, яка отримує від користувача кількість секунд і конвертує їх у хвилини та секунди (наприклад, 125 секунд = 2 хвилини 5 секунд).

```
    # Отримуємо загальну кількість секунд
    total_seconds = int(input("Введіть кількість секунд: "))

    # Цілочисленне ділення на 60 дає кількість повних хвилин
    minutes = total_seconds // 60

    # Остача від ділення на 60 дає секунди, що залишилися
    seconds = total_seconds % 60

    # Форматуємо результат з перевіркою на правильні форми слів
    minutes_word = "хвилина" if minutes == 1 else "хвилини" if 2 <= minutes <= 4 else "хвилин"
    seconds_word = "секунда" if seconds == 1 else "секунди" if 2 <= seconds <= 4 else "секунд"
```

```
    print(f"{total_seconds} секунд це {minutes} {minutes_word} {seconds} {seconds_word}")
```

Приклад вхідних та вихідних даних:

```
Вхід: 125 → Вихід: 125 секунд це 2 хвилини 5 секунд
Вхід: 71 → Вихід: 71 секунд це 1 хвилина 11 секунд
Вхід: 3600 → Вихід: 3600 секунд це 60 хвилин 0 секунд
```

Коментарі:

Задача ідеально ілюструє використання операторів цілочисленого ділення (//) та остачі (%). Оператор % часто використовується для розбиття загальної кількості на окремі одиниці (години-хвилини, гривні-копійки). Ми також почали використовувати умовні вирази для коректного відображення слів, це додає програмі «інтелекту».

Задача 3. Перевірка парних/непарних чисел

Написати програму, яка запитує ціле число і визначає, чи є воно парним, непарним, а також додатним, від'ємним або нулем. Вивести короткий опис числа. Тут ми заздалегідь вводимо умовні конструкції для демонстрації; детальніше – у наступних роботах.

```
    # Отримуємо число від користувача
    number = int(input("Введіть ціле число: "))

    # Визначаємо парність за допомогою оператора %.
    # Якщо остача від ділення на 2 дорівнює 0 – число парне.
    is_even = number % 2 == 0

    # Визначаємо знак числа за допомогою операторів порівняння
    if number > 0:
        sign_description = "додатне"
    elif number < 0:
        sign_description = "від'ємне"
    else:
        sign_description = "нуль"

    # Формуємо фінальний висновок на основі логічної змінної
    parity_description = "парне" if is_even else "непарне"

    print(f"Число {number} є {sign_description} і {parity_description}.")
```

Приклад вхідних та вихідних даних:

```
Вхід: 7 → Вихід: Число 7 є додатне і непарне.
Вхід: -4 → Вихід: Число -4 є від'ємне і парне.
Вхід: 0 → Вихід: Число 0 є нуль і парне.
```

Коментарі:

Задача демонструє комбінацію операторів порівняння (==, >, <) та логічного об'єднання результатів у фінальному рядку. Зверніть увагу, що перевірка number % 2 == 0 повертає логічне значення (True або False), яке потім інтерпретується в текст.

Задача 4. Обмін значень змінних

Написати програму, яка запитує у користувача два числа, зберігає їх у змінних a та b, а потім обмінює їх значення без використання додаткової змінної. Вивести значення до та після обміну.

```python
# Отримуємо два числа
a = int(input("Введіть перше число (a): "))
b = int(input("Введіть друге число (b): "))

print(f"До обміну:   a = {a}, b = {b}")

# Класичний спосіб обміну з використанням третьої змінної
# temp = a
# a = b
# b = temp
# Python-стиль обміну значень
a, b = b, a

print(f"Після обміну: a = {a}, b = {b}")
```

Приклад вхідних та вихідних даних:

```
Вхід: 5 та 10 →
Вихід: До обміну: a = 5, b = 10 та Після обміну: a = 10, b = 5
Вхід: -3 та 7 →
Вихід: До обміну: a = -3, b = 7 та Після обміну: a = 7, b = -3
```

Коментарі:

Задача показує елегантну можливість Python – множинне присвоєння (a, b = b, a). У більшості інших мов для цього потрібна третя, тимчасова змінна. Це гарний приклад для обговорення різниці між присвоєнням значення (=) та порівнянням (==).

Задача 5. Калькулятор середнього балу

Написати програму, яка запитує у студента оцінки за три модулі, обчислює середній бал та виводить його, округленого до двох знаків після коми. Також програма має вивести порівняння середнього балу з пороговим значенням (наприклад, 60 балів).

```
    # Отримуємо три оцінки
    grade1 = float(input("Введіть оцінку за модуль 1: "))
    grade2 = float(input("Введіть оцінку за модуль 2: "))
    grade3 = float(input("Введіть оцінку за модуль 3: "))

    # Обчислюємо середнє арифметичне
    average = (grade1 + grade2 + grade3) / 3

    # Порогове значення для порівняння
    threshold = 60.0

    # Порівнюємо середній бал з порогом
    is_above_threshold = average >= threshold

    # Виводимо результат з форматуванням числа
    print(f"Ваш середній бал: {average:.2f}")
    print(f"Цей бал {'вище або дорівнює' if is_above_threshold
else 'нижче'} порогового значення ({threshold}).")
```

Приклад вхідних та вихідних даних:

Вхід: 70.5, 65.0, 80.0 →

Вихід: Ваш середній бал: 71.83 та Цей бал вище або дорівнює порогового значення (60.0).

Вхід: 45.0, 50.0, 55.0 →

Вихід: Ваш середній бал: 50.00 та Цей бал нижче порогового значення (60.0).

Коментарі:

Задача поєднує арифметичні обчислення, оператори порівняння та форматування виведення чисел (:.2f). Вираз всередині f-рядка для вибору тексту (‘вище або дорівнює’ if ... else ‘нижче’) є прикладом умовного виразу і часто використовується для лаконічного коду.

Типові помилки і шляхи їх усунення

1. TypeError: can only concatenate str (not "int") to str

Причина. Спроба «додати» (+) рядок та число. input() повертає рядок, і без конвертації арифметичні операції неможливі.

Рішення. Завжди конвертуйте введення в потрібний числовий тип (int() або float()) перед обчисленнями.

2. **ValueError: invalid literal for int() with base 10: '12.5'**

Причина. Спроба конвертувати рядок з десятковою точкою ("12.5") в ціле число (int()).

Рішення. Використовуйте float() для конвертації чисел з дробовою частиною. Якщо потрібно ціле, округліть (round()) або відкиньте дробову частину (int(float(...))).

3. **Неправильний результат при діленні**

Причина. Оператор / завжди повертає float. Якщо потрібна ціла частина або остача, це не завжди очевидно.

Рішення. Чітко розрізняйте оператори ділення: / (звичайне), // (цілочисленне), % (остача). Використовуйте // та % для роботи з окремими розрядами або одиницями виміру.

4. **Нерозуміння логіки операторів** and **та** or

Причина. Спроба написати математичний діапазон як 0 < x < 10, але з використанням змінних або складних умов.

Рішення. Пам'ятайте, що Python дозволяє 0 < x < 10. Для складних умов явно вказуйте кожне порівняння та об'єднуйте їх: (x > 0) and (x < 10). Оператор and вимагає виконання всіх умов, or – хоча б однієї.

5. **Плутанина між** = **та** ==

Причина. У математиці = означає рівність. У програмуванні = – це оператор присвоєння, а == – порівняння.

Рішення. Завжди перевіряйте: чи я хочу зберегти значення у змінній (=) чи перевірити на рівність (==)?

Корисні поради

1. **Контроль введення.** Завжди пишіть інформативні підказки в input(). Це допомагає користувачеві зрозуміти, що від нього очікують.

2. **Поетапна розробка.** Не пишіть весь код одразу. Почніть з отримання даних та їх виведення. Потім додайте конвертацію, потім – обчислення, потім – форматування. Так легше знаходити помилки.

3. **Коментарі та назви змінних.** Давайте змінним зрозумілі англійські назви (radius, average_score). Код має бути

самодокументованим. Коментарі пишіть для пояснення чому зроблено саме так, а не що робить рядок (це видно з коду).

4. **Тестування на різних даних.** Запускайте програму не тільки на «звичайних» числах. Спробуйте ввести 0, від'ємні числа, дуже великі числа, текст замість числа (це викличе помилку, і це важливо розуміти). Це допомагає переконатися в надійності програми.

5. **F-рядки – ваш найкращий друг.** Для форматування виведення використовуйте f-рядки (f"Текст {змінна}"). Вони набагато зручніші та читабельніші за конкатенацію рядків (+) або старий метод format().

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Вітання з користувачем

Напишіть програму, яка запитує ім'я користувача, а потім виводить привітання у форматі: "Вітаю, [ім'я]! Сподіваюся, у вас чудовий день!".

```
Вхід: Анна → Вихід: Вітаю, Анна! Сподіваюся, у вас чудовий день!
Вхід: Олег → Вихід: Вітаю, Олег! Сподіваюся, у вас чудовий день!
Вхід: Mary → Вихід: Вітаю, Mary! Сподіваюся, у вас чудовий день!
```

Завдання 1.2. Сума та різниця

Напишіть програму, яка отримує два цілих числа та виводить їх суму і різницю (перше мінус друге) в одному рядку через кому.

```
Вхід: 7, 3 → Вихід: 10, 4
Вхід: -5, 8 → Вихід: 3, -13
Вхід: 0, 0 → Вихід: 0, 0
```

Завдання 1.3. Середнє арифметичне трьох чисел

Напишіть програму, яка отримує три числа (можуть бути дробовими) та обчислює їхнє середнє арифметичне. Результат вивести без форматування.

```
Вхід: 4.5, 5.5, 3.0 → Вихід: 4.333333333333333
Вхід: 10, 20, 30 → Вихід: 20.0
Вхід: 1, 1, 1 → Вихід: 1.0
```

Завдання 1.4. Перевірка подільності

Напишіть програму, яка запитує ціле число і перевіряє, чи ділиться воно на 5 без остачі. Виведіть True, якщо ділиться, і False – якщо ні.

```
Вхід: 25 → Вихід: True
Вхід: 17 → Вихід: False
Вхід: 0 → Вихід: True
```

Завдання 1.5. Порівняння квадратів

Напишіть програму, яка запитує два цілих числа та перевіряє, чи квадрат першого числа більше за друге число. Виведіть результат порівняння (True або False).

```
Вхід: 5, 20 → Вихід: True (5²=25 > 20)
Вхід: 3, 10 → Вихід: False (3²=9 < 10)
Вхід: 4, 16 → Вихід: False (4²=16 не більше, а дорівнює)
```

Завдання 1.6. Належність числа до діапазону

Напишіть програму, яка запитує число і перевіряє, чи належить воно до діапазону [10; 50] (включно з кінцями). Виведіть True або False.

```
Вхід: 25 → Вихід: True
Вхід: 10 → Вихід: True
Вхід: 51 → Вихід: False
```

Завдання 1.7. Перевірка парності та додатності

Напишіть програму, яка запитує ціле число і виводить результат логічної операції: чи є число одночасно парним і додатним. Виведіть True або False.

```
Вхід: 8 → Вихід: True (парне і додатне)
Вхід: -4 → Вихід: False (парне, але не додатне)
Вхід: 7 → Вихід: False (додатне, але не парне)
```

Завдання 1.8. Вихідні або будень

Користувач вводить назву дня тижня (напр., "понеділок"). Напишіть програму, яка виводить True, якщо це вихідний день ("субота" або "неділя"), і False – в іншому випадку. Врахуйте, що введення може бути з малої літери.

```
Вхід: субота → Вихід: True
Вхід: Середа → Вихід: False
Вхід: неділя → Вихід: True
```

Завдання 1.9. Заперечення умови

Напишіть програму, яка запитує число і перевіряє, чи воно НЕ дорівнює нулю. Виведіть результат логічної операції.

```
Вхід: 5 → Вихід: True (5 не дорівнює 0)
Вхід: -3 → Вихід: True (-3 не дорівнює 0)
Вхід: 0 → Вихід: False (0 дорівнює 0, заперечення дає False)
```

Завдання 1.10. Умова для знижки

Магазин дає знижку, якщо покупець старший за 60 років АБО купує більше 10 одиниць товару. Напишіть програму, яка запитує вік покупця та кількість одиниць товару, і виводить True, якщо знижка надається, і False – якщо ні.

```
Вхід: 65, 5 → Вихід: True (вік > 60)
Вхід: 25, 12 → Вихід: True (кількість > 10)
Вхід: 40, 8 → Вихід: False (жодна умова не виконана)
```

Частина 2. Середні завдання

Завдання 2.1. Обчислення залишку від ділення на 7

Напишіть програму, яка отримує тризначне ціле число (переконуватися в тризначності не потрібно) і обчислює суму його цифр, а потім знаходить остачу від ділення цієї суми на 7. Використайте лише арифметичні оператори (без рядків).

```
Вхід: 123 → Вихід: 6 (1+2+3=6, 6%7=6)
Вхід: 999 → Вихід: 6 (9+9+9=27, 27%7=6)
Вхід: 100 → Вихід: 1 (1+0+0=1, 1%7=1)
```

Завдання 2.2. Форматування вартості товару

Напишіть програму, яка запитує назву товару, його ціну (дробове число) та кількість одиниць. Програма повинна обчислити загальну вартість і вивести результат у вигляді: "Товар: [назва]. Загальна вартість: [сума] грн.", де сума має бути виведена з двома знаками після коми.

```
Вхід: Яблука, 15.50, 3 → Вихід: Товар: Яблука. Загальна вартість: 46.50 грн.
Вхід: Хліб, 22.0, 1 → Вихід: Товар: Хліб. Загальна вартість: 22.00 грн.
```

Вхід: Вода, 12.8, 5 → Вихід: Товар: Вода. Загальна вартість: 64.00 грн.

Завдання 2.3. Конвертація метрів

Напишіть програму, яка отримує відстань у метрах (дробове число) та виводить її еквівалент у кілометрах і сантиметрах в одному рядку, розділену комами. Виведіть кілометри з двома знаками після коми, сантиметри – як ціле число.

Вхід: 1250.75 → Вихід: 1.25 км, 125075 см
Вхід: 500 → Вихід: 0.50 км, 50000 см
Вхід: 0.01 → Вихід: 0.00 км, 1 см

Завдання 2.4. Порівняння об'єму кубів

Напишіть програму, яка запитує довжину ребра двох кубів (a та b) та визначає:

1. Об'єм якого куба більший (вивести "Перший", "Другий" або "Однакові").

2. У скільки разів об'єм більшого куба перевищує об'єм меншого (вивести число, а якщо об'єми рівні – число 1). Виведіть обидва результати в одному рядку через крапку з комою.

Вхід: 3, 2 → Вихід: Перший; 3.375 (Об'єми: 27 та 8, 27/8=3.375)
Вхід: 2, 2 → Вихід: Однакові; 1
Вхід: 1, 4 → Вихід: Другий; 64.0 (Об'єми: 1 та 64, 64/1=64)

Завдання 2.5. Час у секундах

Напишіть програму, яка запитує кількість годин, хвилин і секунд окремо (цілі числа) і обчислює загальну кількість секунд. Потім запитує додаткову кількість секунд, додає її до загального часу і виводить нову кількість годин, хвилин і секунд у форматі ГГ:ХХ:СС. Врахуйте, що хвилини та секунди завжди мають бути представлені двома цифрами.

Вхід: 1, 30, 45 та додатково 90 → Вихід: 01:32:15 (1 год 30 хв 45 сек = 5445 сек + 90 = 5535 сек = 1 год 32 хв 15 сек)
Вхід: 0, 0, 59 та додатково 1 → Вихід: 00:01:00
Вхід: 12, 0, 0 та додатково 3600 → Вихід: 13:00:00

Частина 3. Складні завдання

Завдання 3.1. Калькулятор калорій для бігу

Напишіть програму для розрахунку витрачених калорій під час бігу. Відомо, що в середньому витрачається 0.75 калорій на кілограм ваги за кожний кілометр бігу. Програма має:

1. Запитати вагу користувача (кг), відстань пробігу (км) та тривалість пробігу (у форматі "хвилини:секунди", наприклад "30:15").

2. Обчислити загальні витрачені калорії.

3. Обчислити середню швидкість бігу в км/год (округлити до 2 знаків).

4. Вивести детальний звіт у форматі з використанням f-рядків:

=== Звіт про пробіг ===

Вага: [вага] кг

Відстань: [відстань] км

Тривалість: [хвилини] хв [секунди] сек

Витрачено калорій: [калорії] ккал

Середня швидкість: [швидкість] км/год

```
Вхід: 70, 5.2, 30:0 →
Вихід:
=== Звіт про пробіг ===
Вага: 70 кг
Відстань: 5.2 км
Тривалість: 30 хв 0 сек
Витрачено калорій: 273.0 ккал (70 * 5.2 * 0.75)
Середня швидкість: 10.4 км/год (5.2 / (30/60))
Вхід: 60, 3.0, 22:30 → Вихід: ... Витрачено калорій: 135.0 ккал, Середня швидкість: 8.0 км/год
```

Завдання 3.2. Калькулятор кінцевої вартості проекту

Студент-фрілансер розраховує вартість проекту. Він працює за ставкою $X за годину. Напишіть програму, яка:

1. Запитує годинну ставку ($), очікувану кількість годин роботи та відсоток податку, який потрібно утримати (напр., 19.5%).

2. Обчислює вартість роботи до оподаткування (ставка × години).

3. Обчислює суму податку та вартість після оподаткування.

4. Виводить три варіанти форматування кінцевої суми в одному рядку через крапку з комою:

a) Без форматування (як є).

b) З двома знаками після коми та знаком долара на початку.

c) З цілою частиною, розділеною пробілами на тисячі та знаком долара в кінці.

```
Вхід: 25, 160, 19.5 →
Вихід: 3220.0; $3220.00; 3 220$
Розрахунок: 25*160=4000; податок=4000*0.195=780; 4000-780=3220
Вхід: 50, 80, 5 → Вихід: 3800.0; $3800.00; 3 800$ (4000-200)
```

Завдання 3.3. Генератор простих арифметичних завдань

Напишіть програму-генератор для молодших школярів, яка:

1. Випадковим чином генерує два цілих числа від 1 до 20 та операцію (+, -, *). (Для простоти в цій версії програма *не* використовує модуль random, а отримує "зерно" від користувача).

2. Запитує у користувача "число-зерно" (ціле).

3. На основі цього числа визначає перший доданок як (зерно % 20) + 1, другий доданок як ((зерно // 10) % 20) + 1, а операцію: + якщо зерно парне, - якщо непарне, але ділиться на 3, * – в інших випадках.

4. Формує завдання у вигляді "Скільки буде [a] [операція] [b]?".

5. Обчислює правильну відповідь та виводить у форматі:

Завдання: Скільки буде 7 + 3?

Правильна відповідь: 10

Вираз для перевірки: 7 + 3 = 10

```
Вхід: 14 → Вихід:
Завдання: Скільки буде 15 + 2? (14%20+1=15,
(14//10)%20+1=1+1=2? Помилка! Давайте виправимо логіку:
(14//10)=1, 1%20=1, +1=2. Але за умовою другий доданок = ((зерно
// 10) % 20) + 1). Операція: 14 - парне, отже '+')
Правильна відповідь: 17
Вираз для перевірки: 15 + 2 = 17
Вхід: 33 → Вихід:
Завдання: Скільки буде 14 - 4? (33%20+1=14,
(33//10)%20+1=3%20+1=4, 33 - непарне і ділиться на 3? 33%3==0,
так, отже '-')
Правильна відповідь: 10
Вираз для перевірки: 14 - 4 = 10
Вхід: 17 → Вихід:
Завдання: Скільки буде 18 * 2? (17%20+1=18,
(17//10)%20+1=1%20+1=2, 17 - непарне, 17%3!=0, отже '*')
Правильна відповідь: 36
Вираз для перевірки: 18 * 2 = 36
```

6. Питання для самоперевірки

1. Який тип даних завжди повертає функція input()? Як отримати з неї ціле число?

2. Назвіть усі арифметичні оператори Python та поясніть різницю між /, // та %.

3. Що повертають оператори порівняння (>, <, ==, тощо`)?

4. У чому різниця між оператором присвоєння = та оператором порівняння ==?

5. Що таке f-рядок (f-string) і для чого він використовується? Наведіть приклад.

6. Поясніть логіку роботи операторів and, or та not. Які значення вони повертають?

7. Дано код: result = 5 + 3 * 2 ** 2. Чому дорівнюватиме значення змінної result? Поясніть порядок виконання операцій.

8. Наведіть приклад виразу на Python, який перевіряє, чи знаходиться число x у діапазоні від -10 до 10 включно.

9. Який результат виведе цей код та чому? print(not (5 > 3) and (10 == 20) or (15 >= 10))

10. Напишіть фрагмент коду, який запитує вік користувача і виводить повідомлення "Ви повнолітній", якщо вік 18 або більше, і "Ви неповнолітній" – якщо ні. (Опишіть логіку, формально використовуючи оператор порівняння та умовний вираз у f-рядку).

11. Як вивести число 123.456789 з допомогою f-рядка, щоб воно мало лише два знаки після коми?

12. Яка помилка є в наступному коді та як її виправити?

```
user_input = input("Введіть число: ")
doubled = user_input * 2
print(doubled)
```

(При введенні 10 програма виводить 1010, а не 20).

13. Програма має обчислити середнє значення трьох чисел. Чи працюватиме цей код коректно для будь-яких чисел? Як можна його покращити?

```
a = input("Введіть a: ")
b = input("Введіть b: ")
c = input("Введіть c: ")
average = (a + b + c) / 3
print(average)
```

14. Що буде виведено в результаті виконання цього коду? Поясніть кожен крок.

```
x = 10
x += 5
x *= 2
x %= 8
print(f"Результат: {x}")
```

15. Користувач ввів рядок "вісім" замість числа 8. Який тип помилки (назва) виникне при спробі конвертувати це в int()? Що має зробити програміст, щоб запобігти або обробити таку ситуацію? (Подумайте про логіку перевірки, хоча технічні засоби обробки будуть вивчені пізніше).

16. Напишіть вираз на Python, який використовує тільки оператори присвоєння та арифметичні оператори, щоб поміняти значення двох цілих змінних a та b місцями **без використання третьої змінної та конструкції** a, b = b, a.

17. Чи можна створити коректний логічний вираз, який використовує лише оператори порівняння та and/or і завжди повертає True? А завжди False? Наведіть приклади.

18. Як би ви за допомогою одного рядка коду (одного виразу) визначили, чи є задане чотиризначне число year (наприклад, 2024) високосним за григоріанським календарем? (Високосний рік: ділиться на 4, але не ділиться на 100, окрім випадків, коли ділиться на 400). Опишіть логічний вираз.

ЛАБОРАТОРНА РОБОТА №3
Умовні конструкції if-elif-else

1. Мета

Оволодіти практичними навичками використання умовних конструкцій if, elif, else у мові Python для розгалуження логіки виконання програм. Розвинути вміння аналізувати умови, правильно формулювати логічні вирази та будувати ефективні гілкові алгоритми для розв'язання типових задач програмування.

2. Завдання

1. Закріпити знання про синтаксис та семантику умовних операторів if, elif, else.
2. Навчитися складати складні логічні вирази з використанням операторів порівняння (==, !=, <, >, <=, >=) та логічних операторів (and, or, not).
3. Опанувати практику вкладених умовних конструкцій.
4. Розв'язати набір задач різного рівня складності, спрямованих на застосування умовних операторів для прийняття рішень у програмі.
5. Навчитися аналізувати коректність роботи умовних блоків та уникати типові помилки (наприклад, неповне покриття умов, неправильний порядок перевірок).

3. Короткі теоретичні відомості

Умовні конструкції – це основа розгалуження логіки в будь-якій мові програмування. Вони дозволяють програмі приймати рішення та виконувати різні дії залежно від виконання певних умов (істинності логічних виразів). У Python для цього використовуються оператори if, elif та else.

Базовий синтаксис:

```
if умова:
    # блок коду, що виконується, якщо умова істинна (True)
```

Умова – це логічний вираз, який повертає True або False. Після умови ставиться двокрапка, а блок коду, що належить умові, виділяється **відступом** (стандартно – 4 пробіли або 1 таб). Це принципова відмінність Python: відступи визначають структуру коду.

Повна форма розгалуження:

```
if умова_1:
    дії_1
elif умова_2:
    дії_2
elif умова_n:
    дії_n
else:
    дії_якщо_жодна_умова_не_виконалася
```

Ключові моменти:

- Оператор if – обов'язковий початок конструкції. Перевіряється першим.
- Оператор elif (скорочення від «else if») – необов'язковий, може бути в будь-якій кількості. Використовується для додаткових перевірок, якщо попередні умови хибні.
- Оператор else – необов'язковий, може бути тільки один в кінці. Виконується, якщо жодна з умов if/elif не була істинною.
- Виконання йде **зверху вниз**. При виконанні першої істинної умови виконується її блок, а решта умов (elif, else) ігноруються.

Задача 1. Проста умова:

```
temperature = 25
if temperature > 20:
    print("Надворі тепло. Одягнись легко.")
```

Задача 2. Повне розгалуження з elif та else:

```
score = 85
if score >= 90:
    print("Оцінка: A")
elif score >= 75:
    print("Оцінка: B")
elif score >= 60:
    print("Оцінка: C")
else:
    print("Оцінка: F (Незадовільно)")
# Виведе: Оцінка: B
```

Логічні оператори:

Для створення складних умов використовуються оператори:

- and (логічне І): істинне, якщо **обидві** частини істинні.
- or (логічне АБО): істинне, якщо **хоча б одна** частина істинна.
- not (логічне НЕ): інвертує значення (True стає False, і навпаки).

Задача 3. Складні умови:

```
age = 20
has_ticket = True
if age >= 18 and has_ticket:
    print("Вхід дозволено на концерт.")
else:
    print("Вхід заборонено.")
```

Вкладені умовні конструкції:

Умовний оператор може містити всередині себе інші умовні оператори, утворюючи вкладену структуру. Тут важливо дотримуватися правильних відступів для кожної нової рівні.

Задача 4. Вкладений if:

```
number = 10
if number > 0:
    print("Число додатне.")
    if number % 2 == 0:
        print("І воно парне.")
    else:
        print("І воно непарне.")
elif number == 0:
    print("Це нуль.")
else:
    print("Число від'ємне.")
```

Важливі зауваження:

1. **Порівняння на рівність** здійснюється оператором ==, а не = (останній – це оператор присвоєння).

2. **Перевірка на входження в діапазон** часто записується ланцюгом порівнянь: if 0 <= x <= 10:.

3. Умови можуть використовувати не лише порівняння, а й результати викликів функцій, методи, які повертають логічні значення, або навіть будь-які значення, які Python інтерпретує як True або False (наприклад, непорожній рядок – True, 0 – False). Проте на початковому етапі краще використовувати явні логічні вирази.

Таблиця 1. Оператори порівняння та логічні оператори

Оператор	Назва	Приклад (де x=True, y=False)	Результат
==	Дорівнює	5 == 5	True
!=	Не дорівнює	5 != 3	True

<	Менше	5 < 3	False
>	Більше	5 > 3	True
<=	Менше або дорівнює	5 <= 5	True
>=	Більше або дорівнює	5 >= 7	False
and	Логічне І	x and y	False
or	Логічне АБО	x or y	True
not	Логічне НЕ	not x	False

4. Методичні рекомендації

Задача 1. Визначення парності числа

Напишіть програму, яка зчитує ціле число та виводить повідомлення, чи є воно парним чи непарним.

```
# Зчитування числа від користувача
number = int(input("Введіть ціле число: "))

# Перевірка на парність (залишок від ділення на 2 дорівнює 0)
if number % 2 == 0:
    print(f"Число {number} є парним.")
else:
    print(f"Число {number} є непарним.")
```

Приклад вхідних та вихідних даних:

```
Вхід: 4 → Вихід: Число 4 є парним.
Вхід: -7 → Вихід: Число -7 є непарним.
Вхід: 0 → Вихід: Число 0 є парним.
```

Коментарі:

Оператор % (модуль) повертає залишок від ділення. Для парних чисел залишок від ділення на 2 завжди дорівнює 0. Важливо враховувати, що 0 також є парним числом.

Задача 2. Знаходження більшого з трьох чисел

Користувач вводить три різних числа. Програма повинна знайти та вивести найбільше з них.

```
# Отримання трьох чисел
a = float(input("Введіть перше число: "))
b = float(input("Введіть друге число: "))
c = float(input("Введіть третє число: "))
```

```
# Порівняння чисел
if a >= b and a >= c:
    largest = a
elif b >= a and b >= c:
    largest = b
else:
    largest = c
print(f"Найбільше число: {largest}")
```

Приклад вхідних та вихідних даних:

Вхід: 5.5, 2.1, 9.3 → Вихід: Найбільше число: 9.3

Вхід: -10, -5, -1 → Вихід: Найбільше число: -1

Вхід: 7, 7, 3 → Вихід: Найбільше число: 7.0 (Через використання float)

Коментарі:

Використано float для коректного порівняння цілих та дробових чисел. Умову a >= b and a >= c можна спростити до a >= b >= c в Python, але перший варіант більш наочний для новачків. Важливо перевіряти не лише >, а й >=, на випадок рівних значень.

Задача 3. Калькулятор операцій

Напишіть простий калькулятор, який зчитує два числа та символ операції (+, -, *, /) і виводить результат. Обробіть можливу помилку ділення на нуль.

```
# Введення даних
num1 = float(input("Введіть перше число: "))
operator = input("Введіть операцію (+, -, *, /): ")
num2 = float(input("Введіть друге число: "))
# Виконання операції відповідно до вибору
if operator == '+':
    result = num1 + num2
    print(f"{num1} {operator} {num2} = {result}")
elif operator == '-':
    result = num1 - num2
    print(f"{num1} {operator} {num2} = {result}")
elif operator == '*':
    result = num1 * num2
    print(f"{num1} {operator} {num2} = {result}")
elif operator == '/':
    if num2 != 0:  # Перевірка дільника
        result = num1 / num2
```

```
            print(f"{num1} {operator} {num2} = {result}")
        else:
            print("Помилка. Ділення на нуль неможливе!")
    else:
        print("Помилка. невідома операція. Використовуйте +, -, *,
/")
```

Приклад вхідних та вихідних даних:

```
Вхід: 10, +, 5 → Вихід: 10.0 + 5.0 = 15.0
Вхід: 8, /, 0 → Вихід: Помилка. Ділення на нуль неможливе!
Вхід: 7, %, 3 → Вихід: Помилка. невідома операція.
Використовуйте +, -, *, /
```

Коментарі:

Важливим елементом є перевірка ділення на нуль за допомогою умови if num2 != 0: всередині гілки для оператора /. Також показана обробка некоректного вводу операції через блок else.

Задача 4. Визначення кварталу за номером місяця

За номером місяця (1-12) визначте, до якого кварталу року він належить. Виведіть номер кварталу (1-4).

```
month = int(input("Введіть номер місяця (1-12): "))

# Перевірка коректності введеного діапазону
if 1 <= month <= 12:
    if 1 <= month <= 3:
        quarter = 1
    elif 4 <= month <= 6:
        quarter = 2
    elif 7 <= month <= 9:
        quarter = 3
    else:  # Місяці 10, 11, 12
        quarter = 4
    print(f"Місяць {month} належить до {quarter}-го
кварталу.")
else:
    print("Помилка. номер місяця повинен бути від 1 до 12!")
```

Приклад вхідних та вихідних даних:

```
Вхід: 5 → Вихід: Місяць 5 належить до 2-го кварталу.
Вхід: 12 → Вихід: Місяць 12 належить до 4-го кварталу.
Вхід: 15 → Вихід: Помилка. номер місяця повинен бути від 1 до 12!
```

Коментарі:

Показано вкладений умовний оператор (if всередині if). Спочатку перевіряється коректність вхідних даних, а потім визначається квартал. Діапазони можна записувати ланцюжком порівнянь: 1 <= month <= 3.

Задача 5. Перевірка на "щасливий" квиток

Шестизначний номер квитка вважається "щасливим", якщо сума перших трьох цифр дорівнює сумі останніх трьох. Перевірте, чи є введений номер таким.

```
ticket_number = input("Введіть шестизначний номер квитка: ")

# Перевірка довжини та що всі символи – цифри
if len(ticket_number) == 6 and ticket_number.isdigit():
    # Конвертація строки в список цифр
    digits = [int(d) for d in ticket_number]

    # Порівняння сум першої та другої половини
    if sum(digits[:3]) == sum(digits[3:]):
        print("Це щасливий квиток!")
    else:
        print("Це звичайний квиток.")
else:
    print("Помилка. введіть рівно 6 цифр!")
```

Приклад вхідних та вихідних даних:

```
Вхід: 123456 → Вихід: Це звичайний квиток. (1+2+3 ≠ 4+5+6)
Вхід: 123420 → Вихід: Це щасливий квиток! (1+2+3 = 4+2+0)
Вхід: 12ab34 → Вихід: Помилка. введіть рівно 6 цифр!
```

Коментарі:

Показана обробка строкового вводу та перевірка його коректності за допомогою методів len() та isdigit(). Використано зрізи списку (digits[:3] та digits[3:]) для поділу номера на частини та функцію sum() для обчислення сум.

Задача 6. Визначення типу трикутника за сторонами

За довжинами трьох сторін визначте тип трикутника: рівносторонній, рівнобедрений, різносторонній або не існує.

```
# Введення довжин сторін
a = float(input("Введіть довжину першої сторони: "))
```

```
b = float(input("Введіть довжину другої сторони: "))
c = float(input("Введіть довжину третьої сторони: "))

# Перевірка на існування трикутника (нерівність трикутника)
if a + b > c and a + c > b and b + c > a:
    # Визначення типу трикутника
    if a == b == c:
        triangle_type = "рівносторонній"
    elif a == b or a == c or b == c:
        triangle_type = "рівнобедрений"
    else:
        triangle_type = "різносторонній"
    print(f"Трикутник існує та є {triangle_type}.")
else:
    print("Трикутник з такими сторонами не існує!")
```

Приклад вхідних та вихідних даних:

```
Вхід: 3, 3, 3 → Вихід: Трикутник існує та є рівносторонній.
Вхід: 5, 5, 7 → Вихід: Трикутник існує та є рівнобедрений.
Вхід: 3, 4, 5 → Вихід: Трикутник існує та є різносторонній.
Вхід: 1, 2, 10 → Вихід: Трикутник з такими сторонами не існує!
```

Коментарі:

Важливий порядок перевірок: спочатку перевіряється можливість існування трикутника (сума будь-яких двох сторін повинна бути більшою за третю), а потім – його тип. Використано ланцюжок порівнянь a == b == c для рівностороннього трикутника.

Задача 7. Конвертер оцінок

Напишіть програму, яка конвертує числову оцінку (0-100) у буквену за американською системою: A (90-100), B (75-89), C (60-74), D (50-59), F (0-49). Якщо оцінка не в діапазоні 0-100, виведіть помилку.

```
score = int(input("Введіть оцінку (0-100): "))
if 0 <= score <= 100:
    if score >= 90:
        grade = "A"
    elif score >= 75:  # Сюди дістанемося тільки якщо score < 90
        grade = "B"
    elif score >= 60:  # Тут score < 75
        grade = "C"
    elif score >= 50:  # Тут score < 60
```

```
            grade = "D"
        else:                # Тут score < 50
            grade = "F"
        print(f"Буквена оцінка: {grade}")
    else:
        print("Помилка. оцінка повинна бути в діапазоні від 0 до
100!")
```

Приклад вхідних та вихідних даних:

```
  Вхід: 95 → Вихід: Буквена оцінка: A
  Вхід: 68 → Вихід: Буквена оцінка: C
  Вхід: 49 → Вихід: Буквена оцінка: F
  Вхід: -5 → Вихід: Помилка. оцінка повинна бути в діапазоні
від 0 до 100!
```

Коментарі:

Показано ефективне використання elif з каскадними перевірками. Оскільки виконання йде зверху вниз, можна не вказувати верхню межу (score <= 89), достатньо нижньої (score >= 75), бо якщо score >= 90, програма б не дійшла до цієї перевірки.

Типові помилки і шляхи їх усунення

1. **Відсутність двокрапки після умови.**
   Причина. Помилка SyntaxError.
   Рішення. Завжди ставити : після if, elif, else.
2. **Неправильні відступи (IndentationError).**
   Причина. Весь код усередині блоку if/elif/else повинен мати однаковий відступ (рекомендується 4 пробіли). IDE та сучасні редактори зазвичай роблять це автоматично.
3. **Плутанина між оператором присвоєння (=) та порівняння (==):**
   = – присвоює значення
   == – перевіряє рівність
4. **Неповне покриття умов.**
   Причина. Коли для деяких вхідних даних програма не видає жодного результату або видає неправильний.
   Рішення. Уважно перевіряйте всі можливі варіанти вхідних даних, використовуйте else для "всіх інших випадків".
5. **Некоректний порядок перевірок у** if-elif.

Причина. Наприклад, якщо спочатку перевіряти score >= 60, а потім score >= 90, то оцінка 95 потрапить у першу ж умову, що некоректно.

Рішення. Завжди починайте з найбільш специфічної умови (з найбільшим обмеженням).

6. **Необроблені некоректні вхідні дані:**

Причина. Користувач може ввести текст замість числа або число поза очікуваним діапазоном.

Рішення. Використовуйте try-except для перехоплення помилок або перевіряйте дані за допомогою умов (як у задачах 4 та 7).

Корисні поради

1. **Протестуйте на крайніх значеннях.** Завжди перевіряйте роботу програми на мінімальних, максимальних та "особливих" значеннях (наприклад, 0, від'ємні числа, порогові значення).

2. **Використовуйте** elif **замість ланцюжка** if. Якщо умови взаємовиключні, об'єднуйте їх в конструкцію if-elif-else. Це підвищує читабельність та ефективність (після виконання однієї умови інші не перевіряються).

3. **Уникайте занадто складних умов.** Якщо умовний вираз стає довшим за 2-3 оператори, розбийте його на частини або використайте вкладені if.

4. **Дотримуйтесь PEP 8.** Імена змінних – з нижнім підкресленням (user_score), оператори оточувати пробілами (a == b), після коми в списку аргументів ставити пробіл.

5. **Пишіть коментарі для неочевидних умов.** Поясніть, чому саме така умова або що означає конкретне порогове значення.

6. **Візуалізуйте логіку.** Для складних умовних конструкцій можна намалювати блок-схему, щоб наочно побачити всі шляхи виконання програми.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Перевірка додатного числа

Напишіть програму, яка перевіряє, чи є введене число додатним. Виведіть "Додатне", якщо так, або "Не додатне" в іншому випадку (включаючи нуль та від'ємні числа).

```
Вхід: 5 → Вихід: Додатне
Вхід: -3 → Вихід: Не додатне
Вхід: 0 → Вихід: Не додатне
```

Завдання 1.2. Подільність на 3

Користувач вводить ціле число. Перевірте, чи ділиться воно на 3 без остачі. Виведіть "Ділиться", якщо так, або "Не ділиться", якщо ні.

```
Вхід: 9 → Вихід: Ділиться
Вхід: 10 → Вихід: Не ділиться
Вхід: -6 → Вихід: Ділиться
```

Завдання 1.3. Однакова парність

Дано два цілих числа. Перевірте, чи є вони обидва парними або обидва непарними. Виведіть "Однакова парність", якщо так, або "Різна парність", якщо ні.

```
Вхід: 4, 10 → Вихід: Однакова парність
Вхід: 3, 7 → Вихід: Однакова парність
Вхід: 2, 5 → Вихід: Різна парність
```

Завдання 1.4. Максимум двох чисел

Знайдіть більше з двох введених чисел. Якщо числа рівні, виведіть будь-яке з них.

```
Вхід: 14, 25 → Вихід: 25
Вхід: -5, -10 → Вихід: -5
Вхід: 7, 7 → Вихід: 7
```

Завдання 1.5. Доступ до контенту

Програма запитує вік користувача. Якщо вік менше 18, виведіть "Доступ заборонено", інакше - "Доступ дозволено".

```
Вхід: 15 → Вихід: Доступ заборонено
Вхід: 21 → Вихід: Доступ дозволено
Вхід: 18 → Вихід: Доступ дозволено
```

Завдання 1.6. Тернарний оператор (простий варіант)

Використовуючи умовний вираз (тернарний оператор), перевірте, чи є число кратним 5. Виведіть "Кратне 5" або "Не кратне 5".

```
Вхід: 20 → Вихід: Кратне 5
Вхід: 13 → Вихід: Не кратне 5
Вхід: 0 → Вихід: Кратне 5
```

Завдання 1.7. Визначення знаку числа

Визначте знак введеного числа. Виведіть "Додатне", "Від'ємне" або "Нуль".

```
Вхід: 3.5 → Вихід: Додатне
Вхід: -8 → Вихід: Від'ємне
Вхід: 0 → Вихід: Нуль
```

Завдання 1.8. Сезон за номером місяця

За номером місяця (1-12) визначте сезон: зима (12, 1, 2), весна (3-5), літо (6-8), осінь (9-11). Якщо номер не в діапазоні 1-12, виведіть "Помилка".

```
Вхід: 7 → Вихід: Літо
Вхід: 2 → Вихід: Зима
Вхід: 13 → Вихід: Помилка
```

Завдання 1.9. Оцінка за балом

Студент отримав бал від 0 до 100. Виведіть оцінку: "Відмінно" (90-100), "Добре" (75-89), "Задовільно" (60-74), "Незадовільно" (0-59). Якщо бал поза діапазоном, виведіть "Помилка".

```
Вхід: 95 → Вихід: Відмінно
Вхід: 68 → Вихід: Задовільно
Вхід: 105 → Вихід: Помилка
```

Завдання 1.10. День тижня

Користувач вводить номер дня тижня (1-7, де 1 - понеділок). Виведіть назву дня. Якщо номер не в діапазоні, виведіть "Невірний номер".

```
Вхід: 3 → Вихід: Середа
Вхід: 6 → Вихід: Субота
Вхід: 0 → Вихід: Невірний номер
```

Частина 2. Середні завдання

Завдання 2.1. Визначення типу року

Напишіть програму, яка визначає, чи є рік високосним. Рік високосний, якщо він ділиться на 4, але не ділиться на 100, або ділиться на 400. Виведіть "Високосний" або "Не високосний".

```
Вхід: 2024 → Вихід: Високосний
Вхід: 1900 → Вихід: Не високосний
Вхід: 2000 → Вихід: Високосний
```

Завдання 2.2. Розв'язання квадратного рівняння

Дано коефіцієнти a, b, c квадратного рівняння $ax^2 + bx + c = 0$. Визначте кількість розв'язків: 0 (дискримінант < 0), 1 (дискримінант = 0), 2 (дискримінант > 0). Обробіть випадок, коли a = 0 (рівняння не квадратне).

```
Вхід: 1, -3, 2 → Вихід: 2 розв'язки
Вхід: 1, 2, 1 → Вихід: 1 розв'язок
Вхід: 0, 5, 10 → Вихід: Рівняння не квадратне
```

Завдання 2.3. Перевірка трикутника з подальшим визначенням типу

За трьома сторонами спочатку перевірте, чи може трикутник існувати (кожна сторона менша за суму двох інших). Якщо так, визначте його тип: гострокутний (квадрат найбільшої сторони < суми квадратів двох інших), прямокутний (дорівнює), тупокутний (більше).

```
Вхід: 3, 4, 5 → Вихід: Прямокутний трикутник
Вхід: 5, 5, 8 → Вихід: Тупокутний трикутник
Вхід: 1, 2, 10 → Вихід: Трикутник не існує
```

Завдання 2.4. Валідатор пароля

Користувач вводить пароль. Перевірте його на відповідність вимогам: мінімум 8 символів, містить хоча б одну цифру, хоча б одну велику літеру. Виведіть відповідне повідомлення: "Пароль прийнято" або перелік порушених вимог.

```
Вхід: Pass1234 → Вихід: Пароль прийнято
Вхід: abc → Вихід: Занадто короткий, немає цифри, немає
великої літери
Вхід: Password → Вихід: Немає цифри
```

Завдання 2.5. Конвертер часового формату

Користувач вводить час у 24-годинному форматі (години та хвилини). Переведіть його в 12-годинний формат з позначками "AM" (до полудня) та "PM" (після полудня). Обробіть некоректний ввід (години поза 0-23, хвилини поза 0-59).

```
Вхід: 14, 30 → Вихід: 2:30 PM
Вхід: 8, 15 → Вихід: 8:15 AM
Вхід: 25, 70 → Вихід: Некоректний час
```

Частина 3. Складні завдання

Завдання 3.1. Система знижок для інтернет-магазину

Розрахуйте підсумкову суму покупки з урахуванням знижок. Вхідні дані: сума покупки, статус клієнта ("новий", "постійний", "VIP"), наявність промокоду ("SALE10", "SALE20", нічого). Знижки накладаються послідовно:

1. Статус: новий - 5%, постійний - 10%, VIP - 15%
2. Промокод: SALE10 - 10%, SALE20 - 20%
3. При сумі покупки > 1000 грн - додатково 5%
4. Максимальна знижка не може перевищувати 40%. Виведіть кінцеву суму.

```
Вхід: 1200, "постійний", "SALE20" → Вихід: 1200 → 1080 → 864 →
820.8 (остаточна сума)
Вхід: 500, "новий", "" → Вихід: 500 → 475 → 475 (без змін)
Вхід: 2000, "VIP", "SALE10" → Вихід: 2000 → 1700 → 1530 →
1453.5 → 1453.5 (обмеження 40%)
```

Завдання 3.2. Калькулятор податків для фрілансера

Фрілансер вводить свій річний дохід. Розрахуйте суму податку за прогресивною шкалою:

- До 20 000 грн: 5%
- 20 001 - 50 000 грн: 10% з суми понад 20 000
- 50 001 - 100 000 грн: 15% з суми понад 50 000
- Понад 100 000 грн: 20% з суми понад 100 000
- Додатково: якщо дохід > 30 000 грн і є 1 дитина - знижка 2% від загального податку, 2+ дітей - знижка 5%. Мінімальний податок - 100 грн. Виведіть суму податку.

```
Вхід: 15000, 0 → Вихід: 750.0
```

```
Вхід: 75000, 2 → Вихід: 20000*0.05 + 30000*0.1 + 25000*0.15 =
6250 → 6250*0.95 = 5937.5
Вхід: 120000, 1 → Вихід: 20000*0.05 + 30000*0.1 + 50000*0.15 +
20000*0.2 = 15500 → 15500*0.98 = 15190.0
```

Завдання 3.3. Система прийому замовлень в ресторані

Програма приймає замовлення з наступними параметрами:

1. Час доби: "ранок" (8-11), "обід" (12-16), "вечір" (17-23), "ніч" (0-7) - інші значення недійсні

2. Тип замовлення: "сніданок", "обід", "вечеря", "доставка"

3. Кількість персон (1-10)

4. Правила:

- Сніданок доступний лише вранці, обід - в обідній час, вечеря - ввечері, доставка - будь-коли
- Для доставки: +20% до суми, мінімальна сума замовлення 200 грн
- Кожна персона: базово 100 грн за обід/вечерю, 80 грн за сніданок
- Знижка 10% для замовлень > 500 грн
- Для нічного часу (0-7) +30% націнка
- Виведіть підсумкову суму або повідомлення про неможливість замовлення.

```
Вхід: "ранок", "сніданок", 3 → Вихід: 3 * 80 = 240 грн
Вхід: "ніч", "доставка", 2 → Вихід: 2 * 100 = 200 → +20% = 240
→ +30% = 312 грн
Вхід: "обід", "сніданок", 4 → Вихід: Неможливе замовлення:
сніданок недоступний в обідній час
```

6. Питання для самоперевірки

1. Який синтаксис базової умовної конструкції if в Python?

2. Для чого використовується ключове слово elif і чим воно відрізняється від послідовності незалежних операторів if?

3. Яка роль ключового слова else в умовних конструкціях?

4. Які значення в Python інтерпретуються як False в умовних виразах (хибні значення)?

5. Чому після умови в операторах if, elif, else ставиться двокрапка?

6. Як Python визначає, який код належить до блоку if? Що таке відступи (indentation) і які є вимоги до них?

7. Чи можна мати кілька блоків else в одній умовній конструкції? А кілька блоків elif?

8. Що буде, якщо після if написати умову, що завжди істинна? А завжди хибна?

9. Перерахуйте всі оператори порівняння в Python. Який результат їх роботи?

10. Як працюють логічні оператори and, or, not? Який їх пріоритет виконання?

11. Що таке ланцюжкове порівняння (chained comparison) у Python? Наведіть приклад.

12. Яка різниця між операторами = та ==? Чи можна використовувати = в умові?

13. Як правильно організувати перевірку на входження числа в певний діапазон (наприклад, від 0 до 100)?

14. Як обробляти ситуацію, коли потрібно перевірити багато взаємовиключних умов?

15. Що таке вкладені умовні конструкції та коли їх доцільно використовувати?

16. Як уникнути помилки ділення на нуль за допомогою умовних операторів?

17. Яка помилка виникне, якщо забути двокрапку після умови? А якщо зробити неправильний відступ?

18. Чому важливо враховувати порядок перевірки умов у конструкції if-elif-elif-else?

19. Що таке "неповне покриття умов" і як його уникнути?

20. Які є способи перевірки коректності вхідних даних перед виконанням умовних операцій?

21. Що таке тернарний умовний оператор в Python? Наведіть приклад його використання.

22. Чи можна використовувати умовні вирази у списках (list comprehensions)? Як?

23. Як умовні конструкції пов’язані з булевою алгеброю? Наведіть приклад спрощення складного логічного виразу.

24. Що таке "коротке замикання" (short-circuit evaluation) при обчисленні логічних виразів та яке його практичне значення?

25. Коли краще використовувати множинні if замість if-elif-else?

26. Як можна реалізувати "перемикач" (switch-case), якого немає в Python, за допомогою умовних конструкцій?

27. Чи є обмеження на глибину вкладеності умовних конструкцій? Чому важливо уникати надмірної вкладеності?

28. Як умовні конструкції сприяють створенню більш гнучких та адаптивних програм?

29. Що виведе цей код при x = 5? Чому?

```
if x > 0:
    print("A")
if x > 3:
    print("B")
else:
    print("C")
```

30. Який з цих двох варіантів ефективніший і чому?

- Варіант 1:

```
if score >= 90:
    grade = "A"
if score >= 80 and score < 90:
    grade = "B"
```

Варіант 2:

```
if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
```

ЛАБОРАТОРНА РОБОТА №4
Цикли for та while

1. Мета

Освоєння базових концепцій такуального програмування на Python через практичне використання операторів циклів for та while; формування вміння аналізувати умову задачі та вибирати оптимальний тип циклу для її розв'язання; розвиток навичок написання, тестування та налагодження програм з циклами.

2. Завдання

1. Закріпити теоретичні знання щодо синтаксису та семантики циклів for та while.
2. Навчитися визначати необхідність застосування того чи іншого типу циклу залежно від умови задачі.
3. Опанувати методи керування виконанням циклів за допомогою операторів break, continue та else.
4. Набути практичного досвіду створення програм для обробки послідовностей чисел, рядків та розв'язання обчислювальних задач.
5. Розвинути вміння аналізувати та усувати типові помилки, пов'язані з нескінченними циклами та логікою ітерацій.

3. Короткі теоретичні відомості

Цикл – це базова конструкція програмування, що дозволяє багаторазово виконувати один і той же блок інструкцій (тіло циклу) доти, доки виконується певна умова.

1. Цикл while (цикл з передумовою)

Цикл while повторює виконання блоку коду, поки задана логічна умова є істинною (True). Він ідеальний для ситуацій, коли кількість ітерацій заздалегідь невідома і залежить від динамічної умови.

Синтаксис:

```
while умова:
    # тіло циклу
    інструкція_1
    інструкція_2
```

Задача 1. Підрахунок суми цифр числа.

```
number = 3047
sum_of_digits = 0
while number > 0:
    digit = number % 10  # отримуємо останню цифру
    sum_of_digits += digit
    number = number // 10  # видаляємо останню цифру
print("Сума цифр:", sum_of_digits)  # Виведе: 14
```

У цьому прикладі кількість ітерацій залежить від початкового значення number і дорівнює кількості цифр у ньому.

Увага! Нескінченний цикл. Якщо умова циклу while завжди істинна, цикл стане нескінченним. Для безпечного завершення можна використовувати оператор break.

2. Цикл for **(цикл з лічильником або цикл по колекції)**

Цикл for в Python призначений для ітерації (перебору) елементів *ітерованого об'єкта* (iterable), наприклад, рядка, списку, або діапазону чисел, створеного функцією range(). Він часто використовується, коли кількість повторень відома або визначається довжиною послідовності.

Синтаксис:

```
for змінна in ітерований_об'єкт:
    # тіло циклу
    інструкція_1
    інструкція_2
```

Задача 2. Виведення всіх символів рядка та підрахунок чисел у діапазоні.

```
# Ітерація по рядку
for letter in "Python":
    print(letter)  # Виведе кожну літеру в новому рядку: P, y, t, h, o, n
# Ітерація за допомогою range()
sum_numbers = 0
for i in range(1, 11):  # числа від 1 до 10 включно
    sum_numbers += i
print("Сума чисел від 1 до 10:", sum_numbers)  # Виведе: 55
```

Функція range(start, stop, step) генерує послідовність чисел:

- range(5) -> 0, 1, 2, 3, 4
- range(2, 8) -> 2, 3, 4, 5, 6, 7
- range(0, 10, 2) -> 0, 2, 4, 6, 8

3. Оператори керування циклом: break, continue, else

- break – негайно завершує виконання поточного циклу (найближчого, в якому він знаходиться). Керування передається наступному після циклу оператору.
- continue – перериває поточну ітерацію циклу і переходить до наступної, пропускаючи решту коду в тілі циклу для поточної ітерації.
- else – блок else, вказаний після циклу (for або while), виконується **тільки тоді**, коли цикл завершився природним чином, тобто не був перерваний оператором break.

Задача 3. Пошук простих чисел з використанням break та else.

```
for num in range(2, 10):
    for divisor in range(2, num):
        if num % divisor == 0:  # знайдено дільник
            print(f"{num}       дорівнює       {divisor}       *
{num//divisor}")
            break  # вихід з внутрішнього циклу
    else:
        # Цей блок виконається, якщо внутрішній цикл не зустрів
break
        print(f"{num} - просте число")
```

Вивід:

```
2 - просте число
3 - просте число
4 дорівнює 2 * 2
5 - просте число
6 дорівнює 2 * 3
7 - просте число
8 дорівнює 2 * 4
9 дорівнює 3 * 3
```

4. Вибір між for та while

Критерій	**Цикл** for	**Цикл** while
Основне призначення	Ітерація по відомій послідовності або фіксовану кількість разів.	Повторення дій, поки виконується динамічна умова.

Кількість ітерацій	Зазвичай відома заздалегідь (довжина послідовності).	Часто невідома заздалегідь, залежить від умови.
Типова структура	for елемент in послідовність:	while умова:

Вкладена умова в циклах та оператор elif

Під час роботи з циклами часто виникає потреба перевірити кілька взаємовиключних умов всередині однієї ітерації. Для цього, окрім базових if та else, використовується оператор elif (скорочення від "else if").

Призначення. Оператор elif дозволяє послідовно перевіряти низку умов. Виконання блоку elif відбувається тільки якщо всі попередні умови (if та інші elif) були хибними (False), а його власна умова – істинна (True). Після виконання першого ж блоку з істинною умовою (будь то if чи elif), решта конструкції if-elif-else ігнорується.

Синтаксис всередині циклу:

```
for element in sequence:
    # ... деякий код ...
    if умова_1:
        # виконується, якщо умова_1 істинна
    elif умова_2:
        # виконується, якщо умова_1 хибна, а умова_2 істинна
    elif умова_3:
        # виконується, якщо умова_1 та умова_2 хибні, а умова_3 істинна
    else:
        # виконується, якщо всі умови вище (if та elif) хибні
```

Задача 4. Класифікація чисел у послідовності за допомогою for та elif.

```
numbers = [12, -5, 0, 7, -2, 0, 9]
positive_count = 0
negative_count = 0
zero_count = 0

for num in numbers:
    if num > 0:
        positive_count += 1
        print(f"{num} - додатнє число")
    elif num < 0:
```

```
            negative_count += 1
            print(f"{num} – від'ємне число")
        else: # num == 0
            zero_count += 1
            print(f"{num} – нуль")

    print(f"\nПідсумок: Додатних: {positive_count}, Від'ємних:
{negative_count}, Нулів: {zero_count}")
```

Пояснення. Для кожного числа num у списку спочатку перевіряється, чи воно більше 0. Якщо так, лічильник positive_count збільшується, і подальші перевірки (elif та else) пропускаються. Якщо перша умова хибна (число не більше 0), програма переходить до elif num < 0. Якщо і ця умова хибна, виконується блок else, що означає, що число дорівнює 0.

Ключові моменти:

1. elif завжди йде після if та перед else.
2. Кількість elif може бути будь-якою.
3. else не є обов'язковим, на відміну від if.
4. Використання elif є набагато більш ефективним та читабельним, ніж написання ряду незалежних операторів if, оскільки забезпечує взаємовиключність умов.

Цей механізм є незамінним при розгалуженні логіки всередині циклу для категорізації даних, обробки різних випадків або створення меню на основі введення користувача в циклі while.

4. Методичні рекомендації

Задача 1. Підрахунок парних та непарних чисел у діапазоні

Напишіть програму, яка приймає два цілих числа a та b (де a < b) і для діапазону від a до b включно підраховує та виводить кількість парних та непарних чисел.

```
# Отримання меж діапазону від користувача
a = int(input("Введіть початок діапазону (a): "))
b = int(input("Введіть кінець діапазону (b): "))

# Ініціалізація лічильників
even_count = 0  # лічильник парних чисел
odd_count = 0   # лічильник непарних чисел

# Проходимо циклом по всіх числах діапазону включно
for number in range(a, b + 1):
    # Перевіряємо парність за допомогою оператора залишку від ділення (%)
    if number % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

# Вивід результатів
print(f"У діапазоні від {a} до {b}:")
print(f"Парних чисел: {even_count}")
print(f"Непарних чисел: {odd_count}")
```

Приклад вхідних та вихідних даних:

```
Вхід: a = 1, b = 5 → Вихід: Парних чисел: 2, Непарних чисел: 3
Вхід: a = 10, b = 10 → Вихід: Парних чисел: 1, Непарних чисел: 0
Вхід: a = -3, b = 2 → Вихід: Парних чисел: 3, Непарних чисел: 3
```

Коментарі:

Ключова функція для створення послідовності – range(a, b + 1). Важливо пам'ятати, що range не включає праву межу, тому потрібно додавати +1.

Для перевірки парності використовується оператор % (залишок від ділення). Парне число ділиться на 2 без залишку (number % 2 == 0).

Це типова задача на ітерацію з лічильником, де for є ідеальним вибором, оскільки кількість ітерацій відома (дорівнює b - a + 1).

Задача 2. Пошук факторіалу числа

Напишіть програму, яка обчислює факторіал заданого натурального числа N. Факторіал N! = 1 * 2 * 3 * ... * N. Прийнято, що 0! = 1.

```
# Отримання числа від користувача
n = int(input("Введіть натуральне число N: "))

# Ініціалізація змінної для збереження результату
factorial = 1

# Обчислення факторіалу за допомогою циклу for
for i in range(1, n + 1):
    factorial *= i  # Скорочений запис: factorial = factorial * i


# Вивід результату
print(f"Факторіал числа {n}! = {factorial}")
```

Приклад вхідних та вихідних даних:

```
Вхід: N = 5 → Вихід: Факторіал числа 5! = 120
Вхід: N = 1 → Вихід: Факторіал числа 1! = 1
Вхід: N = 0 → Вихід: Факторіал числа 0! = 1
```

Коментарі:

Це класичний приклад циклу з накопиченням результату (акмулятором). Змінна factorial ініціалізується одиницею, оскільки множення на 1 не змінює результат.

Важливо правильно задати діапазон range(1, n + 1), щоб у випадку n = 0 цикл не виконався жодного разу (діапазон range(1, 1) – порожній), і результатом залишиться 1, що відповідає математичному визначенню.

Задача 3. Симуляція банкомату з контролем спроб вводу PIN-коду

Напишіть програму, яка імітує перевірку PIN-коду в банкоматі. Користувач має 3 спроби для введення правильного PIN-коду (заданий у програмі як correct_pin = "1234"). При введенні неправильного коду виводиться попередження. При вичерпанні спроб або успішному вводі програма завершується з відповідним повідомленням.

```
# Заданий правильний PIN-код
correct_pin = "1234"
attempts_left = 3  # Лічильник доступних спроб
```

```
    # Цикл while: працює, поки є спроби
    while attempts_left > 0:
        user_pin = input(f"Введіть PIN-код (залишилось спроб: {attempts_left}): ")

        if user_pin == correct_pin:
            print("PIN-код вірний! Доступ надано.")
            break  # Успіх, виходимо з циклу достроково
        else:
            attempts_left -= 1  # Зменшуємо кількість спроб
            if attempts_left > 0:
                print("Невірний PIN-код. Спробуйте ще раз.")
            else:
                print("Невірний PIN-код. Карту заблоковано.")

    # Цей блок виконається, тільки якщо цикл НЕ був перерваний break
    else:
        print("Увага: Програма завершена через вичерпання спроб.")
```

Приклад вхідних та вихідних даних:

Вхід: "1111", "2222", "1234" → Вихід: "Невірний PIN...", "Невірний PIN...", "PIN-код вірний! Доступ надано."

Вхід: "1234" (з першої спроби) → Вихід: "PIN-код вірний! Доступ надано."

Вхід: "1111", "2222", "3333" → Вихід: "Невірний PIN...", "Невірний PIN...", "Невірний PIN-код. Карту заблоковано." (після цього виконається блок else циклу).

Коментарі:

Ідеальний приклад для while, оскільки кількість ітерацій залежить від дій користувача.

Використання break для миттєвого виходу при успішній автентифікації.

Демонструє використання блоку else для циклу while, який виконується, якщо умова циклу стала хибною (attempts_left > 0), але не було дострокового виходу через break.

Задача 4. Генерація послідовності Фібоначчі

Напишіть програму, яка виводить перші N чисел послідовності Фібоначчі, де кожне наступне число є сумою двох попередніх. Починається послідовність з 0 та 1.

```
# Отримуємо кількість чисел для виведення
n = int(input("Скільки чисел Фібоначчі вивести? "))

# Ініціалізація перших двох чисел
a, b = 0, 1
count = 0

# Виконуємо цикл, поки не виведемо n чисел
while count < n:
    print(a, end=" ")  # Виводимо поточне число в рядок
    # Обчислюємо наступне число
    a, b = b, a + b    # Множинне присвоювання – ключова операція
    count += 1

print()  # Перехід на новий рядок після завершення виводу
```

Приклад вхідних та вихідних даних:

```
Вхід: N = 7 → Вихід: 0 1 1 2 3 5 8
Вхід: N = 1 → Вихід: 0
Вхід: N = 10 → Вихід: 0 1 1 2 3 5 8 13 21 34
```

Коментарі:

Задача демонструє елегантне оновлення змінних за допомогою множинного присвоювання: a, b = b, a + b. Спочатку обчислюються вирази праворуч (b та a + b), а потім їх значення присвоюються змінним a та b ліворуч.

Цикл while тут зручніший за for, хоча кількість ітерацій відома, оскільки логіка оновлення чисел не залежить від лічильника і в циклі for.

Параметр end=" " у функції print() забезпечує виведення чисел в один рядок через пробіл.

Типові помилки і шляхи їх усунення

1. Нескінченний цикл while.

Ознака. Програма "зависає" або безкінечно щось виводить.

Причина. Забули змінити змінну, яка входить в умову циклу, всередині тіла циклу.

Рішення. Переконайтеся, що в тілі циклу є операція, яка в кінцевому підсумку зробить умову хибною (наприклад, збільшення лічильника, зменшення кількості спроб).

2. Помилки з функцією range().

"Пропуск" останнього елемента: for i in range(a, b): – цикл виконається для значень від a до b-1. Для включення b потрібно писати range(a, b + 1).

Порожній цикл при a >= b: Наприклад, range(5, 1). Для зворотного порядку використовуйте range(5, 1, -1).

3. Плутанина між break та continue:

break – повністю виходить з циклу.

continue – перескакує на наступну ітерацію поточного циклу, пропускаючи код після себе.

Рішення. Уявіть break як "СТОП, цикл закінчено", а continue – як "ПЕРЕЙТИ ДАЛІ".

4. Зміна ітерованої послідовності всередині циклу for.

Небезпечний код: for item in my_list: my_list.remove(item)

Проблема. Це може призвести до непередбачуваної поведінки, оскільки Python внутрішньо використовує індекс для ітерації.

Рішення. Ітеруйте за копією списку: for item in my_list.copy(): або створюйте новий список для результатів.

5. Відсутня ініціалізація змінних-лічильників або акумуляторів.

Помилка. sum = sum + number (де sum не визначено до циклу).

Рішення. Завжди ініціалізуйте змінні перед циклом (наприклад, sum = 0, product = 1).

Корисні поради

1. **Перед написанням коду проговоріть/запишіть логіку алгоритму словами або у вигляді блок-схеми.** Це допоможе визначити, який цикл (for чи while) підходить краще.

2. **Використовуйте зрозумілі назви змінних.** Замість i, j, x пишіть counter, attempts_left, current_number. Це робить код самодокументованим.

3. **Декомпозиція складних задач.** Якщо задача складна (наприклад, виведення піраміди з чисел), розбийте її на підзадачі: 1) цикл для рядків, 2) вкладений цикл для пробілів, 3) вкладений цикл для чисел.

4. **Відлагоджуйте за допомогою виведення проміжних значень.** Всередині циклу виводьте значення ключових змінних, щоб відстежити, як змінюється стан програми на кожній ітерації.

5. **Тестуйте на крайових випадках.** Завжди перевіряйте, як працює ваша програма при N = 0, N = 1, при від'ємних числах, порожніх введеннях тощо. Це допомагає виявити логічні помилки.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Сума чисел у заданому діапазоні

Напишіть програму, яка обчислює суму всіх цілих чисел у діапазоні від N до M включно. Значення N та M вводяться користувачем (N ≤ M).

```
Вхід: N=1, M=5 → Вихід: 15
Вхід: N=10, M=10 → Вихід: 10
Вхід: N=-3, M=2 → Вихід: -3 (-3 + -2 + -1 + 0 + 1 + 2)
```

Завдання 1.2. Добуток парних чисел

Обчисліть добуток всіх парних чисел від 2 до заданого числа K включно.

```
Вхід: K=6 → Вихід: 48 (2*4*6)
Вхід: K=1 → Вихід: 1
Вхід: K=10 → Вихід: 3840 (2*4*6*8*10)
```

Завдання 1.3. Виведення квадратів чисел

Виведіть квадрати всіх чисел від a до b у зворотному порядку.

```
Вхід: a=2, b=4 → Вихід: 16 9 4
Вхід: a=-1, b=1 → Вихід: 1 0 1
Вхід: a=5, b=5 → Вихід: 25
```

Завдання 1.4. Підрахунок чисел, кратних 7

Підрахуйте кількість чисел у діапазоні від start до end, які діляться націло на 7.

```
Вхід: start=1, end=20 → Вихід: 2 (7, 14)
Вхід: start=7, end=7 → Вихід: 1
Вхід: start=50, end=60 → Вихід: 0
```

Завдання 1.5. Форматування виведення таблиці

Виведіть таблицю множення на задане число n (від 1 до 10) у форматі: n x i = результат.

```
Вхід: n=5 → Вихід:
5 x 1 = 5
5 x 2 = 10
...
5 x 10 = 50
```

Завдання 1.6. Піднесення до степеня циклом while

Реалізуйте піднесення числа base до степеня exp (exp - ціле невід'ємне) за допомогою циклу while (не використовуйте оператор **).

```
Вхід: base=2, exp=3 → Вихід: 8
Вхід: base=5, exp=0 → Вихід: 1
Вхід: base=3, exp=4 → Вихід: 81
```

Завдання 1.7. Пошук першого числа, більшого за поріг

Знайдіть перше натуральне число, квадрат якого перевищує задане число limit. Використайте цикл while.

```
Вхід: limit=10 → Вихід: 4 (4²=16 > 10)
Вхід: limit=100 → Вихід: 11 (11²=121 > 100)
Вхід: limit=0 → Вихід: 1
```

Завдання 1.8. Сума цифр числа while

Обчисліть суму цифр заданого цілого додатного числа за допомогою циклу while.

```
Вхід: 123 → Вихід: 6
Вхід: 1000 → Вихід: 1
Вхід: 987654 → Вихід: 39
```

Завдання 1.9. Сума послідовності з дробів

Обчисліть суму ряду: 1/1 + 1/2 + 1/3 + ... + 1/N для заданого N.

```
Вхід: N=3 → Вихід: 1.8333 (приблизно)
Вхід: N=1 → Вихід: 1.0
Вхід: N=5 → Вихід: 2.2833 (приблизно)
```

Завдання 1.10. Добуток непарних чисел з перевіркою

Обчисліть добуток всіх непарних чисел від 1 до N, але якщо зустрічається число 13, припиніть множення (не включаючи 13).

```
Вхід: N=7 → Вихід: 105 (1*3*5*7)
Вхід: N=15 → Вихід: 10395 (1*3*5*7*9*11)
Вхід: N=5 → Вихід: 15
```

Частина 2. Середні завдання

Завдання 2.1. Симулятор кидків кубика з перемогою

Симулюйте кидки шестигранного кубика (генеруйте випадкові числа від 1 до 6). Кидки продовжуються, доки не випаде 6 або не буде зроблено 10 кидків. Виведіть кількість кидків та суму всіх випалих чисел.

```
Приклад виконання 1: Випали числа: 2, 4, 1, 6 → Вихід:
Кількість кидків: 4, Сума: 13
Приклад виконання 2: 10 кидків без шістки: 1,2,3,4,5,1,2,3,4,5
→ Вихід: Кількість кидків: 10, Сума: 30
Приклад виконання 3: Перший кидок 6 → Вихід: Кількість кидків:
1, Сума: 6
```

Завдання 2.2. Фільтрація введених чисел

Програма зчитує числа від користувача до введення 0. Виведіть: 1) кількість додатних чисел; 2) максимальне від'ємне число (якщо такі є); 3) суму чисел, які закінчуються на 3.

```
Вхід: 4, -2, 13, -5, 3, 0 → Вихід: Додатних: 3, Макс від'ємне:
-2, Сума чисел що закінчуються на 3: 16
Вхід: 0 → Вихід: Додатних: 0, Макс від'ємне: не введено, Сума
чисел що закінчуються на 3: 0
Вхід: -10, -3, 23, -1, 0 → Вихід: Додатних: 1, Макс від'ємне:
-1, Сума: 20
```

Завдання 2.3. Генератор "ступінчастої" послідовності

Згенеруйте послідовність, де кожне наступне число утворюється додаванням до попереднього його останньої цифри. Починаємо з start. Зупиняємося, коли число перевищує limit.

```
Вхід: start=3, limit=20 → Вихід: 3 6 12 14 18
Вхід: start=1, limit=100 → Вихід: 1 2 4 8 16 22 24 28 36 42 44
48 56 62 64 68 76 82 84 88 96
Вхід: start=9, limit=10 → Вихід: 9
```

Завдання 2.4. Пошук "найсиметричнішого" числа в діапазоні

Знайдіть число в діапазоні [a, b], у якого сума цифр максимальна. Якщо таких чисел декілька, виведіть найменше з них.

```
Вхід: a=10, b=20 → Вихід: 19 (сума цифр = 10)
Вхід: a=1, b=100 → Вихід: 99 (сума цифр = 18)
Вхід: a=123, b=130 → Вихід: 129 (сума цифр = 12)
```

Завдання 2.5. Аналіз часових інтервалів

Користувач вводить кількість секунд total_seconds. Перетворіть її у формат "дні:години:хвилини:секунди" за допомогою циклів (не використовуючи оператор // для цілочисельного ділення між одиницями).

```
Вхід: 3661 → Вихід: 0:1:1:1
Вхід: 100000 → Вихід: 1:3:46:40
Вхід: 59 → Вихід: 0:0:0:59
```

Частина 3. Складні завдання

Завдання 3.1. Шифр "подвійного зсуву"

Реалізуйте шифрування рядка. Кожна літера зміщується на позицію її індексу в алфавіті (a=0, b=1... z=25), а потім результат зсувається ще раз на довжину всього слова. Неалфавітні символи залишаються без змін. Використовуйте цикл для перебору символів та вкладений цикл для пошуку в алфавіті.

```
Вхід: "abc" → Вихід: "def" (a→(0+3)=d, b→(1+3)=e, c→(2+3)=f)
Вхід: "test" → Вихід: "xiwx"
Вхід: "hello world!" → Вихід: "mjqqt btwqi!"
```

Завдання 3.2. Оптимізація маршруту доставки

Є список відстаней між точками: [10, 5, 8, 12, 3]. Це відстані між сусідніми пунктами (A-B, B-C, C-D, D-E, E-A, формується цикл). Знайдіть:

1. Найдовшу ділянку маршруту
2. Середню відстань між пунктами
3. Відстань повного кола (сума всіх)
4. Мінімальну відстань між будь-якими двома несусідніми пунктами (напряму через одну)
5. Використайте лише один прохід циклу для обчислення всіх цих значень.

```
Вхід: [10, 5, 8, 12, 3] → Вихід:
Найдовша ділянка: 12
Середня відстань: 7.6
Повна відстань кола: 38
Мін відстань через одну: 13
```

Завдання 3.3. Використання AI для оптимізації коду циклів

Частина А (5 балів): Напишіть наївну реалізацію програми, яка знаходить усі "щасливі" числа в діапазоні від 1 до N. "Щасливим" вважається число, у якому сума квадратів цифр дорівнює 1 або, повторно застосовуючи цю операцію до результату, можна дійти до 1.

Частина Б (8.3 бали): Зверніться до AI-асистента (наприклад, ChatGPT, Copilot) з запитом: "Як оптимізувати перевірку на щасливе число в Python? Наведи приклад коду з мемоізацією". Проаналізуйте отриману відповідь. Напишіть власну оптимізовану версію, яка використовує множину для зберігання вже перевірених чисел та уникнення повторних обчислень. Порівняйте швидкість роботи обох версій для N=10000.

```
Вхід: N=20 → Вихід: [1, 7, 10, 13, 19]
Вхід: N=50 → Вихід: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49]
Вхід: N=100 → Вихід: (список з 20 чисел)
```

Критерії оцінювання 3.3:

1. Коректність наївної реалізації.
2. Аналіз та розуміння AI-відповіді.
3. Якість власної оптимізованої реалізації.
4. Чітке порівняння продуктивності (можна використати time.time()).

6. Питання для самоперевірки

1. Поясніть принципову різницю між циклами for та while. У яких ситуаціях краще використовувати кожен з них?
2. Що станеться, якщо в циклі while ніколи не зміниться умова, на основі якої він виконується? Наведіть приклад такого коду.
3. Для чого призначена функція range()? Опишіть її основні форми виклику (range(stop), range(start, stop), range(start, stop, step)) та наведіть приклади послідовностей, які вона генерує.

4. Яку роль відіграють оператори break та continue всередині циклу? Чим вони відрізняються? Наведіть короткий приклад для кожного.

5. Коли виконується блок else, що доданий до циклу (for або while)? Чи завжди він виконується після завершення циклу?

6. Що таке "нескінченний цикл" і як його можна навмисно створити? Як безпечно зупинити програму, яка потрапила в нескінченний цикл?

7. Як можна пройтися циклом for по елементах рядка? Наведіть приклад коду, який підраховує кількість певної літери у введеному тексті.

8. Поясніть, як працює вкладений цикл (цикл всередині циклу). Яка загальна кількість ітерацій буде у внутрішньому циклі, якщо зовнішній виконується n разів, а внутрішній – m разів?

9. Що таке змінна-лічильник (accumulator) і змінна-акумулятор? Наведіть приклади задач, де вони використовуються.

10. Як можна "зламати" цикл раніше завершення всіх ітерацій без оператора break? (Підказка: подумайте про зміну умови циклу while або змінної ітерації for).

11. Чи можна використовувати оператор elif без попереднього if? Поясніть правильну структуру використання if-elif-else.

12. Що таке "умовна нескінченність" в циклі while і чому вона небезпечна? Наведіть приклад, де така ситуація може виникнути випадково.

13. Як з допомогою циклу for та функції range() створити зворотний відлік (наприклад, від 10 до 1)?

14. Чи може тіло циклу while не виконатися жодного разу? А циклу for? Наведіть приклади таких ситуацій.

15. Що таке ітерація? Дайте визначення та наведіть приклад ітерації в контексті циклу for.

16. Яка помилка часто виникає при роботі з діапазонами range() у задачах на суму або добуток? (Підказка: включення/виключення границь).

17. Як за допомогою циклу знайти максимальний або мінімальний елемент у послідовності чисел, яка вводиться користувачем?

18. Поясніть логіку роботи циклу на прикладі обчислення факторіала числа. Як би ви модифікували код, щоб обчислити подвійний факторіал (n!!)?

19. У чому небезпека зміни списку (наприклад, видалення елементів) безпосередньо в циклі for, який по ньому ітерується? Як цього уникнути?

20. Як можна використовувати цикл для перевірки введених даних на коректність? Опишіть алгоритм, де програма запитує число доти, доки не буде введено додатнє число.

ЛАБОРАТОРНА РОБОТА №5
Вкладені цикли та складні алгоритми

1. Мета

Сформувати практичні навички проектування та реалізації алгоритмів з використанням вкладених циклів у мові програмування Python; закріпити розуміння принципу вкладеності керуючих структур та навчитися аналізувати складність алгоритмів, що базуються на множинних ітераціях; розвинути логічне та алгоритмічне мислення для розв'язання задач обробки матриць (двовимірних масивів), побудови складних графічних шаблонів та роботи з вкладеними структурами даних.

2. Завдання

1. Освоїти синтаксис та принцип роботи вкладених циклів у Python.
2. Навчитися використовувати вкладені цикли для виведення на екран регулярних графічних шаблонів (трикутники, піраміди, таблиці множення).
3. Опанувати техніку обробки двовимірних масивів (матриць) за допомогою вкладених циклів (введення, виведення, пошук елементів, обчислення сум, добутків).
4. Розвинути вміння аналізувати порядок виконання інструкцій у вкладених циклах та оцінювати складність алгоритмів.
5. Набути досвіду розв'язання практичних задач, які вимагають множинної ітерації, таких як знаходження простих чисел в діапазоні, генерація комбінацій, робота з послідовностями.

3. Короткі теоретичні відомості

Вкладеними циклами називають конструкцію, в якій один цикл (внутрішній) розташований всередині тіла іншого циклу (зовнішнього). Кожен повний прохід зовнішнього циклу запускає виконання внутрішнього циклу від початку до кінця. Це потужний інструмент для роботи з двовимірними даними, такими як таблиці, матриці, координатні сітки, а також для реалізації складних ітеративних процесів.

Ключові концепції:

1. **Порядок виконання:** Для кожного значення лічильника зовнішнього циклу внутрішній цикл виконує *всі* свої ітерації.

2. **Змінні-лічильники:** Зазвичай використовуються різні імена для лічильників (наприклад, i для зовнішнього та j для внутрішнього циклу), щоб уникнути плутанини.

3. **Загальна кількість ітерацій:** Якщо зовнішній цикл виконується n разів, а внутрішній – m разів, то тіло внутрішнього циклу виконається n * m разів. Це ілюструє квадратичну складність $O(n^2)$ у найпростішому випадку.

Основні типи вкладених циклів:

- for в for
- while в while
- for в while та while в for

Задача 1. Виведення прямокутної таблиці символів

```
rows = 4
columns = 6

for i in range(rows):           # Зовнішній цикл по рядках
    for j in range(columns):    # Внутрішній цикл по стовпцях
        print('*', end=' ')
    print()             # Перехід на новий рядок після кожного стовпця
```

Результат:

```
* * * * * *
* * * * * *
* * * * * *
* * * * * *
```

Змінна i відповідає за номер поточного рядка (від 0 до 3). Для кожного фіксованого i виконується повний цикл по j (від 0 до 5), що заповнює рядок зірочками. print() без аргументів в кінці зовнішнього циклу забезпечує перехід на новий рядок.

Задача 2. Побудова трикутного числового шаблону

```
n = 5
for i in range(1, n + 1):          # i - номер рядка та верхня межа для внутр. циклу
    for j in range(1, i + 1):    # j пробігає від 1 до i
        print(j, end=' ')
    print()
```

Результат:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Кількість символів у рядку залежить від номера рядка (i). Це демонструє, як параметр внутрішнього циклу може залежати від лічильника зовнішнього.

Задача 3. Робота з матрицею (списком списків)

```
# Створення матриці 3x3
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Обхід матриці по рядках для обчислення суми всіх елементів
total_sum = 0
for row in matrix:          # row - це список (рядок матриці)
    for element in row:     # element - кожне число в рядку
        total_sum += element
print(f"Сума всіх елементів: {total_sum}")

# Обхід з використанням індексів для отримання діагональних елементів
diagonal = []
for i in range(len(matrix)):          # i - індекс рядка
    for j in range(len(matrix[i])):   # j - індекс стовпця
        if i == j:                    # Елемент на головній діагоналі
            diagonal.append(matrix[i][j])
print(f"Елементи головної діагоналі: {diagonal}")
```

Таблиця: Вибір типу циклу для вкладеності

Ситуація	Рекомендований зовнішній цикл	Рекомендований внутрішній цикл
Відома кількість ітерацій	for	for

(рядки/стовпці)		
Обхід колекції (списки, рядки)	for	for
Ітерація до виконання умови (напр., пошук)	while або for	while
Генерація шаблонів зі змінною кількістю	for	for з залежним діапазоном

Важливо: при роботі з вкладеними циклами слід уважно контролювати моменти ініціалізації змінних, оновлення лічильників та умови виходу, щоб уникнути помилок (наприклад, нескінченних циклів або неправильної кількості ітерацій).

4. Методичні рекомендації

Задача 1. Виведення прямокутника з символів

Написати програму, яка отримує від користувача два цілих числа: rows (кількість рядків) та columns (кількість стовпців). Програма має вивести на екран прямокутник, сформований з символів #, що має вказану кількість рядків і стовпців.

```
# Отримання розмірів від користувача
rows = int(input("Введіть кількість рядків: "))
columns = int(input("Введіть кількість стовпців: "))

# Зовнішній цикл відповідає за рядки
for i in range(rows):
# Внутрішній цикл відповідає за виведення символів у одному рядку
    for j in range(columns):
        print('#', end='')   # Виводимо символ без переходу на новий рядок
    print()   # Після завершення внутрішнього циклу переходимо на новий рядок
```

Приклад вхідних та вихідних даних:

```
Вхід: rows = 3, columns = 5 → Вихід:
#####
#####
#####
Вхід: rows = 1, columns = 4 → Вихід: ####
Вхід: rows = 4, columns = 1 → Вихід:
#
#
```

```
#
#
```

Коментарі:

Це фундаментальний приклад вкладених циклів for. Важливо розуміти, що внутрішній цикл виконується повністю (всі columns ітерацій) для КОЖНОГО кроку зовнішнього циклу. Аргумент end=“ у функції print() забороняє автоматичний перехід на новий рядок після кожного символу. print() без аргументів – це "перехід на новий рядок" після кожного рядка прямокутника.

Задача 2. Побудова числової піраміди

Написати програму, яка отримує від користувача ціле число n і виводить на екран числову піраміду, де перший рядок містить число 1, другий – числа 1 2, третій – 1 2 3 і так далі до n.

```
n = int(input("Введіть ціле число n: "))
# Зовнішній цикл контролює номер поточного рядка (i)
for i in range(1, n + 1):
    # Внутрішній цикл виводить числа від 1 до поточного значення
i
    for j in range(1, i + 1):
        print(j, end=' ')
    # Перехід на новий рядок після завершення внутрішнього
циклу
    print()
```

Приклад вхідних та вихідних даних:

```
Вхід: n = 4 → Вихід:
1
1 2
1 2 3
1 2 3 4
Вхід: n = 1 → Вихід: 1
Вхід: n = 6 → Вихід:
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

Коментарі:

Ключовий момент цього завдання – залежність діапазону внутрішнього циклу (range(1, i + 1)) від лічильника зовнішнього циклу (i). Кожен рядок містить на один елемент більше, ніж попередній. Це класичний приклад формування шаблонів.

Задача 3. Обчислення таблиці множення

Написати програму, яка виводить на екран таблицю множення від 1 до 10 у вигляді відформатованої таблиці (10 рядків на 10 стовпців).

```
# Діапазон чисел для таблиці множення
start = 1
end = 10
# Зовнішній цикл для множників (рядки таблиці)
for i in range(start, end + 1):
    # Внутрішній цикл для множників (стовпці таблиці)
    for j in range(start, end + 1):
        # Виведення добутку з форматуванням на 4 символи
        print(f"{i * j:4}", end='')
    # Перехід на новий рядок після завершення рядка таблиці
    print()
```

Приклад вхідних та вихідних даних:

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Коментарі:

Ця задача демонструє, як вкладені цикли for працюють з фіксованим однаковим діапазоном. Використання f-рядків з форматуванням ({i * j:4}) гарантує, що кожне число займає рівно 4 символи, що забезпечує вирівнювання стовпців. Без форматування таблиця "попливе".

Задача 4. Пошук простих чисел у діапазоні

Написати програму, яка знаходить та виводить усі прості числа в діапазоні від 2 до заданого користувачем числа n.

```
    n = int(input("Введіть верхню межу діапазону (n >= 2): "))
    print(f"Прості числа в діапазоні [2, {n}]:")
    # Зовнішній цикл перебирає всі числа, які перевіряємо на
простоту
    for num in range(2, n + 1):
        is_prime = True  # Прапорець, який вказує, що число поки
що вважається простим
        # Внутрішній цикл перебирає потенційні дільники від 2 до
кореня з num
        for divisor in range(2, int(num ** 0.5) + 1):
            if num % divisor == 0:  # Якщо знайшовся дільник
                is_prime = False
                break   # Припиняємо внутрішній цикл - число не
просте
        # Якщо жоден дільник не знайшовся, виводимо число
        if is_prime:
            print(num, end=' ')
    print()  # Для красивого завершення виведення
```

Приклад вхідних та вихідних даних:

```
Вхід: n = 20 → Вихід: 2 3 5 7 11 13 17 19
Вхід: n = 10 → Вихід: 2 3 5 7
Вхід: n = 2 → Вихід: 2
```

Коментарі:

Це приклад вкладених циклів з умовним виходом (break). Важлива оптимізація: внутрішній цикл йде не до num-1, а лише до int(num ** 0.5) + 1. Якщо число не має дільників до свого квадратного кореня, то воно просте. Прапорець is_prime допомагає відстежити результат перевірки після завершення внутрішнього циклу.

Задача 5. Обробка матриці (знаходження сум)

Дано матрицю (список списків) цілих чисел розміром 3x4. Написати програму, яка:

1. Знаходить суму всіх елементів матриці.
2. Знаходить суму елементів кожного рядка окремо.

```
# Задана матриця 3x4
matrix = [
    [5, 8, -2, 1],
```

```
        [0, 3, 7, 4],
        [6, -1, 9, 2]
    ]
    total_sum = 0  # Змінна для загальної суми

    print("Сума кожного рядка:")
    # Зовнішній цикл проходить по кожному рядку матриці
    for i, row in enumerate(matrix):
        row_sum = 0  # Змінна для суми поточного рядка
        # Внутрішній цикл проходить по кожному елементу рядка
        for element in row:
            row_sum += element
            total_sum += element  # Додаємо елемент і до загальної
суми
        print(f"Рядок {i}: {row_sum}")

    print(f"Загальна сума всіх елементів матриці: {total_sum}")
```

Приклад вхідних та вихідних даних:

```
Сума кожного рядка:
Рядок 0: 12
Рядок 1: 14
Рядок 2: 16
Загальна сума всіх елементів матриці: 42
```

Коментарі:

Це приклад обходу вкладених списків. Змінна row у зовнішньому циклі є списком (рядком матриці). Функція enumerate(matrix) дозволяє отримати одночасно і індекс рядка (i), і сам рядок (row). Важливо правильно ініціалізувати змінні для підсумовування (total_sum = 0 перед зовнішнім циклом, row_sum = 0 на початку кожного рядка).

Типові помилки і шляхи їх усунення

1. **Нескінченний цикл у вкладених while**

   Причина. Виникає, коли забувають оновлювати лічильник внутрішнього циклу або умова виходу ніколи не стає False.

   Рішення. Уважно перевіряйте логіку оновлення змінних (j += 1) та умови (while j < n). Додавайте print-отладку, щоб бачити поточні значення лічильників.

2. **Плутанина з лічильниками i та j**

   Причина. Використання одного й того ж імені змінної в обох циклах призводить до непередбачуваної поведінки.

Рішення. Завжди використовуйте різні імена для лічильників вкладених циклів (стандартно і для зовнішнього, j для внутрішнього).

3. **Неправильне місце ініціалізації або скидання змінних**

   Причина. Наприклад, ініціалізація змінної для підсумовування всередині внутрішнього циклу замість його початку.

   Рішення. Чітко визначайте, для якого рівня циклу потрібна змінна. Якщо вона має "жити" протягом одного проходу зовнішнього циклу (наприклад, сума рядка), її слід ініціалізувати на початку зовнішнього циклу.

4. **Відсутній print() для переходу на новий рядок**

   Причина. Призводить до виведення всіх символів в один довгий рядок.

   Рішення. Пам’ятайте, що print() з аргументом end=“ не переводить рядок. Для переходу потрібен окремий print() після внутрішнього циклу.

5. **Помилки з діапазонами у range()**

   Причина. Особливо при побудові залежних шаблонів (напр., for j in range(i) замість range(1, i+1)).

   Рішення. Уважно аналізуйте, з якого числа має починатися внутрішній цикл і на якому закінчуватися. Зробіть папір-план із значеннями i та j для перших кількох ітерацій.

Корисні поради

1. **Спершу – план на папері.** Перш ніж писати код, намалюйте бажану структуру виведення (шаблон) або опишіть логіку обробки матриці. Запишіть декілька кроків виконання циклів вручну.

2. **Розкладіть задачу на частини.** Складну задачу розбивайте на підзадачі. Наприклад, спочатку напишіть код для введення/створення матриці, потім – для її виведення, і лише потім – для обчислень.

3. **Використовуйте налагоджувальний вивід.** Вставте print(f"i={i}, j={j}") у внутрішній цикл, щоб наочно побачити порядок виконання ітерацій та значення змінних. Це найкращий спосіб зрозуміти роботу вкладених циклів.

4. **Почніть з фіксованих значень.** Замість того, щоб одразу писати код з input(), задайте розміри (rows = 5) або матрицю як константу в коді. Це дозволить зосередитись на логіці циклів, а не на вводі даних.

5. **Пам'ятайте про квадратичну складність.** Алгоритми з вкладеними циклами часто мають складність $O(n^2)$. Для великих n (тисячі, мільйони) вони можуть працювати дуже повільно. Шукайте можливі оптимізації (як у задачі з простими числами).

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Прямокутник з рамкою

Напишіть програму, яка отримує від користувача ширину w та висоту h прямокутника, а потім виводить його на екран. Прямокутник має бути "порожнім" всередині: край складається з символів *, а внутрішня частина – з пробілів.

```
Вхід: w=5, h=4 → Вихід:
*****
*   *
*   *
*****
Вхід: w=3, h=3 → Вихід:
***
* *
***
Вхід: w=2, h=2 → Вихід:
**
**
```

Завдання 1.2. Таблиця підсумів

Напишіть програму, яка генерує таблицю розміром 5x5, де кожен елемент є сумою свого номера рядка та номера стовпця (починаючи з 1). Виведіть таблицю у відформатованому вигляді.

```
  2   3   4   5   6
  3   4   5   6   7
  4   5   6   7   8
  5   6   7   8   9
```

```
  6  7  8  9 10
```

Завдання 1.3. Шахівниця

Напишіть програму, яка виводить на екран шахівницю розміром 8x8. Використовуйте символи # для чорних клітинок та пробіл для білих. Ліва верхня клітинка має бути чорною (#).

```
# # # #
 # # # #
# # # #
 # # # #
# # # #
 # # # #
..
```

Завдання 1.4. Горизонтальна лінійка

Напишіть програму, яка виводить горизонтальну лінійку з позначками. Користувач вводить число n. Програма виводить n поділок: кожна поділка – це вертикальна риска |, а під нею – цифра від 0 до n-1. Цифри розташовуються рівномірно.

```
Вхід: n=5 → Вихід:
| | | | |
0 1 2 3 4
Вхід: n=3 → Вихід:
| | |
0 1 2
```

Завдання 1.5. Діагональна лінія

Напишіть програму, яка виводить квадратну матрицю розміром n x n, де на головній діагоналі стоять символи X, а решта клітинок заповнена крапками ..

```
Вхід: n=4 → Вихід:
X . . .
. X . .
. . X .
. . . X
Вхід: n=2 → Вихід:
X .
. X
```

Завдання 1.6. Правий трикутник

Напишіть програму, яка отримує число n та виводить правий прямокутний трикутник з символів *, де катети дорівнюють n. Трикутник вирівняний по правому краю.

```
Вхід: n=4 → Вихід:
   *
  **
 ***
****
Вхід: n=3 → Вихід:
  *
 **
***
```

Завдання 1.7. Числовий трикутник

Напишіть програму, яка отримує число n та виводить трикутник, де перший рядок містить число 1, другий – 22, третій – 333 і т.д., до n-го рядка.

```
Вхід: n=5 → Вихід:
1
22
333
4444
55555
Вхід: n=3 → Вихід:
1
22
333
```

Завдання 1.8. Зеркальний трикутник

Напишіть програму, яка отримує число n та виводить "дзеркальний" прямокутний трикутник: спочатку зростаючу послідовність від 1 до n зірочок, потім спадаючу від n-1 до 1.

```
Вхід: n=4 → Вихід:
*
**
***
****
***
**
*
```

Завдання 1.9. Трикутник Флойда

Напишіть програму, яка виводить трикутник Флойда заданої висоти n. У першому рядку - число 1, у другому - 2 і 3, у третьому - 4, 5, 6 і т.д. Вирівняйте числа по лівому краю.

```
Вхід: n=4 → Вихід:
1
2 3
4 5 6
7 8 9 10
```

Завдання 1.10. Трикутник з літерами

Напишіть програму, яка отримує число n ($1 \le n \le 26$) та виводить трикутник з літер англійського алфавіту, починаючи з ‘A’. Перший рядок - ‘A’, другий - ‘AB’, третій - ‘ABC’ і т.д.

```
Вхід: n=3 → Вихід:
A
AB
ABC
Вхід: n=5 → Вихід:
A
AB
ABC
ABCD
ABCDE
```

Частина 2. Середні завдання

Завдання 2.1. Ялинка

Напишіть програму, яка отримує число n – кількість ярусів ялинки – та виводить ялинку з символів *. Кожен ярус є трикутником, ширини ярусів зростають на 2. Ялинка має бути центрованою.

```
Вхід: n=3 → Вихід:
  *
 ***
*****
  *
  *  (Останні два рядки - це стовбур ялинки заввишки 2)
```

Завдання 2.2. Ромб

Напишіть програму, яка отримує непарне число n та виводить на екран ромб з символів *, висота якого дорівнює n.

```
Вхід: n=5 → Вихід:
  *
 ***
*****
 ***
  *
```

Завдання 2.3. Числова піраміда

Напишіть програму, яка отримує число n та виводить центровану числову піраміду, де кожен рядок містить непарну кількість чисел, що збільшуються від 1 до номера рядка та назад до 1.

```
Вхід: n=4 → Вихід:
   1
  121
 12321
1234321
```

Завдання 2.4. Матриця-спіраль

Напишіть програму, яка генерує та виводить квадратну матрицю розміром n x n, заповнену числами від 1 до n^2 за спіраллю за годинниковою стрілкою, починаючи з лівого верхнього кута.

```
Вхід: n=4 → Вихід:
 1  2  3  4
12 13 14  5
11 16 15  6
10  9  8  7
```

(форматування може бути спрощеним)

Завдання 2.5. Пошук сідлової точки

Напишіть програму, яка генерує матрицю розміром m x n з випадковими числами від 1 до 100. Знайдіть та виведіть координати всіх "сідлових точок". Елемент є сідловою точкою, якщо він є мінімальним у своєму рядку та максимальним у своєму стовпці одночасно.

```
Вхід: m=3, n=4 (генерується випадкова матриця)
Можливий вихід:
Матриця:
[5, 3, 8, 2]
[6, 7, 4, 1]
[9, 2, 5, 3]
```

```
Сідлова точка: рядок 0, стовпець 3, значення = 2
```

Частина 3. Складні завдання

Завдання 3.1. Генератор усіх можливих паролів

Напишіть програму, яка отримує від користувача максимальну довжину пароля L (1 ≤ L ≤ 4) та список символів (наприклад, [‘a’, ‘b’, ‘c’]). Програма має згенерувати та вивести ВСІ можливі варіанти паролів довжиною від 1 до L, що складаються з заданих символів (символи можуть повторюватися). Виведення має бути в лексикографічному порядку.

```
Вхід: L=2, chars=[‘x’, ‘y’] → Вихід:
x
y
xx
xy
yx
yy
```

Завдання 3.2. Оптимізація розміщення замовлень

У вас є двомірний масив available_time, де available_time[i][j] – це кількість вільних годин у i-го виконавця в j-ий день (індекси з 0). Також є список orders – кортежів (hours, day), де hours – тривалість замовлення в годинах, day – бажаний день виконання. Напишіть програму, яка розподілить замовлення по виконавцях так, щоб мінімізувати кількість задіяних виконавців (при пріоритеті) або максимізувати завантаження. Виведіть матрицю розподілу.

```
Вхід:
available_time = [[4, 2, 5], [3, 1, 0]]  # 2 виконавці, 3 дні
orders = [(2, 0), (3, 1), (1, 2), (4, 0)]
Можливий вихід:
Виконавець 0: Замовлення (2,0) та (1,2). Залишок: [1, 2, 4]
Виконавець 1: Замовлення (3,1). Залишок: [3, -2, 0]
Нерозподілене: (4,0)
```

Завдання 3.3. Аналіз траєкторій у лабіринті

Дано лабіринт як двомірний масив maze розміром N x M, де 0 – вільна клітинка, 1 – стіна. Початкова позиція (start_x, start_y), кінцева (end_x, end_y). Напишіть програму, яка знаходить ВСІ можливі

траєкторії довжиною не більше max_steps, що ведуть від початку до кінця, не проходять через стіни та не відвідують одну й ту ж клітинку двічі. Виведіть кількість знайдених траєкторій та перші 3 траєкторії (як послідовність координат).

```
Вхід:
N=3, M=3
maze = [
    [0, 0, 0],
    [0, 1, 0],
    [0, 0, 0]
]
start = (0,0), end = (2,2), max_steps = 6
Можливий вихід:
Знайдено траєкторій: 4
Траєкторія 1: (0,0)→(0,1)→(0,2)→(1,2)→(2,2)
Траєкторія 2: (0,0)→(1,0)→(2,0)→(2,1)→(2,2)
Траєкторія 3: (0,0)→(0,1)→(1,1)→(2,1)→(2,2)
```

6. Питання для самоперевірки

1. Що таке вкладені цикли та у яких типових задачах їх необхідно застосовувати?

2. Поясніть порядок виконання інструкцій у конструкції, де цикл for j in range(3) вкладений у цикл for i in range(2). Скільки разів виконається тіло внутрішнього циклу?

3. Яка роль змінної-лічильника зовнішнього циклу при формуванні залежних шаблонів (наприклад, трикутників)? Наведіть приклад діапазону внутрішнього циклу, що залежить від i.

4. У чому різниця між виведенням рядка символів з використанням print(‘*’, end=‘‘) та звичайним print(‘*’) у контексті вкладених циклів?

5. Як створити "порожній" прямокутник (рамку) за допомогою вкладених циклів? За якої умови всередині циклів потрібно виводити пробіл, а не символ?

6. Що таке головна діагональ квадратної матриці? Як умова i == j у вкладених циклах допомагає з нею працювати?

7. Як вивести числовий трикутник, де кожен рядок містить однакові числа (1, 22, 333...), а не зростаючу послідовність?

8. Поясніть призначення оператора break у внутрішньому циклі. Наведіть приклад задачі (наприклад, пошук простих чисел), де його використання є доцільним.

9. Як змінити программу побудови прямокутного трикутника, вирівняного по лівому краю, щоб отримати трикутник, вирівняний по правому краю?

10. Що таке "прапорець" (flag) і навіщо він використовується в алгоритмах із вкладеними циклами? Наведіть приклад.

11. Як за допомогою вкладених циклів for можна згенерувати всі пари елементів з двох різних списків list_a та list_b?

12. У чому полягає оптимізація перевірки числа на простоту шляхом обмеження внутрішнього циклу діапазоном до int(num ** 0.5) + 1 замість до num?

13. Що станеться, якщо випадково використати одну й ту саму змінну-лічильник (наприклад, і) і для зовнішнього, і для внутрішнього циклу? Проілюструйте на простому прикладі.

14. Як можна за допомогою вкладених циклів обчислити суму всіх елементів двовимірного списку (матриці)? Чому змінну для акумулювання суми слід ініціалізувати перед початком циклів?

15. Опишіть алгоритм побудови центрованої фігури (наприклад, ялинки) за допомогою вкладених циклів. Як розрахувати кількість пробілів перед кожним рядком?

16. Які типові помилки призводять до нескінченного виконання програми з вкладеними циклами while? Як їх уникнути?

17. За якого принципу працює алгоритм заповнення матриці по спіралі? Які змінні-індекси та їх зміни необхідні для його реалізації?

18. Як можна використати вкладені цикли для розв’язання задачі пошуку елемента у двовимірному списку? Як зупинити пошук після знаходження першого відповідного елемента?

19. У чому складність (Big O) простого алгоритму з двома вкладеними циклами, кожен з яких виконується n разів? Як ця складність впливає на час виконання при збільшенні n?

20. Чим відрізняється логіка генерації "трикутника Паскаля" від логіки генерації звичайного числового трикутника? Яка роль вкладених циклів у цьому процесі?

ЛАБОРАТОРНА РОБОТА №6
Робота з рядками в Python

1. Мета

Оволодіти практичними навичками роботи зі рядками в Python, зокрема освоїти методи їх створення, індексації, обробки (зміна регістру, очищення, розбиття, пошук, заміна), форматування та базового аналізу. Закріпити знання про властивості рядків як незмінних послідовностей символів. Навчитися застосовувати методи рядків для Рішення типових задач, таких як перевірка формату даних, обробка тексту та визначення особливих властивостей (паліндроми, анаграми).

2. Завдання

1. Зрозуміти принципи індексації та зрізів (slicing) рядків.
2. Навчитися модифікувати регістр символів у рядках.
3. Опанувати методи очищення рядків від зайвих символів.
4. Розібратися з пошуком та підрахунком підрядків.
5. Навчитися заміняти частини рядка або розбивати його на складові.
6. Закріпити вміння перевіряти тип вмісту рядка (цифри, букви тощо).
7. Застосувати різні способи форматування рядків для об’єднання даних.
8. Використати отримані знання для Рішення практичних задач (аналіз текстів, робота з паліндромами та анаграмами).

3. Короткі теоретичні відомості

У Python рядок (string) – це **незмінна** (immutable) послідовність символів Unicode. Це означає, що будь-яка операція, що "змінює" рядок, насправді створює новий об’єкт.

Індексація та зрізи (Slicing)

Кожен символ у рядку має свій індекс. Індексація починається з 0. Також підтримується негативна індексація з кінця рядка (-1 – останній символ).

```
my_string = "Algorithm"
print(my_string[0])   # A
print(my_string[-1])  # m
```

Зріз дозволяє отримати підрядок: [start:stop:step]. Параметр stop не включається в результат.

```
print(my_string[0:4])    # Algo (символи з індексами 0,1,2,3)
print(my_string[:4])     # Algo (початок з 0 за замовчуванням)
print(my_string[4:])     # rithm (до кінця)
print(my_string[::2])    # Agrtm (кожен другий символ)
print(my_string[::-1])   # mhtiroglA (обернення рядка)
```

Зміна регістру

• upper(): перетворює всі символи на верхній регістр.

• lower(): перетворює всі символи на нижній регістр.

• capitalize(): перетворює перший символ рядка на верхній регістр, решту – на нижній.

• title(): перетворює перший символ *кожного слова* на верхній регістр.

Очищення рядків

Видаляє пробіли та службові символи (наприклад, \n – переведення рядка) з початку та/або кінця.

• strip(): з обох сторін.

• lstrip(): тільки зліва (початку).

• rstrip(): тільки справа (кінця).

Пошук підрядків

• find(substr): повертає **індекс першого входження** підрядка substr, або -1, якщо не знайдено.

• index(substr): аналогічно find(), але викидає виняток ValueError, якщо підрядок не знайдено.

• count(substr): повертає кількість непересічних входжень підрядка substr.

```
text = "банан"
print(text.find("ан"))   # 1
print(text.count("а"))   # 2
```

Заміна символів (replace)

old_str.replace(old, new[, count]) – повертає копію рядка, де всі (або не більше count) входження підрядка old замінені на new.

```
s = "Hello, World!"
new_s = s.replace("o", "0")   # "Hell0, W0rld!"
```

Розбиття рядків (split) та об'єднання (join)

• split(sep=None): розбиває рядок на список підрядків за роздільником sep (за замовчуванням – за пробілами).

• join(iterable): об'єднує елементи ітерабельного об'єкта (наприклад, списка) в один рядок, вставляючи між ними поточний рядок.

```
data = "apple,banana,cherry"
items = data.split(",")  # ['apple', 'banana', 'cherry']
result = " - ".join(items)  # 'apple - banana - cherry'
```

Перевірка вмісту (is-методи)

Ці методи повертають True або False.

• isdigit(): чи складається рядок лише з цифр?

• isalpha(): чи лише з букв (будь-якої мови)?

• isalnum(): чи лише з букв та/або цифр?

• isspace(): чи лише з пробільних символів?

• startswith(prefix), endswith(suffix): перевірка початку чи кінця рядка.

Форматування рядків

1. **f-рядки (рекомендовано, Python 3.6+):** найзручніший спосіб.

```
name = "Олена"
age = 20
info = f"Мене звати {name} і мені {age} років."
```

2. **format() метод:**

```
info = "Мене звати {} і мені {} років.".format(name, age)
```

3. **Конкатенація (зчеплення):** просте додавання рядків за допомогою +.

Ключові концепти для завдань

• **Паліндром** – слово, фраза або послідовність символів, яка читається однаково в обох напрямках (ігноруючи пробіли, регістр та розділові знаки). Наприклад, "А роза упала на лапу Азора".

• **Анаграма** – слово або фраза, утворена перестановкою букв іншого слова. Наприклад, "кот" – "ток".

4. Методичні рекомендації

Нижче наведені розв'язки типових задач, які демонструють застосування методів роботи з рядками на практиці.

Задача 1. Форматування повного імені

Користувач вводить три слова: прізвище, ім'я та по батькові через пробіл. Програма повинна вивести повне ім'я у трьох форматах:

1. "ПІБ: прізвище ім'я по-батькові" (з великих літер)
2. "Повне ім'я: Ім'я По-батькові Прізвище"
3. "Ініціали: П.І.Б." (кожна літера з крапкою)

```
    # Зчитування введення від користувача
    full_name_input = input("Введіть прізвище, ім'я та по батькові через пробіл: ")

    # Розділяємо рядок на окремі слова
    name_parts = full_name_input.split()

    # Перевіряємо, чи введено три слова
    if len(name_parts) == 3:
        surname, name, patronymic = name_parts

        # 1. Формат "ПІБ: прізвище ім'я по-батькові" (з великих літер)
        pib_format = f"{surname} {name} {patronymic}"
        print(f"ПІБ: {pib_format.upper()}")

        # 2. Формат "Повне ім'я: Ім'я По-батькові Прізвище" (з великої літери кожне слово)
        full_name_format = f"{name.title()} {patronymic.title()} {surname.title()}"
        print(f"Повне ім'я: {full_name_format}")

        # 3. Формат "Ініціали: П.І.Б."
        initials = f"{surname[0].upper()}.{name[0].upper()}.{patronymic[0].upper()}."
        print(f"Ініціали: {initials}")
    else:
        print("Помилка. потрібно ввести рівно три слова!")
```

Приклад вхідних та вихідних даних:

```
Вхід: "іванов петро сидорович"
Вихід:
ПІБ: ІВАНОВ ПЕТРО СИДОРОВИЧ
```

```
Повне ім’я: Петро Сидорович Іванов
Ініціали: І.П.С.

Вхід: "Петренко Марія Іванівна"
Вихід:
ПІБ: ПЕТРЕНКО МАРІЯ ІВАНІВНА
Повне ім’я: Марія Іванівна Петренко
Ініціали: П.М.І.

Вхід: "Шевченко Тарас Григорович"
Вихід:
ПІБ: ШЕВЧЕНКО ТАРАС ГРИГОРОВИЧ
Повне ім’я: Тарас Григорович Шевченко
Ініціали: Ш.Т.Г.
```

Коментарі:

1. Метод split() розбиває рядок на список слів за пробілами.
2. upper() перетворює всі літери у верхній регістр.
3. title() робить першу літеру кожного слова великою.
4. surname[0] отримує першу літеру прізвища через індексацію.

Задача 2. Перевірка паліндрому

Користувач вводить слово. Програма перевіряє, чи є воно паліндромом (читається однаково зліва направо та справа наліво), ігноруючи регістр.

```python
# Зчитування слова від користувача
word = input("Введіть слово для перевірки на паліндром: ")

# Приводимо до нижнього регістру для коректного порівняння
word_lower = word.lower()

# Отримуємо зворотній варіант слова за допомогою зрізу
reversed_word = word_lower[::-1]

# Порівнюємо оригінал та реверс
if word_lower == reversed_word:
    print(f"Слово '{word}' є паліндромом!")
else:
    print(f"Слово '{word}' НЕ є паліндромом.")

# Додатковий вивід для наочності
print(f"Оригінал: {word_lower}")
```

```
print(f"Реверс:   {reversed_word}")
```

Приклад вхідних та вихідних даних:

```
Вхід: "Козак"
Вихід:
Слово 'Козак' є паліндромом!
Оригінал: козак
Реверс:   козак

Вхід: "Python"
Вихід:
Слово 'Python' НЕ є паліндромом.
Оригінал: python
Реверс:   nohtyp

Вхід: "Анна"
Вихід:
Слово 'Анна' є паліндромом!
Оригінал: анна
Реверс:   анна
```

Коментарі:

1. lower() робить всі літери маленькими для нечутливості до регістру.
2. Зріз [::-1] створює обернений рядок – це найпростіший спосіб.
3. Порівняння рядків через == перевіряє, чи вони ідентичні.

Задача 3. Статистика речення

Користувач вводить речення. Програма аналізує його та виводить:

1. Кількість слів
2. Кількість символів (з пробілами та без)
3. Кількість голосних літер
4. Кількість приголосних літер

```
# Зчитування речення від користувача
sentence = input("Введіть речення для аналізу: ")

# 1. Кількість слів
words = sentence.split()
word_count = len(words)

# 2. Кількість символів
total_chars = len(sentence)
```

```
chars_without_spaces = len(sentence.replace(" ", ""))

# 3. Підрахунок голосних та приголосних
vowels = "аеиіоуяюєїАЕИІОУЯЮЄЇ"
consonants = "бвгґджзйклмнпрстфхцчшщБВГҐДЖЗЙКЛМНПРСТФХЦЧШЩ"

vowel_count = 0
consonant_count = 0

for char in sentence:
    if char in vowels:
        vowel_count += 1
    elif char in consonants:
        consonant_count += 1

# Вивід результатів
print("\nРезультати аналізу речення:")
print("=" * 30)
print(f"1. Кількість слів: {word_count}")
print(f"2. Символів (з пробілами): {total_chars}")
print(f"   Символів (без пробілів): {chars_without_spaces}")
print(f"3. Голосних літер: {vowel_count}")
print(f"4. Приголосних літер: {consonant_count}")

# Додатково: кожне слово з його довжиною
print("\nСлова та їх довжина:")
for word in words:
    print(f"  '{word}' - {len(word)} символів")
```

Приклад вхідних та вихідних даних:

```
Вхід: "Python - це потужна мова"
Вихід:
Кількість слів: 5
Символів (з пробілами): 23
Символів (без пробілів): 20
Голосних літер: 8
Приголосних літер: 11
Слова та їх довжина:
  'Python' - 6 символів
  '-' - 1 символів
  'це' - 2 символів
  'потужна' - 7 символів
  'мова' - 4 символів

Вхід: "Алгоритмізація важлива"
```

```
Вихід:
Кількість слів: 2
Символів (з пробілами): 23
Символів (без пробілів): 21
Голосних літер: 10
Приголосних літер: 10
```

Коментарі:

1. split() розбиває речення на слова за пробілами.
2. replace(" ", "") видаляє всі пробіли з рядка.
3. Оператор in перевіряє, чи міститься символ у рядку голосних/приголосних.
4. Цикл for проходить через кожен символ речення.

Задача 4. Перевірка анаграм

Користувач вводить два слова через пробіл. Програма перевіряє, чи є вони анаграмами (складаються з тих самих букв у різному порядку).

```
# Зчитування двох слів
input_words = input("Введіть два слова через пробіл: ")

# Розділяємо на два окремих слова
words_list = input_words.split()
if len(words_list) == 2:
    word1, word2 = words_list

    # Приводимо до нижнього регістру для порівняння
    word1_lower = word1.lower()
    word2_lower = word2.lower()

    # Перевіряємо, чи мають слова однакову довжину
    if len(word1_lower) != len(word2_lower):
        print(f"Слова 'word1' та 'word2' НЕ є анаграмами (різна довжина)")
    else:
        # Сортуємо літери в кожному слові
        sorted_word1 = ''.join(sorted(word1_lower))
        sorted_word2 = ''.join(sorted(word2_lower))

        # Порівнюємо відсортовані варіанти
        if sorted_word1 == sorted_word2:
            print(f"Слова 'word1' та 'word2' є анаграмами!")
```

```
            else:
                print(f"Слова    '{word1}'    та    '{word2}'    НЕ    є
анаграмами")

            # Додаткова інформація для наочності
            print(f"Відсортовані літери першого слова: {sorted_word1}")
            print(f"Відсортовані літери другого слова: {sorted_word2}")
    else:
        print("Помилка. потрібно ввести рівно два слова!")
```

Приклад вхідних та вихідних даних:

```
Вхід: "кот ток"
Вихід:
Слова 'кот' та 'ток' Є анаграмами!
Відсортовані літери першого слова: кот
Відсортовані літери другого слова: кот

Вхід: "listen silent"
Вихід:
Слова 'listen' та 'silent' Є анаграмами!
Відсортовані літери першого слова: eilnst
Відсортовані літери другого слова: eilnst

Вхід: "apple orange"
Вихід:
Слова 'apple' та 'orange' НЕ є анаграмами
Відсортовані літери першого слова: aelpp
Відсортовані літери другого слова: aegnor
```

Коментарі:

1. sorted() повертає список відсортованих символів.
2. “.join() об’єднує список символів назад у рядок.
3. Порівняння довжин слова – проста оптимізація, яка відсікає явно не анаграмні пари.
4. lower() забезпечує нечутливість до регістру.

Задача 5. Шифр Цезаря (простий зсув букв)

Користувач вводить слово та число N. Програма зсуває кожну літеру слова на N позицій в алфавіті (з переносом від ‘я’ до ‘а’).

```
# Зчитування даних
word = input("Введіть слово для шифрування: ")
shift_input = input("Введіть число для зсуву: ")
```

```
    # Перетворюємо введене число в ціле
    try:
        shift = int(shift_input)

        # Український алфавіт
        alphabet = 'абвгґдеєжзиіїйклмнопрстуфхцчшщьюя'

        # Результат шифрування
        encrypted_word = ""

        # Шифруємо кожну літеру
        for letter in word.lower():
            if letter in alphabet:
                # Знаходимо поточну позицію літери
                current_index = alphabet.index(letter)

                # Обчислюємо нову позицію з урахуванням зсуву
                new_index = (current_index + shift) %
len(alphabet)

                # Додаємо зашифровану літеру до результату
                encrypted_word += alphabet[new_index]
            else:
                # Якщо символ не літера, додаємо його як є
                encrypted_word += letter

        # Вивід результату
        print(f"\nОригінальне слово: {word}")
        print(f"Зсув: {shift}")
        print(f"Зашифроване слово: {encrypted_word}")

        # Дешифрування для перевірки
        decrypted_word = ""
        for letter in encrypted_word:
            if letter in alphabet:
                current_index = alphabet.index(letter)
                new_index = (current_index - shift) %
len(alphabet)
                decrypted_word += alphabet[new_index]
            else:
                decrypted_word += letter

        print(f"Розшифроване слово: {decrypted_word}")
```

```
except ValueError:
    print("Помилка. друге значення має бути числом!")
```

Приклад вхідних та вихідних даних:

```
Вхід: слово "привіт", зсув 3
Вихід:
Оригінальне слово: привіт
Зсув: 3
Зашифроване слово: туелїх
Розшифроване слово: привіт

Вхід: слово "алгоритм", зсув 5
Вихід:
Оригінальне слово: алгоритм
Зсув: 5
Зашифроване слово: епчмфухс
Розшифроване слово: алгоритм

Вхід: слово "Python", зсув 1
Вихід:
Оригінальне слово: Python
Зсув: 1
Зашифроване слово: python
Розшифроване слово: python
```

Коментарі:

1. index() знаходить позицію літери в алфавіті.
2. Оператор % (остача від ділення) забезпечує "зациклювання" алфавіту.
3. Цикл for проходить через кожну літеру слова.
4. lower() забезпечує роботу з маленькими літерами.
5. Умовний оператор if letter in alphabet: перевіряє, чи є символ літерою українського алфавіту.

Типові помилки і шляхи їх усунення

1. **Неправильне використання індексів**

```
# Помилка - вихід за межі рядка:
word = "Hello"
print(word[10])  # IndexError

# Правильно - перевіряйте довжину:
```

```
if 10 < len(word):
    print(word[10])
```

2. **Забувають перетворювати рядок у число**

```
# Помилка.
number = input("Введіть число: ")
result = number + 5  # TypeError

# Правильно:
number = int(input("Введіть число: "))
result = number + 5
```

3. **Плутають методи, що повертають новий рядок, з тими, що змінюють поточний**

```
# Помилка.
text = "  Hello  "
text.strip()  # Нічого не змінюється!
print(f"'{text}'")  # Виведе '  Hello  '

# Правильно:
text = "  Hello  "
text = text.strip()  # Зберігаємо результат
print(f"'{text}'")  # Виведе 'Hello'
```

4. **Неправильне форматування рядків**

```
# Помилка.
name = "Іван"
age = 20
print(name + " має " + age + " років")  # TypeError

# Правильно:
print(name + " має " + str(age) + " років")
print(f"{name} має {age} років")  # Найкращий варіант
```

5. **Проблеми з кодуванням українських літер**

Завжди вказуйте кодування у верхній частині файлу:
-*- coding: utf-8 -*-
Або зберігайте файл у UTF-8

Корисні поради

1. **Використовуйте f-рядки** для простого форматування:

```
name = "Марія"
score = 95
print(f"{name} отримала {score} балів")  # Найзрозуміліше
```

2. **Розбивайте складні задачі на прості кроки**:

```
# Замість одного складного рядка:
# result = word.lower().replace(" ", "").replace(".", "")
# Краще покроково:
word_lower = word.lower()
word_no_spaces = word_lower.replace(" ", "")
result = word_no_spaces.replace(".", "")
```

3. **Перевіряйте введення користувача**:

```
# Завжди перевіряйте, чи не порожній ввід:
user_input = input("Введіть текст: ")
if user_input.strip():  # Якщо не порожній
    # Обробляємо
else:
print("Ви нічого не ввели!")
```

4. **Використовуйте змінні для проміжних результатів** для кращої читабельності:

```
# Краще:
first_letter = word[0]
last_letter = word[-1]
middle_part = word[1:-1]

# Ніж:
result = word[-1] + word[1:-1] + word[0]
```

5. **Експериментуйте в інтерактивному режимі Python** (python або python3 у терміналі) для швидкого тестування методів.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Вилучення символа за індексом

Напишіть програму, яка отримує рядок та ціле число N (індекс) від користувача. Виведіть символ, який знаходиться на позиції N. Якщо індекс виходить за межі рядка, виведіть повідомлення "Індекс поза межами рядка".

```
Вхід: "Програмування", 4 → Вихід: "а"
Вхід: "Python", 0 → Вихід: "P"
Вхід: "Hello", 10 → Вихід: "Індекс поза межами рядка"
```

Завдання 1.2. Форматування номера телефона

Користувач вводить номер телефону у форматі 10 цифр (наприклад, 0951234567). Програма має переформатувати його до вигляду: "+38(0XX)XXX-XX-XX", де XX - відповідні цифри.

```
Вхід: "0951234567" → Вихід: "+38(095)123-45-67"
Вхід: "0639876543" → Вихід: "+38(063)987-65-43"
Вхід: "0501112233" → Вихід: "+38(050)111-22-33"
```

Завдання 1.3. Перша та остання літера

Напишіть програму, яка приймає рядок та виводить першу і останню літеру цього рядка разом з їх індексами. Якщо рядок порожній, виведіть "Рядок порожній".

```
Вхід: "Алгоритм" → Вихід: "Перша: 'А'(0), остання: 'м'(7)"
Вхід: "Python" → Вихід: "Перша: 'P'(0), остання: 'n'(5)"
Вхід: "О" → Вихід: "Перша: 'О'(0), остання: 'О'(0)"
```

Завдання 1.4. Кожен другий символ

Користувач вводить слово. Виведіть це слово, а під ним - тільки кожен другий символ (починаючи з першого). Використовуйте зрізи (slicing).

```
Вхід: "Програмування" → Вихід:
Програмування
Пормуаня

Вхід: "Комп'ютер" → Вихід:
Комп'ютер
Кмюе

Вхід: "1234567890" → Вихід:
1234567890
13579
```

Завдання 1.5. Нормалізація імені користувача

Користувач вводить своє ім'я у будь-якому регістрі (наприклад, "оЛЕНА", "іВАН"). Програма має вивести це ім'я у правильному форматі: перша літера велика, решта - маленькі.

```
Вхід: "оЛЕНА" → Вихід: "Олена"
Вхід: "іВАН" → Вихід: "Іван"
Вхід: "пЕТРО" → Вихід: "Петро"
```

Завдання 1.6. Очищення електронної пошти

Користувач вводить електронну адресу, яка може містити зайві пробіли на початку або в кінці. Програма повинна очистити адресу та перевести всі символи до нижнього регістру.

```
Вхід: "  User@Example.COM " → Вихід: "user@example.com"
Вхід: "\tstudent@univ.edu.ua\n" → Вихід: "student@univ.edu.ua"
Вхід: "  INFO@SITE.UA  " → Вихід: "info@site.ua"
```

Завдання 1.7. Форматування назви курсу

Користувач вводить назву навчального курсу у вигляді "алгоритмізація та програмування". Програма має перетворити її на формат заголовка (кожне слово з великої літери).

```
Вхід: "алгоритмізація та програмування" → Вихід: "Алгоритмізація Та Програмування"
Вхід: "основи комп'ютерних наук" → Вихід: "Основи Комп'ютерних Наук"
Вхід: "веб-технології та дизайн" → Вихід: "Веб-Технології Та Дизайн"
```

Завдання 1.8. Пошук позиції слова

Користувач вводить речення та слово для пошуку. Програма має знайти та вивести позицію (індекс) першого входження цього слова у реченні. Якщо слово не знайдено, вивести "-1".

```
Вхід: "Python - це потужна мова", "потужна" → Вихід: 11
Вхід: "Україна - це наша держава", "наша" → Вихід: 13
Вхід: "Сонячно та тепло", "дощ" → Вихід: -1
```

Завдання 1.9. Заміна роздільника

Користувач вводить дату у форматі "дд.мм.рррр". Програма має перетворити її у формат "дд/мм/рррр".

```
Вхід: "15.03.2024" → Вихід: "15/03/2024"
Вхід: "01.12.2023" → Вихід: "01/12/2023"
Вхід: "25.07.2025" → Вихід: "25/07/2025"
```

Завдання 1.10. Підрахунок входжень символу

Користувач вводить рядок та символ. Програма має підрахувати, скільки разів цей символ зустрічається у рядку (без урахування регістру для літер).

```
Вхід: "Малинова ароматерапія", "а" → Вихід: 5
```

```
Вхід: "Programming in Python", "p" → Вихід: 2
Вхід: "123-45-67", "-" → Вихід: 2
```

Частина 2. Середні завдання

Завдання 2.1. Форматування CSV-даних

Користувач вводить рядок у форматі "Ім’я;Прізвище;Вік;Місто" (наприклад, "Іван;Петренко;20;Київ"). Програма має розбити цей рядок, переформатувати та вивести у вигляді таблиці з заголовками та вирівнюванням.

```
Вхід: "Іван;Петренко;20;Київ"
Вихід:
```

Ім’я	Прізвище	Вік	Місто
Іван	Петренко	20	Київ

```
Вхід: "Марія;Іваненко;19;Львів"
Вихід: (аналогічна таблиця з новими даними)
```

Завдання 2.2. Генератор паролів (базовий)

Напишіть програму, яка генерує пароль за такими правилами:

1. Користувач вводить слово-основу
2. Програма замінює певні символи: ‘a’ → ‘@’, ‘i’ → ‘!’, ‘o’ → ‘0’, ‘s’ → ‘$’
3. Додає до кінця дві останні цифри поточного року
4. Переводить першу літеру у верхній регістр

```
Вхід: "password" → Вихід: "P@$$w0rd24" (припустимо, рік 2024)
Вхід: "security" → Вихід: "Security24"
Вхід: "algorithm" → Вихід: "@lgorithm24"
```

Завдання 2.3. Аналіз логу доступу

Користувач вводить рядок логу у форматі "IP-адреса - дата - запит - код відповіді" (наприклад, "192.168.1.1 - 15/Mar/2024 - GET /index.html - 200"). Програма має виокремити та вивести окремо кожну частину, а також визначити тип відповіді: 2xx - успішно, 3xx - перенаправлення, 4xx - помилка клієнта, 5xx - помилка сервера.

```
Вхід: "192.168.1.1 - 15/Mar/2024 - GET /index.html - 200"
```

```
Вихід:
IP: 192.168.1.1
Дата: 15/Mar/2024
Запит: GET /index.html
Код: 200 (успішно)

Вхід: "10.0.0.5 - 16/Mar/2024 - POST /login.php - 404"
Вихід:
IP: 10.0.0.5
Дата: 16/Mar/2024
Запит: POST /login.php
Код: 404 (помилка клієнта)
```

Завдання 2.4. Перевірка формату ISBN

Користувач вводить номер ISBN-10 (10 символів, може містити цифри та символ ‘X’ на останній позиції). Програма має перевірити правильність формату та обчислити контрольну суму за алгоритмом ISBN-10. Формат: сума добутків цифри на її позицію (з 1 до 10) має ділитися на 11. ‘X’ означає 10.

```
Вхід: "0306406152" → Вихід: "ISBN коректний"
Вхід: "030640615X" → Вихід: "ISBN коректний" (X = 10)
Вхід: "1234567890" → Вихід: "ISBN некоректний"
```

Завдання 2.5. Аналізатор тексту

Користувач вводить текст (речення). Програма має вивести:

1. Кількість слів
2. Кількість унікальних слів (без урахування регістру)
3. Найдовше слово та його довжину
4. Слово, яке зустрічається найчастіше (якщо декілька - перше)

```
Вхід: "Python це мова програмування. Python популярний."
Вихід:
Слів: 6
Унікальних слів: 5
Найдовше слово: програмування (13 символів)
Найчастіше слово: Python (2 рази)

Вхід: "Київ столиця України. Україна це держава."
Вихід:
Слів: 7
Унікальних слів: 6
Найдовше слово: столиця (7 символів)
```

```
Найчастіше слово: Україна (2 рази)
```

Частина 3. Складні завдання

Завдання 3.1. Розширений аналізатор паліндромів

Напишіть програму, яка аналізує текст на наявність паліндромів. Програма повинна:

1. Знайти всі слова, що є паліндромами (довжиною 3+ символів)
2. Для кожного паліндрому вивести його довжину та тип (слово-паліндром чи фраза-паліндром, якщо містить пробіли)
3. Визначити найдовший паліндром у тексті
4. Перевірити, чи можна з усіх знайдених паліндромів скласти новий паліндром (використовуючи перші літери)

```
Вхід: "Козак з казок. А роза упала на лапу Азора. Тут і там."
Вихід:
Знайдені паліндроми:
1. Козак (5) - слово
2. А роза упала на лапу Азора (24) - фраза
Найдовший: "А роза упала на лапу Азора"
З літер паліндромів: "КА" → не паліндром

Вхід: "Level, radar, civic, rotor"
Вихід:
Знайдені паліндроми:
1. Level (5) - слово
2. radar (5) - слово
3. civic (5) - слово
4. rotor (5) - слово
Найдовший: (всі однакової довжини)
З літер паліндромів: "LRCR" → не паліндром
```

Завдання 3.2. Детектор анаграм та підрахунок очок

Користувач вводить список слів через кому. Програма повинна:

1. Знайти всі групи анаграм (слова, що складаються з тих самих літер)
2. Для кожної групи вивести слова та їх "вартість" за правилами Scrabble (українські літери):

3. А=1, Б=2, В=1, Г=2, Ґ=3, Д=2, Е=1, Є=3, Ж=4, З=2, И=1, І=1, Ї=4, Й=4, К=2, Л=2, М=2, Н=1, О=1, П=2, Р=2, С=2, Т=1, У=1, Ф=4, Х=4, Ц=4, Ч=4, Ш=4, Щ=6, Ь=4, Ю=4, Я=2

4. Визначити найціннішу анаграму в кожній групі

5. Вивести загальну статистику

```
Вхід: "кот, ток, від, дів, ліс, сила, лис, салі"
Вихід:
Група 1: кот(8), ток(8) - найцінніша: обидва по 8
Група 2: від(8), дів(8) - найцінніша: обидва по 8
Група 3: ліс(4), сила(6), лис(4), салі(6) - найцінніша: сила, салі по 6
Загалом: 3 групи анаграм, 8 слів
```

Завдання 3.3. Система шифрування з ключем

Реалізуйте систему шифрування, де користувач вводить текст та ключ-слово. Програма повинна:

1. Шифрувати текст за допомогою шифру Віженера (зсув кожної літери на значення відповідної літери ключа)

2. Підтримувати український алфавіт (33 літери)

3. Мати режими шифрування та дешифрування

4. Аналізувати частоту літер у зашифрованому тексті та порівнювати з частотами української мови

5. Перевіряти стійкість пароля (ключа) за критеріями: довжина, різноманітність символів, наявність різних регістрів

```
Вхід (шифрування):
Текст: "Програмування це цікаво"
Ключ: "Python"
Режим: шифрування
Вихід:
Зашифрований текст: "Хцтхєєчюцтво иж йїомтч"
Аналіз частоти: (таблиця з частотами літер)
Стійкість ключа: середня (довжина 6, тільки малі літери)
Вхід (дешифрування):
Текст: "Хцтхєєчюцтво иж йїомтч"
Ключ: "Python"
Режим: дешифрування
Вихід:
Розшифрований текст: "Програмування це цікаво"
```

6. Питання для самоперевірки

1. Яка основна властивість рядків у Python стосується їх змінюваності? Наведіть приклад, що демонструє цю властивість.

2. Поясніть різницю між індексами s[2] та s[-2] для рядка s = "Python". Що повернуть ці вирази?

3. Що поверне вираз "Hello World"[6:11]? Поясніть, чому останній індекс не включається до результату.

4. У чому різниця між методами find() та index()? Коли краще використовувати кожен з них?

5. Який результат дасть вираз " Python ".strip()? А що повернуть методи lstrip() та rstrip() для цього ж рядка?

6. Напишіть три різні способи отримати обернений рядок (реверс) за допомогою зрізів.

7. Чим відрізняються методи capitalize(), title() та upper()? Наведіть приклади для рядка "hello world".

8. Яку помилку можна допустити при використанні replace()? Наведіть код, що демонструє цю помилку.

9. Що поверне "123abc".isalnum()? А "123 abc".isalnum()? Поясніть різницю.

10. Як розбити рядок "apple,banana,cherry" на список слів за комою? Як потім об’єднати цей список назад у рядок з роздільником " - "?

11. Поясніть принцип роботи f-рядків. Наведіть приклад форматування, де виводяться ім’я та вік з двома знаками після коми для віку.

12. Як перевірити, чи починається рядок з певного підрядка без використання зрізів? Наведіть два методи.

13. Що таке паліндром? Напишіть алгоритм перевірки паліндрому, який ігнорує пробіли, регістр та розділові знаки.

14. Як визначити, чи є два слова анаграмами? Наведіть два різних підходи до розв’язання цієї задачі.

15. Чому split() без параметрів краще використовувати для розбиття тексту на слова, ніж split(" ")?

16. Як правильно об’єднати багато рядків у один для ефективної роботи програми? Порівняйте ефективність join() та конкатенації через +.

17. Що повернуть такі вирази для рядка s = "Програмування":
    - s[100:]
    - s.find("z")
    - s.count("a")

- s.replace("a", "o").replace("o", "a")

18. Як обробити виняток ValueError, який може виникнути при використанні index(), якщо підрядок не знайдено?

19. Поясніть, як працює шифр Цезаря. Напишіть формулу для зсуву символу на N позицій з урахуванням зациклювання алфавіту.

20. Які методи рядків використали б для перевірки валідності електронної пошти (базова перевірка)? Опишіть алгоритм.

21. Знайдіть помилки у наступному коді та виправте їх:

```
text = "  Hello World  "
text.strip("Hello")
print(text)

result = "Python" + 3.9
print(result)

s = "example"
s[0] = "E"
print(s)
```

22. Напишіть код, який:

- Перевіряє, чи містить рядок лише цифри
- Замінює всі голосні літери на символ "*"
- Знаходить найдовше слово у реченні
- Перетворює дату з формату "дд-мм-рррр" у "рррр/мм/дд"

23. Проаналізуйте складність наступного алгоритму та запропонуйте оптимізацію:

```
result = ""
for word in ["Hello", "World", "Python"]:
    result += word + " "
result = result.strip()
```

ЛАБОРАТОРНА РОБОТА №7
Списки та кортежі

1. Мета

Засвоїти теоретичні знання та набути практичних навичок роботи з основними структурами даних послідовного типу в Python: списками та кортежами. Студент навчиться створювати, модифікувати, обробляти та аналізувати списки й кортежі, використовуючи вбудовані методи та операції, а також розрізняти області застосування цих колекцій.

2. Завдання

1. Оволодіти синтаксисом створення списків та кортежів.
2. Навчитися здійснювати базові операції модифікації списків: додавання, видалення, вставку елементів.
3. Застосовувати методи сортування та пошуку в списках.
4. Освоїти роботу зі зрізами (slice) для отримання підпослідовностей.
5. Набути досвіду роботи з вкладеними списками.
6. Зрозуміти основні властивості та операції з кортежами.
7. Навчитися конвертувати списки в кортежі та навпаки.
8. Застосувати техніку розпакування колекцій.

3. Короткі теоретичні відомості

У Python списки (list) та кортежі (tuple) є вбудованими типами даних, які представляють впорядковані колекції об'єктів. Вони дозволяють зберігати набори елементів різних типів.

Списки (Lists)

Список – це змінна (mutable), впорядкована послідовність елементів. Елементи в списку індексуються, починаючи з 0.

Створення списку:

```
# Пустий список
empty_list = []
empty_list_2 = list()

# Список з елементами
numbers = [1, 2, 3, 4, 5]
mixed_list = [10, "Hello", 3.14, True]
```

Додавання елементів:

- append(item) – додає елемент в кінець списку.
- insert(index, item) – вставляє елемент у вказану позицію.
- extend(iterable) – розширює список, додаючи всі елементи з ітерабельного об’єкта.

```
fruits = ["apple", "banana"]
fruits.append("orange")        # ["apple", "banana", "orange"]
fruits.insert(1, "kiwi")         # ["apple", "kiwi", "banana",
"orange"]
fruits.extend(["grape", "mango"])      #   ["apple",   "kiwi",
"banana", "orange", "grape", "mango"]
```

Видалення елементів:

- remove(item) – видаляє перший елемент з вказаним значенням.
- pop([index]) – видаляє елемент за індексом (або останній) та повертає його.
- del – оператор для видалення елемента або зрізу.

```
numbers = [1, 2, 3, 4, 5, 4]
numbers.remove(4)       # Видаляє першу 4: [1, 2, 3, 5, 4]
popped = numbers.pop(1)  # Видаляє елемент з індексом 1 (2):
[1, 3, 5, 4], popped = 2
del numbers[0]          # Видаляє перший елемент: [3, 5, 4]
del numbers[1:3]        # Видаляє зріз: [3]
```

Сортування:

- sort(reverse=False) – сортує список на місці.
- sorted(iterable, reverse=False) – повертає новий відсортований список.
- reverse() – змінює порядок елементів у списку на зворотний.

```
data = [5, 2, 8, 1]
data.sort()                 # [1, 2, 5, 8]
data.reverse()              # [8, 5, 2, 1]
new_sorted = sorted(data)   # [1, 2, 5, 8], data залишається
[8, 5, 2, 1]
```

Пошук елементів:

- index(item) – повертає індекс першого входження елемента.
- count(item) – повертає кількість входжень елемента.
- in – оператор перевірки наявності елемента.

```
letters = ['a', 'b', 'c', 'a', 'd']
idx = letters.index('c')    # 2
cnt = letters.count('a')    # 2
```

```
is_present = 'b' in letters # True
```

Зрізи (Slicing):

Синтаксис: list[start:stop:step]. Дозволяє отримувати підсписки.

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
part = numbers[2:6]     # [2, 3, 4, 5]
even = numbers[::2]     # [0, 2, 4, 6, 8]
reversed_copy = numbers[::-1]   # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Вкладені списки:

Списки можуть містити інші списки, утворюючи багатовимірні структури.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
element = matrix[1][2]  # 6 (другий рядок, третій стовпець)
```

Кортежі (Tuples)

Кортеж – це незмінна (immutable), впорядкована послідовність. Часто використовується для даних, які не повинні змінюватися.

Створення кортежу:

```
empty_tuple = ()
single_tuple = (5,)    # Важливо: кома для кортежу з одного елемента
coordinates = (10, 20)
mixed_tuple = (1, "text", 3.14)
```

Кортежі підтримують індексацію, зрізи, методи count() та index(), але не мають методів для зміни вмісту.

```
point = (3, 7, 5)
x = point[0]          # 3
sub = point[1:]       # (7, 5)
cnt = point.count(7)  # 1
idx = point.index(5)  # 2
```

Конвертація list ↔ tuple:

```
list_data = [1, 2, 3]
tuple_data = tuple(list_data)  # (1, 2, 3)
new_list = list(tuple_data)    # [1, 2, 3]
```

Розпакування колекцій:

Позволяє присвоїти елементи послідовності окремим змінним.

```
coordinates = (4, 9)
```

```
x, y = coordinates  # x = 4, y = 9

values = [10, 20, 30]
first, *rest = values  # first = 10, rest = [20, 30]
```

Порівняння списків та кортежів:

Характеристика	Список (list)	Кортеж (tuple)
Змінність (mutable)	Так	Ні
Швидкість доступу	Повільніше	Швидше
Пам'ять	Більше	Менше
Типове використання	Динамічні дані, колекції, що змінюються	Константні дані, ключі словників, запису

Розуміння цих структур та їх властивостей є фундаментальним для ефективного програмування на Python, оскільки вони широко використовуються для організації та обробки даних.

4. Методичні рекомендації

Нижче наведено розв'язок типових задач, які демонструють роботу зі списками та кортежами.

Задача 1. Формування та базова обробка списку чисел

Користувач вводить через пробіл послідовність цілих чисел. Створіть список з цих чисел. Знайдіть суму, мінімум, максимум та середнє арифметичне елементів списку. Результати виведіть на екран.

```
# Отримуємо введення від користувача та розбиваємо його на окремі рядки
input_string = input("Введіть числа через пробіл: ")
# Перетворюємо кожен підрядок на ціле число та формуємо список
numbers = [int(num) for num in input_string.split()]

# Обчислюємо основні статистики
total_sum = sum(numbers)
minimum_value = min(numbers)
maximum_value = max(numbers)
```

```
    average_value = total_sum / len(numbers) if numbers else 0  # Уникаємо ділення на нуль

    # Виводимо результати
    print(f"Список: {numbers}")
    print(f"Сума: {total_sum}")
    print(f"Мінімальне значення: {minimum_value}")
    print(f"Максимальне значення: {maximum_value}")
    print(f"Середнє арифметичне: {average_value:.2f}")  # Форматування до двох знаків
```

Приклад вхідних та вихідних даних:

```
Вхід: 1 2 3 4 5 → Вихід: Список: [1, 2, 3, 4, 5], Сума: 15, Мінімальне: 1, Максимальне: 5, Середнє: 3.00
Вхід: 10 -5 0 7 → Вихід: Список: [10, -5, 0, 7], Сума: 12, Мінімальне: -5, Максимальне: 10, Середнє: 3.00
Вхід: (пустий рядок) → Вихід: Список: [], Сума: 0, Мінімальне: (помилка, бо список порожній), Максимальне: (помилка), Середнє: 0.00
```

Коментарі:

У коді використано list comprehension для ефективного перетворення рядків на числа.

Важливо обробляти випадок порожнього списку, щоб уникнути помилок при обчисленні min, max та діленні.

Метод .split() без аргументів розбиває рядок за будь-якою кількістю пробілів.

Задача 2. Видалення дублікатів зі збереженням порядку

Користувач вводить через пробіл послідовність слів (рядків). Створіть список, видаліть усі дублікати слів, зберігаючи початковий порядок першого входження кожного слова. Результат виведіть на екран.

```
    # Отримуємо список слів від користувача
    words = input("Введіть слова через пробіл: ").split()

    unique_words = []  # Створюємо порожній список для унікальних слів
    for word in words:
        # Додаємо слово до результату, лише якщо воно ще не зустрічалося
        if word not in unique_words:
```

```
            unique_words.append(word)

    # Виводимо результат
    print("Список без дублікатів:", unique_words)
```

Приклад вхідних та вихідних даних:

Вхід: hello world hello python world → Вихід: ['hello', 'world', 'python']
Вхід: apple banana apple orange banana apple → Вихід: ['apple', 'banana', 'orange']
Вхід: один два три один → Вихід: ['один', 'два', 'три']

Коментарі:

Це класичне завдання на фільтрацію зі збереженням порядку. Алгоритм має часову складність $O(n^2)$ через оператор in у циклі, що прийнятно для невеликих списків.

Для великих обсягів даних ефективніше використовувати словник або множину для відстеження унікальних елементів, але зі збереженням порядку це потребує додаткових зусиль.

Задача 3. Обробка матриці (вкладеного списку)

Дано матрицю (список списків) цілих чисел розміром 3x3 (задати в коді). Знайдіть суму елементів головної діагоналі (зліва направо) та суму елементів побічної діагоналі (справа наліво). Результати виведіть.

```
    # Задаємо матрицю 3x3
    matrix = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    main_diag_sum = 0
    secondary_diag_sum = 0
    n = len(matrix)  # Розмір матриці (кількість рядків/стовпців)

    for i in range(n):
        # Сума головної діагоналі: елементи з однаковими індексами
        main_diag_sum += matrix[i][i]
        # Сума побічної діагоналі: елементи, де стовпець = n - 1 - рядок
        secondary_diag_sum += matrix[i][n - 1 - i]

    print("Матриця:")
```

```
for row in matrix:
    print(row)
print(f"Сума головної діагоналі: {main_diag_sum}")
print(f"Сума побічної діагоналі: {secondary_diag_sum}")
```

Приклад вхідних та вихідних даних:

```
Для заданої матриці вихід:
Матриця:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Сума головної діагоналі: 15
Сума побічної діагоналі: 15
Якщо змінити матрицю на [[2, 4, 6], [1, 3, 5], [7, 8, 9]], то
вихід: Головна діагональ: 14, Побічна діагональ: 16.
```

Коментарі:

Головна діагональ – це елементи з однаковими індексами рядка та стовпця.

Для побічної діагоналі індекс стовпця обчислюється як n - 1 - i.

Важливо переконатися, що матриця квадратна, інакше алгоритм не працюватиме коректно.

Задача 4. Робота з кортежами: обмін значень та конвертація

Дано два кортежі tuple_a = (1, 2, 3) та tuple_b = (‘a’, ‘b’, ‘c’). Виконайте наступні дії: 1) конвертуйте їх у списки; 2) обміняйте місцями їх перші елементи; 3) конвертуйте списки назад у кортежі. Виведіть вихідні та отримані кортежі.

```
# Вихідні кортежі
tuple_a = (1, 2, 3)
tuple_b = (‘a’, ‘b’, ‘c’)

print("Вихідні кортежі:")
print(f"tuple_a = {tuple_a}")
print(f"tuple_b = {tuple_b}")

# Конвертація кортежів у списки
list_a = list(tuple_a)
list_b = list(tuple_b)

# Обмін першими елементами
```

```
list_a[0], list_b[0] = list_b[0], list_a[0]

# Конвертація списків назад у кортежі
new_tuple_a = tuple(list_a)
new_tuple_b = tuple(list_b)

print("\nКортежі після обміну першими елементами:")
print(f"new_tuple_a = {new_tuple_a}")
print(f"new_tuple_b = {new_tuple_b}")
```

Приклад вхідних та вихідних даних:

```
Вихідні кортежі:
tuple_a = (1, 2, 3)
tuple_b = ('a', 'b', 'c')
Кортежі після обміну:
new_tuple_a = ('a', 2, 3)
new_tuple_b = (1, 'b', 'c')
Якщо змінити вихідні кортежі на (10, 20) та ('x', 'y'), то
вихід: new_tuple_a = ('x', 20), new_tuple_b = (10, 'y').
```

Коментарі:

Демонструє незмінність кортежів: щоб змінити елемент, потрібно конвертувати кортеж у змінний список.

Показує механізм обміну значень у Python без використання тимчасової змінної.

Конвертація tuple ↔ list виконується за допомогою конструкторів tuple() та list().

Задача 5. Використання зрізів для перетворення списку

Дано список цілих чисел. Створіть новий список, який містить: 1) всі елементи з парними індексами; 2) всі елементи у зворотному порядку; 3) кожен другий елемент, починаючи з другого. Виведіть всі три результати.

```
# Вихідний список
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Елементи з парними індексами (починаючи з 0)
even_indices = numbers[::2]

# Список у зворотному порядку
reversed_list = numbers[::-1]
```

```
# Кожен другий елемент, починаючи з другого (індекс 1)
every_second = numbers[1::2]

print("Вихідний список:", numbers)
print("Елементи з парними індексами:", even_indices)
print("Список у зворотному порядку:", reversed_list)
print("Кожен другий елемент (з індексу 1):", every_second)
```

Приклад вхідних та вихідних даних:

```
Для [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
  Елементи з парними індексами: [0, 2, 4, 6, 8]
  Зворотний порядок: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
  Кожен другий: [1, 3, 5, 7, 9]
Для [10, 20, 30, 40, 50]:
  Елементи з парними індексами: [10, 30, 50]
  Зворотний порядок: [50, 40, 30, 20, 10]
  Кожен другий: [20, 40]
```

Коментарі:

Зріз [start:stop:step] – потужний інструмент для отримання підпослідовностей.

Крок 2 дає кожен другий елемент, -1 – зворотний порядок.

Зрізи завжди повертають новий список, не змінюючи оригінал.

Задача 6. Пошук та заміна елементів у списку

Користувач вводить рядок слів через пробіл. Замініть у списку всі входження певного слова (наприклад, "старе") на нове слово ("нове"). Якщо слова "старе" у списку немає, виведіть відповідне повідомлення. Реалізуйте два варіанти: 1) заміна лише першого входження; 2) заміна всіх входжень.

```
# Отримуємо список слів
words = input("Введіть слова через пробіл: ").split()
old_word = input("Введіть слово для заміни: ")
new_word = input("Введіть нове слово: ")

# Варіант 1: Заміна першого входження
if old_word in words:
    # Знаходимо індекс першого входження
    index = words.index(old_word)
    # Замінюємо елемент за знайденим індексом
    words[index] = new_word
```

```
        print(f"Після заміни першого входження: {words}")
    else:
        print(f"Слово '{old_word}' не знайдено у списку.")

    # Варіант 2: Заміна всіх входжень
    # Створюємо новий список, де замінюємо всі входження
    all_replaced = [new_word if word == old_word else word for
word in words]
    print(f"Після заміни всіх входжень: {all_replaced}")
```

Приклад вхідних та вихідних даних:

```
Вхід: слова=apple banana apple orange, старе=apple, нове=pear
→
  Після заміни першого: ['pear', 'banana', 'apple', 'orange']
  Після заміни всіх: ['pear', 'banana', 'pear', 'orange']
Вхід: слова=one two three, старе=four, нове=five →
  Слово 'four' не знайдено...
  Після заміни всіх: ['one', 'two', 'three'] (без змін)
```

Коментарі:

Метод .index() викличе ValueError, якщо елемент не знайдено, тому попередня перевірка in обов'язкова.

List comprehension у другому варіанті є більш "pythonic" способом створення модифікованого списку.

Зверніть увагу, що перший варіант змінює оригінальний список, а другий створює новий.

Задача 7. Розпакування колекцій та робота з вкладеними структурами

Дано список кортежів, де кожен кортеж містить ім'я студента та його оцінку. Знайдіть студента з найвищою оцінкою та обчисліть середню оцінку. Використайте розпакування кортежів у циклі.

```
    # Список кортежів (ім'я, оцінка)
    students = [("Анна", 85), ("Богдан", 92), ("Катерина", 78),
("Дмитро", 95)]

    # Ініціалізація змінних для пошуку максимальної оцінки
    best_student = ""
    best_score = 0
    total_score = 0

    # Обхід списку з розпакуванням кортежів
```

```
for name, score in students:
    # Додаємо оцінку до загальної суми
    total_score += score
    # Перевіряємо, чи є ця оцінка найвищою
    if score > best_score:
        best_score = score
        best_student = name

# Обчислюємо середню оцінку
average_score = total_score / len(students)

print("Список студентів:", students)
print(f"Найвища оцінка: {best_student} - {best_score}")
print(f"Середня оцінка: {average_score:.1f}")
```

Приклад вхідних та вихідних даних:

```
Для заданого списку:
Найвища оцінка: Дмитро - 95
Середня оцінка: 87.5
Якщо додати ("Олена", 88), то: Середня: 87.6
Якщо всі оцінки однакові, [("А", 80), ("Б", 80)], то буде
знайдений перший студент з такою оцінкою.
```

Коментарі:

Розпакування for name, score in students робить код чистішим та зрозумілішим, ніж робота з індексами.

Важливо ініціалізувати best_score значенням, меншим за будь-яку можливу оцінку (або першим елементом списку).

Для обчислення середнього потрібно перевести суму у float або використати ділення з майбутнім імпортом from __future__ import division.

Типові помилки і шляхи їх усунення

1. IndexError: list index out of range

Причина. Спроба звернутися до індексу, якого не існує в списку.

Рішення. Перевіряйте довжину списку len(list) перед доступом за індексом. Використовуйте цикл for element in list: замість for i in range(...):, якщо індекс не потрібен.

2. ValueError: list.remove(x): x not in list

Причина. Виклик remove() для елемента, якого немає в списку.

Рішення. Завжди перевіряйте наявність елемента оператором in перед видаленням:

```
if element in my_list:
    my_list.remove(element)
```

3. **Неправильне розуміння мутабельності**

   Причина. Спроба змінити кортеж або використання методу, який змінює список, очікуючи, що він поверне новий список.

   Рішення. Пам'ятайте:
   list.sort() змінює оригінал, sorted(list) повертає новий список.
   Кортежі незмінні – для змін конвертуйте їх у список.

4. **Плутанина між = (присвоєння) та == (порівняння)**

   Причина. Використання = у умовних операторах замість ==.

   Рішення. Уважно перевіряйте умови. IDE зазвичай попереджає про цю помилку.

5. **Проблеми зі зрізами та копіюванням**

   Причина. Змінні new_list = old_list створюють посилання на той самий об'єкт, а не копію.

   Рішення. Для повної копії списку використовуйте new_list = old_list.copy() або new_list = old_list[:].

6. **TypeError при конвертації**

   Причина. Спроба конвертувати невідповідний тип даних, наприклад int("abc").

   Рішення. Використовуйте try-except для обробки винятків або перевіряйте дані перед конвертацією.

Корисні поради

1. **Використовуйте вбудовані функції:** sum(), min(), max(), len() ефективніші за власні цикли для стандартних операцій.

2. **Зрізи – ваші друзі:** Освоїть синтаксис зрізів [start:stop:step]. Вони працюють однаково для списків, кортежів та рядків.

3. **Перевіряйте наявність елемента:** Завжди використовуйте if element in collection: перед викликом index() або remove().

4. **Розпакування спрощує код:** Замість x = point[0]; y = point[1] пишіть x, y = point.

5. **Для великих списків уникайте** list.remove() **у циклі:** Це має квадратичну складність O(n^2). Краще створюйте новий список.

6. **Конвертація типів:** Пам'ятайте про прості конструктори: list(), tuple(), set() для перетворення між типами колекцій.

7. **Налагодження:** Використовуйте print() для проміжних результатів або Python Debugger (pdb). Для списків корисний print(list_name).

8. **PEP 8:** Дотримуйтесь стилю кодування: відступи 4 пробіли, змінні в snake_case, пробіли навколо операторів.

9. **Документування:** Коментуйте не *що* робить код, а *навіщо* він це робить, особливо для нетривіальних алгоритмів.

10. **Експериментуйте в інтерактивному режимі:** Використовуйте Python REPL для швидкої перевірки роботи методів списків та кортежів.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Створення списку з користувацьким введенням

Створіть програму, яка запитує у користувача 5 цілих чисел (по одному за раз) та додає їх до списку. Після введення всіх чисел виведіть отриманий список, його довжину та суму всіх елементів.

```
Вхід: 4, 7, -2, 0, 11 → Вихід: Список: [4, 7, -2, 0, 11], Довжина: 5, Сума: 20
Вхід: 1, 2, 3, 4, 5 → Вихід: Список: [1, 2, 3, 4, 5], Довжина: 5, Сума: 15
Вхід: 0, 0, 0, 0, 0 → Вихід: Список: [0, 0, 0, 0, 0], Довжина: 5, Сума: 0
```

Завдання 1.2. Додавання елементів різними методами

Створіть порожній список. Додайте до нього число 10 за допомогою append(). Потім додайте список [20, 30] за допомогою extend(). Нарешті, вставте число 5 на початок списку за допомогою insert(). Виведіть кожен проміжний результат.

```
Вихід після кожного кроку:
1) [10]
2) [10, 20, 30]
3) [5, 10, 20, 30]
```

Завдання 1.3. Формування списку парних чисел

Задано список numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Створіть новий список, який містить лише парні числа з оригінального списку. Виведіть обидва списки.

```
Вхід: заданий список → Вихід: Оригінал: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], Парні: [2, 4, 6, 8, 10]
Якщо змінити оригінал на [11, 12, 13, 14] → Вихід: [12, 14]
Якщо оригінал [1, 3, 5] → Вихід: []
```

Завдання 1.4. Поєднання двох списків

Дано два списки: list1 = ["a", "b", "c"] та list2 = [1, 2, 3]. Створіть новий список, який по черзі містить елементи з обох списків: спочатку перший елемент з list1, потім перший з list2, другий з list1, другий з list2 і т.д.

```
Вхід: задані списки → Вихід: ["a", 1, "b", 2, "c", 3]
```

```
Вхід: list1 = ["x", "y"], list2 = [10, 20, 30] → Вихід: ["x", 10, "y", 20]
Вхід: list1 = [], list2 = [7, 8] → Вихід: []
```

Завдання 1.5. Видалення за значенням та індексом

Дано список items = ["яблуко", "банан", "апельсин", "ківі", "банан", "вишня"]. Видаліть перше входження "банан" за допомогою remove(). Потім видаліть елемент з індексом 2 за допомогою pop() і виведіть видалений елемент. Виведіть кінцевий список.

```
Вихід: Видалено за pop(): ківі, Кінцевий список: ["яблуко", "апельсин", "банан", "вишня"]
Якщо спробувати remove("груша") (не існує) → обробіть цей випадок повідомленням.
```

Завдання 1.6. Очищення та копіювання списку

Створіть список original = [1.5, 2.5, 3.5, 4.5]. Створіть його повну копію copied. Потім очистіть оригінальний список за допомогою clear(). Виведіть обидва списки, щоб продемонструвати, що copied залишився незмінним.

```
Вихід: Оригінал після clear(): [], Копія: [1.5, 2.5, 3.5, 4.5]
Вхід: original = ["a", "b"] → Вихід: Оригінал: [], Копія: ["a", "b"]
Вхід: original = [] → Вихід: Оригінал: [], Копія: []
```

Завдання 1.7. Підрахунок входжень

Користувач вводить рядок слів через пробіл. Підрахуйте, скільки разів кожне унікальне слово зустрічається у списку. Виведіть результат для кожного слова у форматі "слово: кількість".

```
Вхід: hello world hello python world → Вихід: hello: 2, world: 2, python: 1
Вхід: так так ні ні може бути → Вихід: так: 2, ні: 2, може: 1, бути: 1
Вхід: один → Вихід: один: 1
```

Завдання 1.8. Створення та доступ до кортежу

Створіть кортеж із трьох різних типів даних: цілого числа, рядка та дробового числа. Виведіть кортеж цілком, потім кожен елемент окремо за індексом. Спробуйте змінити перший елемент кортежу (що призведе до помилки - закоментуйте цей рядок).

```
Вихід:
Весь кортеж: (10, 'text', 3.14)
Перший елемент: 10
Другий елемент: text
Третій елемент: 3.14
Спроба змінити: TypeError
```

Завдання 1.9. Конвертація між списком та кортежем

Дано кортеж data_tuple = (100, 200, 300, 400, 500). Конвертуйте його у список. Додайте до списку число 600. Потім конвертуйте змінений список назад у кортеж. Виведіть усі проміжні та кінцеві результати.

```
Вихід:
Оригінальний кортеж: (100, 200, 300, 400, 500)
Список після конвертації: [100, 200, 300, 400, 500]
Список після додавання: [100, 200, 300, 400, 500, 600]
Новий кортеж: (100, 200, 300, 400, 500, 600)
```

Завдання 1.10. Базове розпакування кортежу

Створіть кортеж з координатами точки у 3D-просторі: point = (3, 7, -2). Розпакуйте кортеж у три окремі змінні x, y, z. Обчисліть відстань від точки до початку координат за формулою $\sqrt{(x^2 + y^2 + z^2)}$ (використайте ** 0.5 для кореня). Виведіть координати та результат.

```
Вихід: Координати: x=3, y=7, z=-2, Відстань: 7.87
Вхід: point = (1, 0, 0) → Вихід: Координати: x=1, y=0, z=0,
Відстань: 1.0
Вхід: point = (2, 3, 6) → Вихід: Координати: x=2, y=3, z=6,
Відстань: 7.0
```

Частина 2. Середні завдання

Завдання 2.1. Сортування за різними критеріями

Дано список кортежів, де кожен кортеж містить назву міста та його населення (у тисячах):

cities = [("Київ", 2800), ("Львів", 720), ("Одеса", 990), ("Харків", 1440), ("Дніпро", 980)].

Відсортуйте список: 1) за назвою міста (за алфавітом); 2) за населенням (за спаданням). Виведіть обидва відсортовані варіанти.

```
Вихід 1: [('Дніпро', 980), ('Київ', 2800), ('Львів', 720),
('Одеса', 990), ('Харків', 1440)]
Вихід 2: [('Київ', 2800), ('Харків', 1440), ('Одеса', 990),
('Дніпро', 980), ('Львів', 720)]
Якщо додати ("Бердичів", 75) → в алфавітному порядку буде
першим.
```

Завдання 2.2. Пошук елементів за умовою

Користувач вводить список цілих чисел через пробіл. Знайдіть індекси всіх елементів, які більші за середнє арифметичне всього списку. Якщо таких елементів немає, виведіть відповідне повідомлення.

```
Вхід: 10 20 30 40 50 → Вихід: Середнє: 30.0, Індекси елементів
> середнього: [3, 4] (елементи 40, 50)
Вхід: 5 5 5 5 → Вихід: Середнє: 5.0, Немає елементів більших
за середнє
Вхід: 1 2 3 4 10 → Вихід: Середнє: 4.0, Індекси: [4] (тільки
10)
```

Завдання 2.3. Робота зі зрізами

Дано список: numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. Використовуючи тільки зрізи (slice), отримайте та виведіть:

1. Останні 5 елементів
2. Кожен третій елемент, починаючи з другого
3. Список у зворотному порядку без першого та останнього елементів
4. Елементи з індексами 5-9 включно у зворотному порядку

```
Вихід:
1) [10, 11, 12, 13, 14]
2) [1, 4, 7, 10, 13]
3) [13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
4) [9, 8, 7, 6, 5]
```

Завдання 2.4. Обробка матриці (вкладеного списку)

Дано матрицю 4×4 у вигляді вкладеного списку:

```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
]
```

Знайдіть: 1) суму всіх елементів матриці; 2) суму елементів кожного рядка окремо; 3) максимальний елемент кожного стовпця. Результати виведіть у читабельному форматі.

```
Вихід:
Загальна сума: 136
Сума рядків: [10, 26, 42, 58]
Максимуми стовпців: [13, 14, 15, 16]
Для матриці 2x2 [[1,2],[3,4]]: Загальна сума: 10, Сума рядків:
[3, 7], Максимуми стовпців: [3, 4]
```

Завдання 2.5. Об'єднання та розділення списків

Користувач вводить через пробіл послідовність чисел. Створіть два списки: перший містить елементи з парними індексами, другий - з непарними. Потім об'єднайте ці два списки назад у один, але в такому порядку: спочатку всі елементи з другого списку, потім всі з першого. Виведіть усі етапи.

```
Вхід: 10 20 30 40 50 60 →
Вихід: Парні індекси: [10, 30, 50], Непарні індекси: [20, 40,
60], Об'єднаний: [20, 40, 60, 10, 30, 50]
Вхід: 1 2 3 → Вихід: Парні: [1, 3], Непарні: [2], Об'єднаний:
[2, 1, 3]
Вхід: 7 → Вихід: Парні: [7], Непарні: [], Об'єднаний: [7]
```

Частина 3. Складні завдання

Завдання 3.1. Система обліку товарів на складі

Реалізуйте просту систему обліку товарів. Кожен товар представлений кортежем (код_товару, назва, кількість, ціна). Дано список товарів:

```
inventory = [
    (101, "Ноутбук", 5, 25000),
    (102, "Монітор", 12, 5000),
    (103, "Клавіатура", 25, 800),
    (104, "Мишка", 30, 400)
]
```

Програма повинна:

1. Знайти товар з найбільшою кількістю одиниць на складі
2. Обчислити загальну вартість кожного товару (кількість × ціна)
3. Вивести список товарів, вартість яких перевищує 10000 грн

4. Знайти середню ціну товару на складі

5. Виведіть результати у форматі таблиці.

```
Вихід:
Товар з найбільшою кількістю: Мишка (30 од.)
Загальна вартість кожного товару:
Ноутбук: 125000 грн
Монітор: 60000 грн
Клавіатура: 20000 грн
Мишка: 12000 грн
Товари з вартістю > 10000 грн: Ноутбук, Монітор, Клавіатура
Середня ціна товару: 7800.0 грн
```

Завдання 3.2. Аналіз результатів іспиту

Дано список кортежів, де кожен кортеж містить ім'я студента, його вік та бал за іспит:

students = [("Анна", 20, 85), ("Богдан", 22, 92), ("Катерина", 19, 78), ("Дмитро", 21, 95), ("Олена", 20, 88), ("Федір", 22, 62)].

Програма повинна:

1. Розділити студентів на дві групи: молодші за 21 рік та старші/рівні 21

2. Для кожної групи обчислити середній бал

3. Знайти студента з найвищим балом у кожній групі

4. Вивести список студентів, чий бал вище за загальний середній бал усіх студентів

5. Посортувати студентів за віком (зростання), а при однаковому віці - за балом (спадання)

6. Виведіть всі результати структуровано.

```
Вихід (приблизний):
Група <21 років (середній бал: 81.67):
Найкращий: Катерина - 78
Група >=21 років (середній бал: 83.0):
Найкращий: Дмитро - 95
Студенти вище загального середнього (83.33): Богдан, Дмитро, Олена
Сортування за віком/балом: [("Катерина", 19, 78), ("Анна", 20, 85), ("Олена", 20, 88), ("Дмитро", 21, 95), ("Богдан", 22, 92), ("Федір", 22, 62)]
```

Завдання 3.3. Шифрування та розшифрування повідомлення

Реалізуйте простий шифр на основі списків. Програма повинна:

1. Прийняти від користувача рядок (латинські літери та пробіли)
2. Перетворити кожен символ у його ASCII-код (використайте ord())
3. Зсунути кожен код на певне значення (наприклад, +5)
4. Розбити отриманий список кодів на блоки по 4 числа (останній блок може бути неповним)
5. Для розшифрування виконати зворотні операції (зсув -5, перетворення кодів у символи chr())
6. Реалізуйте функціонал шифрування та розшифрування. Протестуйте на різних вхідних данах.

```
Вхід: "Hello World" зі зсувом 5 →
Шифрування:
Коди: [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
Зсунуті коди: [77, 106, 113, 113, 116, 37, 92, 116, 119, 113,
105]
Блоки по 4: [[77, 106, 113, 113], [116, 37, 92, 116], [119,
113, 105]]
Розшифрування: "Hello World"
Вхід: "Python" зі зсувом 3 → Розшифрування має повернути
"Python"
Вхід: "Test 123" (з пробілом) → має коректно обробляти пробіли
та цифри
```

6. Питання для самоперевірки

1. Яка основна відмінність між списком (list) і кортежем (tuple) у Python? Наведіть приклади ситуацій, коли краще використовувати кожен з них.
2. Поясніть різницю між методами append(), extend() та insert(). Наведіть приклади їх використання.
3. Чим відрізняються методи remove(), pop() та del для видалення елементів зі списку? У яких випадках виникають винятки при їх використанні?
4. Яка різниця між методами sort() та sorted()? Чому sort() не працює з кортежами?
5. Опишіть синтаксис зрізів (slice) у Python. Що поверне вираз my_list[2:7:2], якщо my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]?
6. Як працюють методи index() та count()? Що станеться, якщо викликати index() для елемента, якого немає в списку?

7. Як конвертувати список у кортеж і навпаки? Чи змінюються оригінальні об’єкти після такої конвертації?

8. Що таке вкладений список? Як отримати доступ до елемента вкладених списків? Наведіть приклад для матриці 3x3.

9. Поясніть концепцію розпакування (unpacking) колекцій. Як працює оператор * при розпакуванні списків?

10. Які операції можна виконувати з кортежами, а які ні? Чому кортежі вважаються "безпечнішими" за списки?

11. Що поверне вираз [1, 2, 3] + [4, 5]? А що поверне [1, 2, 3] * 3?

12. Як створити глибоку копію списку, щоб зміни в копії не впливали на оригінал? Чим відрізняється copy() від deepcopy()?

13. Що таке list comprehension? Наведіть приклад створення списку квадратів чисел від 1 до 10.

14. Як перевірити, чи містить список певний елемент? Які є способи і який з них найефективніший?

15. Чому при створенні кортежу з одного елемента потрібно ставити кому після нього: (5,)? Що поверне type((5))?

16. Як отримати останній елемент списку двома різними способами? Який спосіб вважається більш "pythonic"?

17. Що таке негативні індекси у списках та кортежах? Наведіть приклади їх використання.

18. Як можна "перевернути" список (змінити порядок елементів на зворотний) двома різними способами?

19. Чи можна мати список як ключ у словнику? А кортеж? Поясніть чому.

20. Які переваги та недоліки використання списків порівняно з кортежами з точки зору продуктивності та використання пам’яті?

ЛАБОРАТОРНА РОБОТА №8
Словники та множини

1. Мета

Оволодіти практичними навичками роботи з основними структурами даних Python - словниками (dict) та множинами (set). Закріпити теоретичні знання про принципи організації, створення та маніпулювання цими колекціями. Навчитися застосовувати словники для зберігання пар "ключ-значення" та множини для роботи з унікальними елементами. Опанувати методи доступу, модифікації, ітерації та виконання операцій над цими структурами для розв'язання практичних задач обробки даних.

2. Завдання

1. Зрозуміти теоретичні основи роботи зі словниками та множинами.
2. Навчитися створювати, заповнювати та модифікувати словники різними способами.
3. Опанувати методи доступу до елементів словника ([], .get()).
4. Закріпити навички додавання, оновлення та видалення елементів зі словників.
5. Навчитися здійснювати ітерацію по словниках (за ключами, значеннями та парами).
6. Розібратися з принципами роботи з вкладеними словниками.
7. Навчитися створювати множини та виконувати над ними основні операції: об'єднання, перетин, різниця.
8. Застосовувати множини для перевірки унікальності даних.
9. Використовувати словники для підрахунку частоти елементів у послідовності.
10. Розв'язати серію практичних задач різного рівня складності для закріплення матеріалу.

3. Короткі теоретичні відомості

3.1. Словники (Dictionaries)

Словник (dict) – це впорядкована (починаючи з Python 3.7), змінна та індексована за *ключем* колекція даних. Вона призначена для

зберігання пар об'єктів у форматі **ключ: значення**. Ключ виступає унікальним ідентифікатором для доступу до відповідного значення.

Створення словника:

```
# Пустий словник
student = {}
student = dict()

# Словник з початковими даними
student = {
    "name": "Anna",
    "age": 20,
    "university": "NULP",
    "courses": ["Programming", "Math"]
}

# Використання конструктора dict()
grades = dict(Physics=90, Algorithms=85)
```

Доступ до елементів:

```
# Через квадратні дужки []
name = student["name"]  # "Anna"

# Метод .get() - безпечніший, повертає None або задане значення за замовчуванням, якщо ключа немає
age = student.get("age")  # 20
phone = student.get("phone")  # None
phone = student.get("phone", "Not specified")  # "Not specified"
```

Додавання та зміна елементів: оскільки словник – змінний об'єкт, його вміст можна оновлювати.

```
 Додавання нової пари
student["group"] = "KN-101"

# Зміна значення за ключем
student["age"] = 21

# Оновлення декількох значень одночасно
student.update({"age": 22, "city": "Lviv"})
```

Видалення елементів:

```
# Оператор del
del student["city"]

# Метод .pop() - видаляє та повертає значення
removed_age = student.pop("age")

# Метод .clear() - повністю очищає словник
grades.clear()
```

Ітерація по словнику:

```
# За ключами (за замовчуванням)
for key in student:
    print(key)

# Явно за ключами
for key in student.keys():
    print(f"Key: {key}")

# За значеннями
for value in student.values():
    print(f"Value: {value}")

# За парами (ключ, значення)
for key, value in student.items():
    print(f"{key}: {value}")
```

Вкладені словники: словники можуть містити інші словники, списки тощо.

```
university = {
    "faculty": {
        "name": "Computer Sciences",
        "head": "Dr. Ivanov"
    },
    "departments": ["Software Engineering", "Data Science"],
    "address": {
        "city": "Lviv",
        "street": "Universytetska",
        "building": 1
    }
}
# Доступ до вкладених даних
print(university["faculty"]["name"])  # "Computer Sciences"
```

3.2. Множини (Sets)

Множина (set) – це невпорядкована, змінна колекція *унікальних* та *незмінних* об'єктів. Вона використовується для видалення дублікатів, перевірки членства та виконання математичних операцій над множинами.

Створення множини:

```
# Пуста множина (увага: {} створює словник!)
unique_numbers = set()

# Множина з елементів
programming_languages = {"Python", "Java", "C++", "Python"}  # Дублікат буде видалено
fruits = set(["apple", "banana", "apple", "orange"])
```

Основні операції над множинами:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Об'єднання (union) -> елементи, які є в A або B
union_set = A | B  # {1, 2, 3, 4, 5, 6}
union_set = A.union(B)

# Перетин (intersection) -> елементи, які є і в A, і в B
intersection_set = A & B  # {3, 4}
intersection_set = A.intersection(B)

# Різниця (difference) -> елементи, які є в A, але не в B
difference_set = A - B  # {1, 2}
difference_set = A.difference(B)

# Симетрична різниця (symmetric difference) -> елементи, які є або в A, або в B, але не в обох
sym_diff_set = A ^ B  # {1, 2, 5, 6}
sym_diff_set = A.symmetric_difference(B)
```

Перевірка унікальності: Множина – найпростіший інструмент для видалення дублікатів зі списку.

```
data_list = [10, 20, 30, 20, 40, 10]
unique_data = set(data_list)  # {40, 10, 20, 30}
list_without_duplicates = list(unique_data)  # [40, 10, 20, 30]
```

3.3. Підрахунок частоти елементів

Поєднання словників та множин є потужним інструментом для аналізу даних. Класичним прикладом є підрахунок частоти входження елементів у послідовність (гістограма).

```
text = "hello world hello python"
words = text.split()  # ['hello', 'world', 'hello', 'python']

word_count = {}
for word in words:
    # Якщо слова ще немає в словнику, ініціалізуємо лічильник нулем
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)  # {'hello': 2, 'world': 1, 'python': 1}
```

Ці базові знання є фундаментом для успішного виконання лабораторної роботи. Уважно вивчіть наведені приклади, перш ніж переходити до практичних завдань.

4. Методичні рекомендації

Задача 1. Підрахунок частоти символів у рядку

Напишіть програму, яка приймає рядок від користувача та підраховує, скільки разів кожен символ зустрічається у цьому рядку. Результат вивести у вигляді словника, де ключ – символ, значення – кількість його входжень.

```
# Отримуємо вхідний рядок від користувача
user_input = input("Введіть рядок: ")
# Створюємо порожній словник для зберігання підрахунків
char_count = {}

# Проходимо по кожному символу у рядку
for char in user_input:
    # Якщо символ вже є у словнику, збільшуємо лічильник на 1
    # Якщо немає - додаємо зі значенням 1
    if char in char_count:
        char_count[char] += 1
    else:
        char_count[char] = 1

# Виводимо результат
```

```
print("Частота символів:")
print(char_count)
```

Приклад вхідних та вихідних даних:

```
Вхід: "hello" → Вихід: {'h': 1, 'e': 1, 'l': 2, 'o': 1}
Вхід: "banana" → Вихід: {'b': 1, 'a': 3, 'n': 2}
Вхід: "a b c a b" → Вихід: {'a': 2, ' ': 4, 'b': 2, 'c': 1}
```

Коментарі:

Для перевірки наявності ключа використано оператор in

Важливо враховувати пробіли як окремі символи

Порядок елементів у словнику може відрізнятися, оскільки до Python 3.7 словники були невпорядкованими

Задача 2. Об'єднання двох словників з оновленням значень

Напишіть програму, яка об'єднує два словники. Якщо ключ присутній в обох словниках, у підсумковому словнику має залишитись значення з другого словника.

```
# Задаємо два словники для об'єднання
dict1 = {"a": 1, "b": 2, "c": 3}
dict2 = {"b": 20, "c": 30, "d": 40}

# Створюємо копію першого словника
result = dict1.copy()

# Додаємо всі елементи з другого словника
# Якщо ключ вже є, значення буде перезаписано
for key, value in dict2.items():
    result[key] = value

# Виводимо результат
print("Перший словник:", dict1)
print("Другий словник:", dict2)
print("Об'єднаний словник:", result)
```

Приклад вхідних та вихідних даних:

```
Вхід:
  dict1 = {"a": 1, "b": 2, "c": 3}
  dict2 = {"b": 20, "c": 30, "d": 40}
Вихід: {"a": 1, "b": 20, "c": 30, "d": 40}
Вхід:
  dict1 = {"x": 10, "y": 20}
```

```
  dict2 = {"y": 25, "z": 35}
Вихід: {"x": 10, "y": 25, "z": 35}
Вхід:
  dict1 = {}
  dict2 = {"name": "John", "age": 25}
Вихід: {"name": "John", "age": 25}
```

Коментарі:

Використання методу .copy() забезпечує, що оригінальний словник не зміниться

Цикл for key, value in dict2.items() дозволяє отримати одночасно ключ і значення

При збігу ключів значення з другого словника перезаписує значення з першого

Задача 3. Пошук спільних елементів у двох множинах

Напишіть програму, яка знаходить спільні елементи у двох множинах (перетин множин) та повертає їх у вигляді нової множини.

```
# Створюємо дві множини
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
# Створюємо порожню множину для результатів
common_elements = set()

# Перевіряємо кожен елемент першої множини
for element in set1:
    # Якщо елемент є у другій множині, додаємо його до результатів
    if element in set2:
        common_elements.add(element)

# Виводимо результати
print("Перша множина:", set1)
print("Друга множина:", set2)
print("Спільні елементи:", common_elements)

# Альтернативний спосіб (для порівняння)
print("Спільні елементи (за допомогою оператора &):", set1 & set2)
```

Приклад вхідних та вихідних даних:

```
Вхід: {1, 2, 3, 4}, {3, 4, 5, 6} → Вихід: {3, 4}
```

Вхід: {"apple", "banana", "orange"}, {"banana", "kiwi", "grape"} → Вихід: {"banana"}
Вхід: {10, 20, 30}, {40, 50, 60} → Вихід: set() (порожня множина)

Коментарі:

- Метод .add() додає елемент до множини
- Оператор in перевіряє належність елемента до множини
- Можна використовувати вбудований оператор & для перетину множин
- Порожня множина відображається як set()

Задача 4. Робота з вкладеними словниками (студенти та оцінки)

Створіть програму, яка додає нову оцінку студенту за предмет. Словник містить інформацію про студентів: ключ – ім'я студента, значення – словник з предметами та оцінками.

```
# Створюємо словник з даними студентів
students = {
    "Іван Петренко": {
        "Математика": [85, 90, 88],
        "Фізика": [92, 87]
    },
    "Марія Коваленко": {
        "Математика": [91, 89, 94],
        "Програмування": [95, 96, 98]
    }
}

# Функціонал додавання нової оцінки
student_name = "Іван Петренко"
subject = "Математика"
new_grade = 93

# Перевіряємо, чи існує студент
if student_name in students:
    # Перевіряємо, чи існує предмет у студента
    if subject in students[student_name]:
        # Додаємо нову оцінку до списку
        students[student_name][subject].append(new_grade)
        print(f"Оцінку {new_grade} додано студенту {student_name} з предмету {subject}")
```

```
        else:
            # Якщо предмету немає, створюємо новий список оцінок
            students[student_name][subject] = [new_grade]
            print(f"Створено новий предмет {subject} для студента
{student_name} з оцінкою {new_grade}")
    else:
        print(f"Студента {student_name} не знайдено")

    # Виводимо оновлені дані
    print("\nОновлені дані студентів:")
    for student, subjects in students.items():
        print(f"\n{student}:")
        for subject, grades in subjects.items():
            avg_grade = sum(grades) / len(grades) if grades else 0
            print(f"  {subject}: {grades} (середня: {avg_grade:.1f})")
```

Приклад вхідних та вихідних даних:

```
Вхід: студент "Іван Петренко", предмет "Математика", оцінка
93
Вихід: Оновлений словник з доданою оцінкою 93 до списку [85,
90, 88, 93]
Вхід: студент "Іван Петренко", предмет "Хімія", оцінка 85
Вихід: Додано новий предмет "Хімія" з оцінкою [85]
Вхід: студент "Петро Сидоренко", предмет "Математика", оцінка
88
Вихід: "Студента Петро Сидоренко не знайдено"
```

Коментарі:

- Демонструє роботу з багаторівневими словниками
- Важливо перевіряти наявність ключів на кожному рівні
- Списки оцінок дозволяють зберігати кілька значень для одного предмета
- Використано форматування рядків для зручного виводу

Задача 5. Видалення дублікатів зі збереженням порядку

Напишіть програму, яка видаляє дублікати зі списку, зберігаючи початковий порядок елементів.

```
    # Початковий список з дублікатами
    original_list = ["apple", "banana", "apple", "orange",
"banana", "grape", "apple"]

    # Створюємо порожній список для результату
```

```
    unique_list = []
    # Створюємо множину для відстеження вже доданих елементів
    seen_elements = set()

    # Проходимо по всіх елементах оригінального списку
    for item in original_list:
        # Якщо елемент ще не зустрічався
        if item not in seen_elements:
            # Додаємо до списку результатів
            unique_list.append(item)
            # Додаємо до множини відстежених елементів
            seen_elements.add(item)


    # Виводимо результати
    print("Оригінальний список:", original_list)
    print("Список без дублікатів:", unique_list)
    print("Кількість дублікатів видалено:", len(original_list) -
len(unique_list))
```

Приклад вхідних та вихідних даних:

```
Вхід: [1, 2, 2, 3, 4, 4, 5] → Вихід: [1, 2, 3, 4, 5]
Вхід: ["a", "b", "a", "c", "b", "a"] → Вихід: ["a", "b", "c"]
Вхід: [10, 20, 30, 20, 10, 40] → Вихід: [10, 20, 30, 40]
```

Коментарі:

- Використано комбінацію списку та множини для ефективного розв'язання
- Множина seen_elements забезпечує швидку перевірку наявності елемента
- Список unique_list зберігає порядок елементів
- Це ефективніший спосіб, ніж перевірка if item not in unique_list для кожного елемента

Задача 6. Підрахунок слів у тексті з ігноруванням регістру

Напишіть програму, яка підраховує кількість входжень кожного слова у тексті, ігноруючи регістр букв та розділові знаки.

```
    # Вхідний текст
    text = "Hello, World! Hello, Python. World is great, Python is
awesome!"

    # Перетворюємо текст у нижній регістр
```

```
    lower_text = text.lower()

    # Створюємо порожній словник для підрахунку слів
    word_count = {}

    # Розділяємо текст на слова (розділювачі: пробіли, коми,
крапки, знаки оклику)
    # Простий спосіб: замінити розділові знаки на пробіли та
розділити
    for char in ",.!?":
        lower_text = lower_text.replace(char, " ")
    # Розділяємо на слова
    words = lower_text.split()

    # Підраховуємо слова
    for word in words:
        if word:  # Перевіряємо, що слово не порожнє
            if word in word_count:
                word_count[word] += 1
            else:
                word_count[word] = 1

    # Виводимо результати
    print("Вхідний текст:", text)
    print("\nЧастота слів (ігнорується регістр):")
    for word, count in word_count.items():
        print(f"{word}: {count}")
```

Приклад вхідних та вихідних даних:

```
Вхід: "Hello world. Hello Python. World is big."
Вихід: {'hello': 2, 'world': 2, 'python': 1, 'is': 1, 'big':
1}
Вхід: "Cat, cat, CAT, dog, Dog"
Вихід: {'cat': 3, 'dog': 2}
Вхід: "a a b b b c c c c"
Вихід: {'a': 2, 'b': 3, 'c': 4}
```

Коментарі:

- Метод .lower() перетворює весь текст у нижній регістр
- Метод .replace() замінює розділові знаки на пробіли
- Метод .split() розділяє рядок на слова за пробілами
- Важливо перевіряти, що слово не порожнє після обробки

Задача 7. Робота з множинами: операції над трьома множинами

Дано три множини. Знайдіть: 1) елементи, які є в усіх трьох множинах; 2) елементи, які є тільки в першій множині; 3) всі унікальні елементи з усіх множин.

```
# Задаємо три множини
set_a = {1, 2, 3, 4, 5}
set_b = {3, 4, 5, 6, 7}
set_c = {5, 6, 7, 8, 9}

# 1. Елементи, які є в усіх трьох множинах
common_all = set()
for element in set_a:
    if element in set_b and element in set_c:
        common_all.add(element)

# 2. Елементи, які є тільки в першій множині
only_in_a = set()
for element in set_a:
    if element not in set_b and element not in set_c:
        only_in_a.add(element)

# 3. Всі унікальні елементи з усіх множин
all_unique = set()
# Додаємо всі елементи з кожної множини
for element in set_a:
    all_unique.add(element)
for element in set_b:
    all_unique.add(element)
for element in set_c:
    all_unique.add(element)

# Виводимо результати
print("Множина A:", set_a)
print("Множина B:", set_b)
print("Множина C:", set_c)
print("\n1. Елементи в усіх трьох множинах:", common_all)
print("2. Елементи тільки в множині A:", only_in_a)
print("3. Всі унікальні елементи:", all_unique)

# Альтернативні способи (для порівняння)
print("\nАльтернативні способи:")
print("1. (A & B & C):", set_a & set_b & set_c)
```

```
print("2. (A - B - C):", set_a - set_b - set_c)
print("3. (A | B | C):", set_a | set_b | set_c)
```

Приклад вхідних та вихідних даних:

```
Вхід: A = {1, 2, 3}, B = {2, 3, 4}, C = {3, 4, 5}
Вихід:
{3}
{1}
{1, 2, 3, 4, 5}
Вхід: A = {"a", "b"}, B = {"b", "c"}, C = {"c", "d"}
Вихід:
set()
{"a"}
{"a", "b", "c", "d"}
Вхід: A = {10, 20}, B = {10, 20}, C = {10, 20}
Вихід:
{10, 20}
set()
{10, 20}
```

Коментарі:

- Демонструє логічні операції з множинами
- Показано як "вручну", так і з використанням операторів
- Операції з множинами можуть бути каскадними
- Порожня множина позначається як set()

Типові помилки і шляхи їх усунення

1. **KeyError при доступі до неіснуючого ключа в словнику**

   Помилка. KeyError: ‘key_name’

   Причина. Спроба отримати значення за ключем, якого немає в словнику за допомогою dict[key]

   Рішення. Використовувати метод .get() або перевіряти наявність ключа через in

```
# Погано:
value = my_dict["unknown_key"]  # Може викликати KeyError

# Добре:
value = my_dict.get("unknown_key")  # Поверне None, якщо ключа немає
value = my_dict.get("unknown_key", "default_value")  # Поверне значення за замовчуванням
# Або:
```

```
if "unknown_key" in my_dict:
    value = my_dict["unknown_key"]
```

2. **Зміна множини під час ітерації**

Помилка. RuntimeError: Set changed size during iteration

Причина. Модифікація множини (додавання/видалення елементів) під час перебору її елементів

Рішення. Створювати копію множини для ітерації або збирати елементи для зміни в окремому списку

```
# Погано:
for item in my_set:
    if condition:
        my_set.remove(item)  # Помилка!

# Добре:
for item in my_set.copy():  # Ітеруємо по копії
    if condition:
        my_set.remove(item)
# Або:
items_to_remove = []
for item in my_set:
    if condition:
        items_to_remove.append(item)
for item in items_to_remove:
    my_set.remove(item)
```

3. **Сплутування порожньої множини та порожнього словника**

Помилка. невірна ініціалізація структури

Причина. {} створює словник, а не множину

Рішення. Для створення порожньої множини використовувати set()

```
# Погано:
empty_set = {}  # Це словник, а не множина!
print(type(empty_set))  # <class 'dict'>

# Добре:
empty_set = set()  # Це дійсно множина
print(type(empty_set))  # <class 'set'>
```

4. **Модифікація словника під час ітерації**

Помилка. RuntimeError: dictionary changed size during iteration

Причина. Додавання або видалення ключів під час перебору словника

Рішення. Ітерувати по копії словника або списку ключів

```
# Погано:
for key in my_dict:
    if condition:
        del my_dict[key]  # Помилка!

# Добре:
for key in list(my_dict.keys()):  # Створюємо список ключів
    if condition:
        del my_dict[key]
```

5. **Забувають, що множини можуть містити тільки незмінні об'єкти**

Помилка. TypeError: unhashable type: 'list'

Причина. Спроба додати список або словник до множини

Рішення. Використовувати кортежі замість списків, якщо потрібна множина

```
# Погано:
my_set = {[1, 2], [3, 4]}  # Помилка!

# Добре:
my_set = {(1, 2), (3, 4)}  # Кортежі є незмінними
```

Корисні поради

1. **Вибір між списком, множиною та словником:**

- використовуйте **список**, коли потрібно зберігати послідовність елементів з можливістю дублікатів та важливий порядок;
- використовуйте **множину**, коли потрібна колекція унікальних елементів або швидка перевірка належності;
- використовуйте **словник**, коли потрібно встановити зв'язок між ключами та значеннями (асоціативний масив).

2. **Ефективність операцій:**

- перевірка належності елемента: множина (O(1)) набагато швидша за список (O(n));
- для видалення дублікатів зі збереженням порядку: комбінуйте список та множину;
- для підрахунку частоти: словник – найоптимальніший вибір.

3. **Стиль кодування:**

- називайте змінні описово: student_grades замість sg, word_frequency замість wf;

- використовуйте метод .get() для безпечного доступу до словників;
- для перебору пар ключ-значення використовуйте .items() замість окремого отримання ключів та значень.

4. **Налагодження:**
- використовуйте print() для виводу проміжних результатів;
- перевіряйте типи даних: print(type(variable));
- для складних вкладених структур використовуйте pprint.pprint() для кращого відображення.

5. **Робота з вкладеними структурами:**
- будьте уважні при доступі до вкладених елементів – завжди перевіряйте наявність ключів на кожному рівні;
- для глибокого копіювання вкладених словників використовуйте copy.deepcopy();
- розбивайте складні операції на простіші кроки для кращої читабельності.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Створення та зміна словника з інформацією про книгу

Створіть словник book з такими ключами: "title", "author", "year". Заповніть його довільними значеннями. Потім змініть значення ключа "year" на поточний рік і додайте новий ключ "pages" зі значенням 300.

```
Вхід: book = {"title": "Python Basics", "author": "John Smith", "year": 2020}
Вихід: {"title": "Python Basics", "author": "John Smith", "year": 2024, "pages": 300}
Вхід: book = {"title": "Algorithms", "author": "Robert Brown", "year": 2019}
Вихід: {"title": "Algorithms", "author": "Robert Brown", "year": 2024, "pages": 300}
Вхід: book = {"title": "Data Science", "author": "Anna White", "year": 2021}
Вихід: {"title": "Data Science", "author": "Anna White", "year": 2024, "pages": 300}
```

Завдання 1.2. Підрахунок кількості парних чисел у словнику

Дано словник numbers, де ключі - це рядки ("num1", "num2", тощо), а значення - цілі числа. Порахуйте, скільки парних чисел міститься у словнику.

```
Вхід: {"num1": 10, "num2": 15, "num3": 22, "num4": 37, "num5": 40} → Вихід: 3
Вхід: {"a": 1, "b": 2, "c": 3, "d": 4} → Вихід: 2
Вхід: {"x": 11, "y": 13, "z": 17} → Вихід: 0
```

Завдання 1.3. Знаходження ключа за максимальним значенням

Дано словник із числовими значеннями. Знайдіть ключ, який відповідає максимальному значенню у словнику.

```
Вхід: {"apple": 50, "banana": 30, "orange": 75, "grape": 20} → Вихід: "orange"
Вхід: {"Math": 95, "Physics": 88, "History": 92} → Вихід: "Math"
Вхід: {"January": 15, "February": 12, "March": 18} → Вихід: "March"
```

Завдання 1.4. Об’єднання двох словників з додаванням значень

Дано два словники з числовими значеннями. Об’єднайте їх так, щоб у разі спільного ключа значення додавалися, а не перезаписувалися.

```
Вхід:
  dict1 = {"a": 10, "b": 20, "c": 30}
  dict2 = {"b": 5, "c": 15, "d": 25}
Вихід: {"a": 10, "b": 25, "c": 45, "d": 25}
Вхід:
  dict1 = {"x": 100, "y": 200}
  dict2 = {"y": 50, "z": 300}
Вихід: {"x": 100, "y": 250, "z": 300}
Вхід:
  dict1 = {"cat": 3, "dog": 5}
  dict2 = {"dog": 2, "bird": 4}
Вихід: {"cat": 3, "dog": 7, "bird": 4}
```

Завдання 1.5. Фільтрація словника за значенням

Дано словник зі студентами та їх середніми балами. Створіть новий словник, який міститиме лише тих студентів, чий середній бал вищий за 75.

```
Вхід: {"Anna": 82.5, "Bohdan": 71.0, "Kateryna": 90.0,
"Mykola": 68.5, "Olena": 77.8}
Вихід: {"Anna": 82.5, "Kateryna": 90.0, "Olena": 77.8}
Вхід: {"Ivan": 60, "Maria": 85, "Petro": 74, "Natalia": 92}
Вихід: {"Maria": 85, "Natalia": 92}
Вхід: {"Alex": 70, "Diana": 65, "Serhiy": 72}
Вихід: {} (порожній словник)
```

Завдання 1.6. Створення множини з унікальних букв у слові

Дано слово. Створіть множину, яка містить усі унікальні букви цього слова (ігноруйте регістр).

```
Вхід: "програмування" → Вихід: {"п", "р", "о", "г", "а", "м",
"у", "в", "н", "і", "я"}
Вхід: "Mississippi" → Вихід: {"m", "i", "s", "p"}
Вхід: "Python" → Вихід: {"p", "y", "t", "h", "o", "n"}
```

Завдання 1.7. Перевірка, чи є всі елементи списку унікальними

Дано список елементів. Перевірте, чи всі елементи у списку є унікальними (не повторюються). Використайте множини для розв'язання.

```
Вхід: [1, 2, 3, 4, 5] → Вихід: True
Вхід: ["apple", "banana", "apple", "orange"] → Вихід: False
Вхід: [10, 20, 30, 10, 40] → Вихід: False
```

Завдання 1.8. Знаходження спільних елементів у двох списках

Дано два списки. Знайдіть всі елементи, які є в обох списках (перетин), і поверніть їх у вигляді множини.

```
Вхід:
  list1 = [1, 2, 3, 4, 5]
  list2 = [3, 4, 5, 6, 7]
Вихід: {3, 4, 5}
Вхід:
```

```
  list1 = ["a", "b", "c"]
  list2 = ["b", "c", "d", "e"]
Вихід: {"b", "c"}
Вхід:
  list1 = [10, 20, 30]
  list2 = [40, 50, 60]
Вихід: set() (порожня множина)
```

Завдання 1.9. Об'єднання множин з видаленням дублікатів

Дано три списки чисел. Об'єднайте їх у одну множину, щоб отримати всі унікальні числа з усіх списків.

```
Вхід:
  list1 = [1, 2, 3]
  list2 = [2, 3, 4]
  list3 = [3, 4, 5]
Вихід: {1, 2, 3, 4, 5}
Вхід:
  list1 = [10, 20]
  list2 = [20, 30]
  list3 = [30, 40]
Вихід: {10, 20, 30, 40}
Вхід:
  list1 = [5, 5, 5]
  list2 = [5, 5, 5]
  list3 = [5, 5, 5]
Вихід: {5}
```

Завдання 1.10. Різниця множин: що є в першій, але немає в другій

Дано дві множини. Знайдіть елементи, які є в першій множині, але відсутні в другій.

```
Вхід:
  set1 = {1, 2, 3, 4, 5}
  set2 = {3, 4, 5, 6, 7}
Вихід: {1, 2}
Вхід:
  set1 = {"apple", "banana", "orange"}
  set2 = {"banana", "kiwi", "grape"}
Вихід: {"apple", "orange"}
Вхід:
  set1 = {"a", "b", "c"}
  set2 = {"a", "b", "c"}
```

```
Вихід: set() (порожня множина)
```

Частина 2. Середні завдання

Завдання 2.1. Інвертування словника (ключ↔значення)

Напишіть програму, яка інвертує словник: ключі стають значеннями, а значення - ключами. Врахуйте, що значення у вихідному словнику можуть повторюватися. У такому разі зберігайте всі відповідні ключі у списку.

```
Вхід: {"a": 1, "b": 2, "c": 3}
Вихід: {1: "a", 2: "b", 3: "c"}
Вхід: {"apple": "fruit", "carrot": "vegetable", "banana":
"fruit"}
Вихід: {"fruit": ["apple", "banana"], "vegetable": ["carrot"]}
Вхід: {"Math": "Smith", "Physics": "Johnson", "Chemistry":
"Smith"}
Вихід: {"Smith": ["Math", "Chemistry"], "Johnson":
["Physics"]}
```

Завдання 2.2. Аналіз успішності студентів за допомогою вкладених словників

Дано словник студентів, де ключ - ім'я студента, а значення - словник з предметами та оцінками. Знайдіть:

1. Студента з найвищим середнім балом
2. Предмет з найвищою середньою оцінкою серед усіх студентів

```
Вхід:
students = {
"Anna": {"Math": [85, 90], "Physics": [88, 92]},
"Bohdan": {"Math": [78, 82], "Physics": [85, 88]},
"Kateryna": {"Math": [92, 95], "Physics": [90, 94]}
}
Вихід:
Найкращий студент: Kateryna (середній бал: 92.75)
Найкращий предмет: Physics (середня оцінка: 91.5)
Вхід:
students = {
"Ivan": {"Programming": [95, 98], "Algorithms": [88, 92]},
"Maria": {"Programming": [92, 94], "Algorithms": [85, 90]}
}
Вихід:
```

```
Найкращий студент: Ivan (середній бал: 93.25)
Найкращий предмет: Programming (середня оцінка: 94.75)
```

Завдання 2.3. Групування слів за довжиною

Дано список слів. Згрупуйте їх у словник, де ключ - довжина слова, а значення - список слів цієї довжини.

```
Вхід: ["я", "ти", "він", "вона", "ми", "ви", "вони",
"програмування", "Python"]
Вихід: {
1: ["я"],
2: ["ти", "ми", "ви"],
3: ["він"],
4: ["вона"],
5: ["вони"],
12: ["програмування"],
6: ["Python"]
}
Вхід: ["cat", "dog", "elephant", "bird", "lion", "tiger"]
Вихід: {
3: ["cat", "dog"],
8: ["elephant"],
4: ["bird", "lion"],
5: ["tiger"]
}
Вхід: ["a", "ab", "abc", "abcd"]
Вихід: {1: ["a"], 2: ["ab"], 3: ["abc"], 4: ["abcd"]}
```

Завдання 2.4. Симетрична різниця трьох множин

Дано три множини. Знайдіть елементи, які зустрічаються тільки в одній з трьох множин (симетрична різниця для трьох множин).

```
Вхід:
  A = {1, 2, 3, 4}
  B = {3, 4, 5, 6}
  C = {4, 6, 7, 8}
Вихід: {1, 2, 5, 7, 8}
Вхід:
  A = {"apple", "banana"}
  B = {"banana", "orange"}
  C = {"orange", "grape"}
Вихід: {"apple", "grape"}
Вхід:
  A = {10, 20, 30}
```

```
  B = {20, 30, 40}
  C = {30, 40, 50}
Вихід: {10, 50}
```

Завдання 2.5. Пошук унікальних елементів у списку списків

Дано список, що містить інші списки. Знайдіть всі унікальні елементи з усіх внутрішніх списків та поверніть їх у вигляді множини.

```
Вхід: [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
Вихід: {1, 2, 3, 4, 5}
Вхід: [["a", "b"], ["b", "c", "d"], ["d", "e"]]
Вихід: {"a", "b", "c", "d", "e"}
Вхід: [[10, 20], [20, 30], [10, 20, 30]]
Вихід: {10, 20, 30}
```

Частина 3. Складні завдання

Завдання 3.1. Система телефонної книги з розширеним функціоналом

Створіть програму телефонної книги, яка підтримує такі операції:

1. Додавання нового контакту (ім’я, список телефонів, email, група)
2. Пошук контакту за іменем (частковий збіг)
3. Виведення всіх контактів певної групи
4. Видалення телефону з контакту
5. Підрахунок кількості контактів у кожній групі

Структура даних: словник, де ключ - унікальний ID контакту, значення - словник з інформацією про контакт.

```
Вхід:
Додати: Іван Петренко, тел: ["+380501234567", "+380671234567"], email: "ivan@example.com", група: "Друзі"
Додати: Марія Коваленко, тел: ["+380631234567"], email: "maria@example.com", група: "Робота"
Додати: Петро Сидоренко, тел: ["+380501111111"], email: "petro@example.com", група: "Друзі"
Вихід при пошуку "Іван": Контакт знайдено: Іван Петренко
Вихід для групи "Друзі": 2 контакти
Вихід статистики: {"Друзі": 2, "Робота": 1}
```

Завдання 3.2. Система інвентаризації товарів магазину

Створіть систему для обліку товарів у магазині. Кожен товар має: код, назву, категорію, ціну, кількість на складі. Реалізуйте функції:

1. Додавання нового товару
2. Продаж товару (зменшення кількості)
3. Поповнення запасів товару
4. Пошук товарів за категорією
5. Визначення товарів, кількість яких менша за мінімальний запас (наприклад, менше 10 одиниць)
6. Підрахунок загальної вартості всіх товарів на складі

```
Вхід:
Додати: Код: "A001", Назва: "Молоко", Категорія: "Молочні",
Ціна: 30, Кількість: 50
Додати: Код: "A002", Назва: "Хліб", Категорія: "Хлібобулочні",
Ціна: 20, Кількість: 100
Продати: "A001", кількість: 10
Поповнити: "A002", кількість: 20
Вихід для категорії "Молочні": Молоко (40 шт., 30 грн.)
Вихід для товарів з низьким запасом (якщо мінімум = 15):
(список)
Загальна вартість: (розрахунок)
```

Завдання 3.3. Аналіз текстових даних з підвищеною складністю

Напишіть програму для аналізу тексту, яка:

1. Рахує частоту кожного слова (ігноруючи регістр та розділові знаки)
2. Знаходить 5 найбільш вживаних слів
3. Знаходить 5 найменш вживаних слів (що зустрічаються хоча б 2 рази)
4. Групує слова за довжиною
5. Знаходить всі унікальні символи, що зустрічаються в тексті
6. Видаляє стоп-слова (заданий список, наприклад: ["і", "та", "у", "в", "на"])

```
Вхід текст: "Програмування - це мистецтво. Програмування - це
наука. Python - мова програмування."
Вихід:
Найпопулярніші слова: програмування (3), це (2), python (1),
мистецтво (1), наука (1)
Найменш популярні (з частотою ≥2): це (2)
```

```
Групи за довжиною: {10: ["програмування"], 2: ["це"], 6:
["python", "наука"], 8: ["мистецтво"]}
Унікальні символи: (множина всіх символів)
Без стоп-слів: (текст без "це")
```

Питання для самоперевірки

1. Яка основна відмінність між словником (dict) та множиною (set) у Python? Наведіть приклади використання кожної структури.
2. Які типи даних можуть бути ключами в словнику Python? Чому список не може бути ключем?
3. Який спосіб доступу до елемента словника є безпечнішим: dict[key] чи dict.get(key)? Поясніть різницю.
4. Що станеться, якщо спробувати отримати значення за ключем, якого немає в словнику, використовуючи:
   a) dict[key]
   b) dict.get(key)
   c) dict.get(key, "default_value")
5. Які три методи використовуються для ітерації по словнику? Напишіть приклади кожного з них.
6. Чому {} створює порожній словник, а не порожню множину? Як правильно створити порожню множину?
7. Що таке вкладений словник? Наведіть приклад та покажіть, як отримати доступ до внутрішнього значення.
8. Які математичні операції можна виконувати над множинами? Напишіть оператори та методи для:
   a) Об’єднання
   b) Перетину
   c) Різниці
   d) Симетричної різниці
9. Як можна видалити дублікати зі списку зі збереженням порядку елементів? Напишіть код прикладу.
10. Як перевірити, чи містить множина певний елемент? Який час виконання цієї операції порівняно зі списком?
11. Що таке "хешований" об’єкт? Чому множина може містити тільки хешовані об’єкти?
12. Яка різниця між set.pop() та set.remove()? Що станеться, якщо викликати ці методи для порожньої множини або для неіснуючого елемента?

13. Як об’єднати два словники, якщо при спільному ключі потрібно:

a) Зберегти значення з першого словника

b) Зберегти значення з другого словника

c) Додати значення разом

14. Напишіть код, який підраховує частоту символів у рядку без використання методів count() або Counter.

15. Як можна інвертувати словник (поміняти місцями ключі та значення)? Що робити, якщо значення повторюються?

16. Що таке "симетрична різниця" множин? Наведіть приклад із трьох множин.

17. Як перевірити, чи є один словник підмножиною іншого (всі ключ-значення одного є в другому)?

18. Що таке "view objects" у контексті словників (dict.keys(), dict.values(), dict.items())? Чим вони відрізняються від списків?

19. Як можна сортувати словник:

a) За ключами

b) За значеннями

Наведіть приклади.

20. Які переваги використання множин для операцій перевірки належності порівняно зі списками? У яких практичних задачах це особливо корисно?

ЛАБОРАТОРНА РОБОТА №9
Створення та використання функцій

1. Мета

Оволодіння концепцією функцій в Python як основного інструменту структурного програмування. Формування практичних навичок у створенні, виклику та використанні функцій різних типів, застосуванні їх для декомпозиції складних завдань на простіші блоки. Закріплення розуміння областей видимості змінних та правил документування коду.

2. Завдання

1. Навчитися оголошувати та викликати функції.
2. Опанувати використання параметрів та оператора return.
3. Розрізняти та застосовувати позиційні, іменовані аргументи та параметри за замовчуванням.
4. Зрозуміти різницю між локальними та глобальними змінними.
5. Набути навичок документування функцій за допомогою docstrings.
6. Застосувати принцип декомпозиції для розбиття пракичної задачі на окремі функції.
7. Реалізувати функції для математичних обчислень та обробки даних.

3. Короткі теоретичні відомості

3.1. Означення та призначення функцій

Функція – це іменований, ізольований блок коду, призначений для виконання конкретної задачі. Основна ідея – "один раз визначити, багато разів використати". Функції покращують читабельність, структурованість коду, уникнення дублювання логіки (принцип DRY – Don’t Repeat Yourself) та полегшують налагодження.

3.2. Оголошення та виклик. Функції без параметрів та без return.

Функція оголошується за допомогою ключового слова def.

```
def greet_user():
    """Виводить вітальне повідомлення."""  # Це docstring
    print("Hello, welcome to the program!")
```

```
# Виклик функції (виконання її коду)
greet_user()  # Виведе: Hello, welcome to the program!
```

Така функція виконує дію, але не приймає вхідних даних і не повертає результату.

3.3. Функції з параметрами

Параметри – це змінні, перелічені в дужках при оголошенні функції. Вони отримують значення (аргументи) під час виклику.

```
def describe_pet(pet_type, pet_name):
    """Виводить інформацію про домашню тварину."""
    print(f"I have a {pet_type}. Its name is {pet_name}.")

# Передача аргументів за позицією (позиційні аргументи)
describe_pet('hamster', 'Harry')  # I have a hamster. Its name is Harry.
# Передача аргументів за іменем (іменовані аргументи)
describe_pet(pet_name='Whiskers', pet_type='cat')  # I have a cat. Its name is Whiskers.
```

3.4. Функції з оператором return

Оператор return завершує виконання функції та повертає значення в місце виклику.

```
def calculate_square(side_length):
    """Повертає площу квадрата."""
    area = side_length ** 2
    return area  # Функція "віддає" результат зовні

my_square_area = calculate_square(5)  # Викликає функцію, результат (25) записується у змінну
print(f"Площа: {my_square_area}")  # Площа: 25
```

Функція може повертати будь-який тип даних (int, float, str, list, bool тощо) або кілька значень (у вигляді кортежу).

3.5. Параметри за замовчуванням

Параметр може отримати значення за умовчанням, яке використовується, якщо аргумент не передали.

```
def describe_pet(pet_name, pet_type='dog'):  # pet_type має значення за замовчуванням
    """Виводить інформацію про домашню тварину."""
    print(f"I have a {pet_type}. Its name is {pet_name}.")
```

```
describe_pet('Rex')  # I have a dog. Its name is Rex.
describe_pet('Molly', 'parrot')  # I have a parrot. Its name is Molly.
```

Важливо: параметри зі значенням за замовчуванням повинні слідувати після параметрів без значень за замовчуванням.

3.6. Локальні та глобальні змінні

Локальна змінна – оголошена всередині функції. Вона існує тільки під час виконання цієї функції і недоступна ззовні.

Глобальна змінна – оголошена поза будь-якою функцією, на рівні модуля. Доступна для читання всередині функцій.

```
global_var = "I am global"

def my_function():
    local_var = "I am local"  # Локальна змінна
    print(global_var)  # Читання глобальної змінної - працює
    print(local_var)  # Працює

my_function()
# print(local_var)  # Помилка! Змінна local_var не визначена в цій області видимості.
```

Для зміни глобальної змінної всередині функції потрібно використати ключове слово global (але це вважається поганою практикою, краще передавати значення через параметри та return).

3.7. Документування функцій (docstrings)

Docstring – це рядок у потрійних лапках одразу після заголовка def. Він служить для документації: опису призначення функції, її параметрів та значення, що повертається. Це обов'язкова частина професійного коду.

```
def add_numbers(a, b):
    """
    Повертає суму двох чисел.
    Параметри:
    a (int або float): Перше доданок.
    b (int або float): Друге доданок.

    Повертає:
    int або float: Сума a та b.
    """
    return a + b
```

До docstring можна потім звернутися через help(add_numbers) або print(add_numbers.__doc__).

3.8. Декомпозиція задачі на функції

Складну задачу слід розбивати на простіші, логічно завершені підзадачі, кожну з яких реалізувати окремою функцією. Наприклад, для програми "Калькулятор площ фігур" можна створити окремі функції: calculate_circle_area(), calculate_triangle_area(), get_user_input(), display_menu().

3.9. Приклад комплексного використання:

```
def power(base, exponent=2):
    """
    Підносить число base до ступеня exponent.

    Параметри:
        base (float): Основа.
        exponent (float, необов'язковий): Показник ступеня. За замовчуванням 2.

    Повертає:
        float: Результат base ** exponent.
    """
    return base ** exponent

# Виклики функції
result1 = power(3, 4)       # 81.0 (позиційні аргументи)
result2 = power(5)          # 25.0 (використано exponent за замовчуванням)
result3 = power(exponent=3, base=2)     # 8.0 (іменовані аргументи)
```

4. Методичні рекомендації

Задача 1. Обчислення суми та добутку двох чисел

Створіть функцію calculate, яка приймає два числа та символ операції (+ або *) і повертає результат відповідної операції. Якщо символ операції невизначений, функція має повертати None.

```
def calculate(num1, num2, operation):
    """
    Виконує арифметичну операцію над двома числами.
```

```
        Параметри:
            num1 (float): Перше число.
            num2 (float): Друге число.
            operation (str): Символ операції '+' або '*'.

        Повертає:
            float або None: Результат операції або None при
невірній операції.
        """
        if operation == '+':
            return num1 + num2
        elif operation == '*':
            return num1 * num2
        else:
            # Якщо операція не '+' і не '*', повертаємо None
            return None

    # Приклади виклику функції
    result1 = calculate(5, 3, '+')
    result2 = calculate(4, 2.5, '*')
    result3 = calculate(10, 7, '-')

    print(f"5 + 3 = {result1}")
    print(f"4 * 2.5 = {result2}")
    print(f"10 - 7 = {result3}")  # Операція не визначена
```

Приклад вхідних та вихідних даних:

```
Вхід: calculate(5, 3, '+') → Вихід: 8
Вхід: calculate(4, 2.5, '*') → Вихід: 10.0
Вхід: calculate(10, 7, '-') → Вихід: None
```

Коментарі:

- Функція демонструє використання параметрів і оператора return з умовною логікою.
- Важливо обробити випадок з некоректним вводом (операцією). Повернення None – один з прийнятних варіантів.

Задача 2. Форматування повного імені користувача

Напишіть функцію format_name, яка приймає ім'я, прізвище та по батькові (останній параметр необов'язковий) і повертає рядок у форматі "Прізвище І.Б.", де Б. – перша літера по батькові (якщо воно передано). Використовуйте параметри за замовчуванням.

```
    def format_name(last_name, first_name, patronymic=None):
        """
        Форматує повне ім’я у вигляді "Прізвище І.Б.".

        Параметри:
            last_name (str): Прізвище.
            first_name (str): Ім’я.
            patronymic (str, необов’язковий): По батькові. За
замовчуванням None.

        Повертає:
            str: Відформатоване ім’я.
        """
        # Форматуємо базову частину: Прізвище та перша літера
імені
        formatted = f"{last_name} {first_name[0]}."

        # Додаємо першу літеру по батькові, якщо воно передано
        if patronymic:
            formatted += f"{patronymic[0]}."

        return formatted

    # Приклади виклику функції
    name1 = format_name("Петренко", "Іван", "Олексійович")
    name2 = format_name("Шевченко", "Марія")  # Без по батькові
    name3 = format_name(last_name="Коваленко",
first_name="Андрій", patronymic="Володимирович")

    print(name1)
    print(name2)
    print(name3)
```

Приклад вхідних та вихідних даних:

Вхід: `format_name("Петренко", "Іван", "Олексійович")` → Вихід: `"Петренко І.О."`

Вхід: `format_name("Шевченко", "Марія")` → Вихід: `"Шевченко М."`

Вхід: `format_name("Коваленко", "Андрій", "Володимирович")` → Вихід: `"Коваленко А.В."`

Коментарі:

- Завдання ілюструє використання параметра за замовчуванням (patronymic=None) та умовну обробку аргументів.

- Зверніть увагу на індексацію рядків для отримання першої літери (first_name[0]).
- Можна використовувати як позиційні, так і іменовані аргументи.

Задача 3. Перевірка простоти числа

Створіть функцію is_prime, яка приймає ціле число більше 1 і повертає True, якщо число просте, та False – якщо ні. Функція має містити докладний docstring.

```
    def is_prime(n):
        """
        Перевіряє, чи є задане число простим.

        Просте число - натуральне число більше 1, яке має рівно два дільники:
        1 та саме себе.

        Параметри:
            n (int): Число для перевірки. Має бути > 1.

        Повертає:
            bool: True, якщо число просте, False - якщо складене.

        Викидає:
            ValueError: Якщо n <= 1.
        """
        # Перевірка коректності вхідних даних
        if n <= 1:
            raise ValueError("Число має бути більше 1")

        # Перевірка дільників від 2 до квадратного кореня з n
        for i in range(2, int(n ** 0.5) + 1):
            if n % i == 0:
                return False  # Знайшли дільник - число складене
        return True  # Дільників не знайшли - число просте

    # Приклади виклику функції з обробкою винятків
    test_numbers = [2, 17, 25, 1]
    for num in test_numbers:
        try:
            print(f"{num} - просте? {is_prime(num)}")
        except ValueError as e:
            print(f"Помилка для {num}: {e}")
```

Приклад вхідних та вихідних даних:

```
Вхід: is_prime(2) → Вихід: True
Вхід: is_prime(17) → Вихід: True
Вхід: is_prime(25) → Вихід: False
Вхід: is_prime(1) → Вихід: ValueError: Число має бути більше
1
```

Коментарі:

- Функція демонструє повноцінне документування (docstring), що включає опис, параметри, значення, що повертається, та винятки.
- Оптимізація алгоритму: перевірка дільників йде тільки до квадратного кореня з числа.
- Важливо обробляти некоректні вхідні дані (у цьому прикладі – через raise).

Задача 4. Аналіз списку чисел

Реалізуйте функцію analyze_numbers, яка приймає список чисел і повертає словник з основними статистичними характеристиками: мінімум, максимум, суму та середнє арифметичне. Використовуйте вбудовані функції Python.

```
def analyze_numbers(numbers):
    """
    Обчислює основні статистичні характеристики списку чисел.

    Параметри:
        numbers (list of int/float): Список чисел для аналізу.

    Повертає:
        dict: Словник з ключами:
            - 'min': мінімальне значення
            - 'max': максимальне значення
            - 'sum': сума всіх чисел
            - 'average': середнє арифметичне
    """
    if not numbers:  # Обробка порожнього списку
        return {'min': None, 'max': None, 'sum': 0, 'average':
None}

    # Обчислення характеристик за допомогою вбудованих функцій
```

```
        stats = {
            'min': min(numbers),
            'max': max(numbers),
            'sum': sum(numbers),
            'average': sum(numbers) / len(numbers)
        }
        return stats

    # Приклади виклику функції
    list1 = [5, 2, 8, 1, 9, 3]
    list2 = [-10, 0, 10, 20]
    list3 = []  # Порожній список

    result1 = analyze_numbers(list1)
    result2 = analyze_numbers(list2)
    result3 = analyze_numbers(list3)

    print("Аналіз list1:", result1)
    print("Аналіз list2:", result2)
    print("Аналіз list3:", result3)
```

Приклад вхідних та вихідних даних:

Вхід: [5, 2, 8, 1, 9, 3] → Вихід: {'min': 1, 'max': 9, 'sum': 28, 'average': 4.666...}

Вхід: [-10, 0, 10, 20] → Вихід: {'min': -10, 'max': 20, 'sum': 20, 'average': 5.0}

Вхід: [] → Вихід: {'min': None, 'max': None, 'sum': 0, 'average': None}

Коментарі:

- Функція повертає складний тип даних – словник, що ілюструє гнучкість return.
- Важливо обробити крайовий випадок з порожнім списком, щоб уникнути помилок ділення на нуль.
- Використання вбудованих функцій min(), max(), sum() робить код ефективним і чистим.

Типові помилки і шляхи їх усунення

1. **Зміна глобальної змінної без ключового слова global:**
   Помилка. UnboundLocalError: local variable 'x' referenced before assignment.

Причина. Всередині функції відбувається присвоєння змінній з іменем, що існує у глобальній області видимості, інтерпретатор вважає її новою локальною.

Рішення. Якщо дійсно потрібно змінити глобальну змінну, використовуйте global x на початку функції. Але кращий підхід – передавати значення через параметри і повертати результат.

2. **Неочікувана зміна мутабельного аргументу (наприклад, списку):**

Помилка. Зміни всередині функції зачіпають оригінальний список, переданий ззовні.

Причина. У функцію передається посилання на той самий об'єкт списку.

Рішення. Якщо функція не повинна змінювати вхідні дані, створюйте копію: def func(data): working_list = data.copy(). Або явно документуйте, що функція змінює вхідний аргумент.

3. **Використання параметра зі значенням за замовчуванням-мутабелом (списком, словником):**

Помилка. def add_item(item, my_list=[]): – усі виклики без другого аргументу використовують один і той самий список.

Причина. Значення за замовчуванням обчислюється один раз при оголошенні функції.

Рішення. Використовуйте None: def add_item(item, my_list=None): my_list = my_list or [].

4. **Відсутність оператора return або повернення не в тому місці:**

Помилка. Функція повертає None замість очікуваного результату.

Причина. Забули return, або return виконується занадто рано (наприклад, всередині циклу, коли потрібно після нього).

Рішення. Уважно перевіряйте логіку функції та позицію оператора return. Використовуйте print() для налагодження.

Корисні поради

1. **Думайте про функцію як про "чорну скриньку".** Вона повинна отримувати чітко визначені вхідні дані (параметри) і повертати результат. Мінімізуйте побічні ефекти (наприклад, вивід на екран, зміну глобальних змінних) – це робить код передбачуваним і тестованим.

2. **Давайте функціям описові імена.** Ім'я функції має відображати її дію (calculate_average, is_valid_email, load_config). Використовуйте дієслова. Це покращує читабельність.

3. **Функція має виконувати одну дію.** Принцип єдиної відповідальності (Single Responsibility Principle). Якщо функція робить забагато (наприклад, читає файл, обробляє дані і зберігає результат), розбийте її на декілька.

4. **Обов'язково пишіть docstrings.** Це інвестування в майбутнє – для вас та інших розробників. Він пояснює *що* робить функція, а не *як* (як це роблять коментарі в тілі).

5. **Починайте з інтерфейсу.** Подумайте спочатку, як функцію буде зручно викликати (які параметри, які значення за замовчуванням), а потім вже реалізовуйте її внутрішню логіку.

6. **Не бійтеся створювати багато невеликих функцій.** Це значно краще, ніж одна велика. Код стає модульним, його легше тестувати, налагоджувати та повторно використовувати.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Вивід привітання

Створіть функцію say_hello без параметрів, яка виводить у консоль українською мовою: "Вітаю! Ласкаво просимо до програми обчислень."

```
Функція не приймає вхідних даних
Вивід: "Вітаю! Ласкаво просимо до програми обчислень."
```

Завдання 1.2. Малювання рамки

Створіть функцію draw_frame без параметрів, яка виводить у консоль рамку з зірочок розміром 5x20 (5 рядків висоти, 20 символів ширини).

```
Функція не приймає вхідних даних
Вивід:
********************
*                  *
*                  *
*                  *
********************
```

Завдання 1.3. Поточна дата та час

Створіть функцію show_datetime без параметрів, яка виводить поточну дату та час у форматі: "Сьогодні: DD.MM.YYYY, час: HH:MM"

```
Функція не приймає вхідних даних
Вивід: "Сьогодні: 15.05.2024, час: 14:30" (актуальна дата та час)
```

Завдання 1.4. Таблиця множення на 7

Створіть функцію multiplication_table_7 без параметрів, яка виводить таблицю множення числа 7 від 1 до 10 у форматі "7 × 1 = 7".

```
Функція не приймає вхідних даних
Вивід:
7 × 1 = 7
7 × 2 = 14
...
7 × 10 = 70
```

Завдання 1.5. Калькулятор віку

Створіть функцію calculate_age з одним параметром birth_year, яка виводить вік людини, якщо поточний рік - 2024.

```
Вхід: calculate_age(2000) → Вихід: "Ваш вік: 24 роки"
Вхід: calculate_age(2010) → Вихід: "Ваш вік: 14 років"
Вхід: calculate_age(1995) → Вихід: "Ваш вік: 29 років"
```

Завдання 1.6. Конвертер температури

Створіть функцію celsius_to_fahrenheit з одним параметром celsius, яка конвертує температуру з градусів Цельсія у Фаренгейти за формулою: F = C × 9/5 + 32.

```
Вхід: celsius_to_fahrenheit(0) → Вихід: "0°C = 32.0°F"
Вхід: celsius_to_fahrenheit(25) → Вихід: "25°C = 77.0°F"
Вхід: celsius_to_fahrenheit(-10) → Вихід: "-10°C = 14.0°F"
```

Завдання 1.7. Генератор паролю

Створіть функцію generate_password з одним параметром length, яка виводить випадковий пароль заданої довжини, що складається тільки з літер латинського алфавіту (нижній регістр).

```
Вхід: generate_password(5) → Вихід: "kdmwr" (випадкові літери)
Вхід: generate_password(8) → Вихід: "axfhjklp"
Вхід: generate_password(12) → Вихід: "qwetryuiopas"
```

Завдання 1.8. Обчислення площі трикутника

Створіть функцію triangle_area з параметрами base та height, яка **повертає** площу трикутника за формулою: S = ½ × основа × висота.

```
Вхід: triangle_area(10, 5) → Вихід: 25.0
Вхід: triangle_area(7, 3) → Вихід: 10.5
Вхід: triangle_area(15.5, 8.2) → Вихід: 63.55
```

Завдання 1.9. Перевірка парності числа

Створіть функцію is_even з одним параметром number, яка **повертає** True, якщо число парне, та False, якщо непарне.

```
Вхід: is_even(4) → Вихід: True
Вхід: is_even(7) → Вихід: False
Вхід: is_even(0) → Вихід: True
```

Завдання 1.10. Форматування ціни

Створіть функцію format_price з одним параметром price, яка **повертає** рядок у форматі "Ціна: XXX грн.", де XXX - ціна, округлена до двох знаків після коми.

```
Вхід: format_price(45.5) → Вихід: "Ціна: 45.50 грн."
Вхід: format_price(123.456) → Вихід: "Ціна: 123.46 грн."
Вхід: format_price(78) → Вихід: "Ціна: 78.00 грн."
```

Частина 2. Середні завдання

Завдання 2.1. Калькулятор знижок

Створіть функцію calculate_discount з параметрами price, discount_percent та is_member (останній параметр за замовчуванням False). Якщо is_member=True, то до знижки додатково застосовується 5%. Функція повинна повертати кінцеву ціну.

```
Вхід: calculate_discount(1000, 10) → Вихід: 900.0
Вхід: calculate_discount(500, 20, True) → Вихід: 380.0 (500 -
20% = 400, -5% = 380)
Вхід: calculate_discount(750, 15, False) → Вихід: 637.5
```

Завдання 2.2. Форматування повної адреси

Створіть функцію format_address з параметрами: city, street, house, apartment=None. Функція повинна повертати адресу у форматі "м. City, вул. Street, буд. House, кв. Apartment". Якщо квартира не вказана, не включати "кв. Apartment".

```
Вхід: format_address("Київ", "Хрещатик", "1") → Вихід: "м. Київ, вул. Хрещатик, буд. 1"
Вхід: format_address("Львів", "Свободи", "15", "42") → Вихід: "м. Львів, вул. Свободи, буд. 15, кв. 42"
Вхід: format_address("Одеса", "Дерибасівська", "22", apartment="5") → Вихід: "м. Одеса, вул. Дерибасівська, буд. 22, кв. 5"
```

Завдання 2.3. Статистика тексту

Створіть функцію text_statistics з одним параметром text, яка повертає словник з наступною статистикою: кількість символів, кількість слів, кількість речень (речення закінчуються на ".", "!" або "?").

```
Вхід: text_statistics("Привіт! Як справи?") → Вихід: {'символів': 17, 'слів': 3, 'речень': 2}
Вхід: text_statistics("Python - чудова мова. Вона проста та потужна.") → Вихід: {'символів': 46, 'слів': 7, 'речень': 2}
Вхід: text_statistics("") → Вихід: {'символів': 0, 'слів': 0, 'речень': 0}
```

Завдання 2.4. Калькулятор вартості доставки

Створіть **дві функції** для розрахунку вартості доставки:

1. calculate_weight_cost(weight) - розраховує вартість за вагою (до 5 кг - 50 грн, 5-10 кг - 80 грн, більше 10 кг - 80 + 10 за кожен кг понад 10)

2. calculate_delivery(distance, weight, is_express=False) - використовує першу функцію та додає вартість за відстань (1 грн/км, якщо експрес – 2 грн/км)

```
Вхід: calculate_delivery(50, 4) → Вихід: 100 (50км×1 + 50 = 100)
Вхід: calculate_delivery(100, 14) → Вихід: 220 (100×1 + 80 + (14-10)×10 = 100 + 120 = 220)
Вхід: calculate_delivery(30, 8, True) → Вихід: 140 (30×2 + 80 = 60 + 80 = 140)
```

Завдання 2.5. Система обліку завдань

Створіть **дві функції** для роботи зі списком завдань:

1. add_task(task_list, task_name, priority="medium") - додає завдання у форматі словника {"name": task_name, "priority": priority, "completed": False}

2. complete_task(task_list, task_name) - позначає завдання як виконане

```
    tasks = []
    add_task(tasks, "Вивчити Python", "high")
    add_task(tasks, "Купити молоко")
    complete_task(tasks, "Купити молоко")
    # tasks = [{'name': 'Вивчити Python', 'priority': 'high', 'completed': False},
    #          {'name': 'Купити молоко', 'priority': 'medium', 'completed': True}]
```

Частина 3. Складні завдання

Завдання 3.1. Система управління бібліотекою

Створіть програму з **п'ятьма функціями** для управління бібліотекою книг:

1. add_book(library, title, author, year, genre) - додає книгу

2. find_books_by_author(library, author) - повертає список книг автора

3. find_books_by_genre(library, genre) - повертає список книг жанру

4. get_books_published_after(library, year) - повертає книги після вказаного року

5. get_library_statistics(library) - повертає статистику: загальна кількість, кількість за авторами, за жанрами

```
    library = []
    add_book(library, "Мастер и Маргарита", "Булгаков", 1966, "роман")
    add_book(library, "Собачье сердце", "Булгаков", 1925, "повесть")
    add_book(library, "1984", "Оруэлл", 1949, "антиутопия")
    # Статистика покаже: всього 3 книги, 2 автора, 3 жанру
```

Завдання 3.2. Аналізатор фінансових витрат

Створіть програму з **чотирма функціями** для аналізу фінансових операцій:

1. add_expense(expenses, date, category, amount, description="") - додає витрату

2. get_expenses_by_category(expenses, category) - витрати за категорією

3. get_monthly_summary(expenses, year, month) - підсумок за місяць

4. get_category_statistics(expenses) - відсотковий розподіл за категоріями

Кожна витрата - словник з полями: дата (рядок "DD.MM.YYYY"), категорія, сума, опис.

```
expenses = []
add_expense(expenses, "15.05.2024", "Їжа", 350, "Продукти")
add_expense(expenses, "16.05.2024", "Транспорт", 50, "Таксі")
add_expense(expenses, "17.05.2024", "Їжа", 200, "Кафе")
# Статистика за категоріями: Їжа - 73.3%, Транспорт - 26.7%
```

Завдання 3.3. Генератор звітів про студентів

Створіть програму з **шістьма функціями** для роботи зі студентами:

1. add_student(students, name, group) - додає студента з порожнім журналом оцінок

2. add_grade(students, name, subject, grade) - додає оцінку студенту

3. get_student_gpa(students, name) - розраховує середній бал студента

4. get_group_gpa(students, group) - середній бал групи

5. get_subject_statistics(students, subject) - статистика з предмету

6. generate_report(students) - генерує детальний звіт у вигляді відформатованого рядка

```
students = []
add_student(students, "Іваненко", "КН-101")
add_grade(students, "Іваненко", "Математика", 90)
add_grade(students, "Іваненко", "Програмування", 85)
# Звіт містить: список студентів, їх оцінки, середні бали, найкращі студенти
```

6. Питання для самоперевірки

1. Що таке функція в програмуванні та які основні переваги їх використання?

2. Поясніть різницю між параметром та аргументом функції.

3. Яке призначення ключового слова return? Що повертає функція, якщо в ній відсутній оператор return?

4. Яка різниця між позиційними та іменованими аргументами? Наведіть приклад.

5. Що таке параметр за замовчуванням? Чому не рекомендується використовувати мутабельні типи даних (списки, словники) як значення за замовчуванням?

6. Що таке область видимості змінної? Поясніть різницю між локальною та глобальною змінною.

7. Для чого призначений docstring? Як його можна переглянути під час виконання програми?

8. Що таке декомпозиція задачі? Які переваги вона дає при створенні великих програм?

9. Чи може функція повертати кілька значень? Як це реалізується в Python?

10. Що станеться, якщо ви передасте функції більше або менше аргументів, ніж вона очікує (кількість параметрів)?

11. Що таке None в Python? У яких ситуаціях функція може повертати це значення?

12. Чи може функція викликати саму себе? Як називається такий принцип і які в нього небезпеки?

13. Поясніть, що таке побічний ефект (side effect) функції. Чи завжди це погано?

14. Чим відрізняється виклик функції зі значенням, що повертається, від виклику функції, яка просто виконує дію (наприклад, вивід на екран)?

15. Як правильно називати функції та їх параметри згідно з конвенціями PEP 8?

16. Що таке "чиста" функція (pure function)? Наведіть її ознаки.

17. Як аргументи передаються в функцію в Python: за значенням або за посиланням? Поясніть на прикладі зі зміною числа та списку всередині функції.

18. Чи можна оголошувати функцію всередині іншої функції? Яка в такого підходу особливість?

19. Що таке pass і коли його використовують при оголошенні функції?

20. Як би ви пояснили принцип роботи функції за допомогою аналогії з реального життя (наприклад, кухонний прилад, конвеєр тощо)?

ЛАБОРАТОРНА РОБОТА №10
Розширені можливості функцій: lambda, рекурсія, функції вищого порядку

1. Мета

Сформувати у студентів практичні навички створення та застосування розширених механізмів роботи з функціями в Python. Робота спрямована на засвоєння концепцій функцій як об'єктів першого класу, зокрема створення анонімних lambda-функцій, використання функцій вищого порядку (map(), filter(), reduce()) для функціонального стилю програмування, роботи з довільною кількістю аргументів (*args та **kwargs), а також на розуміння та реалізацію рекурсивних алгоритмів. Здобуті вміння є основою для написання гнучкого, модульного та ефективного програмного коду.

2. Завдання

1. Оволодіти синтаксисом створення та використання lambda-функцій.
2. Навчитися застосовувати вбудовані функції вищого порядку (map(), filter(), reduce()) для обробки послідовностей.
3. Зрозуміти принципи роботи з довільною кількістю позиційних та іменованих аргументів у функціях.
4. Освоїти базовий підхід до реалізації рекурсії: визначення базового випадку та рекурсивного виклику.
5. Розглянути приклади функцій, що приймають інші функції як аргументи або повертають функції як результат.
6. Виконати практичні завдання різного рівня складності для закріплення теоретичного матеріалу.

3. Короткі теоретичні відомості

У Python функції є об'єктами **першого класу** (first-class citizens). Це означає, що з ними можна працювати так само, як і з будь-якими іншими об'єктами (числами, рядками, списками): їх можна присвоювати змінним, передавати як аргументи в інші функції, повертати як значення з функцій. Ця властивість відкриває шлях до потужних парадигм програмування, таких як функціональне програмування.

3.1. Lambda-функції

Lambda-функції – це невеликі анонімні (без імені) функції, створені за допомогою ключового слова lambda. Вони можуть приймати будь-яку кількість аргументів, але повертати лише одне значення (результат обчислення виразу). Їхній синтаксис обмежений, тому вони ідеальні для простих операцій.

Синтаксис: lambda arguments: expression

```
# Звичайна функція
def square(x):
    return x ** 2
# Еквівалентна lambda-функція
square_lambda = lambda x: x ** 2

print(square(5))          # 25
print(square_lambda(5))   # 25
```

Lambda часто використовуються разом з функціями вищого порядку, коли потрібна одноразова проста операція.

```
Сортування списку рядків за останньою літерою
names = ['Anna', 'Max', 'Olga', 'Ivan']
sorted_names = sorted(names, key=lambda name: name[-1])
print(sorted_names)  # ['Anna', 'Olga', 'Max', 'Ivan']
```

3.2. Функції вищого порядку: map(), filter(), reduce()

Це функції, які приймають іншу функцію як аргумент або повертають функцію. До найвживаніших належать map(), filter() та reduce() (з модуля functools).

- map(function, iterable, ...) – застосовує *function* до кожного елементу *iterable* та повертає ітератор з результатами.

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # [1, 4, 9, 16]
```

- filter(function, iterable) – фільтрує елементи *iterable*, залишаючи тільки ті, для яких *function* повертає True.

```
numbers = [1, 2, 3, 4, 5, 6]
even = list(filter(lambda x: x % 2 == 0, numbers))
print(even)  # [2, 4, 6]
```

- reduce(function, iterable[, initializer]) (з functools) – послідовно застосовує *function* до елементів *iterable*, згортаючи його до єдиного значення. Функція має приймати два аргументи.

```
from functools import reduce
numbers = [1, 2, 3, 4]
product = reduce(lambda a, b: a * b, numbers)  # (((1*2)*3)*4)
print(product)  # 24
```

3.3. Робота з довільною кількістю аргументів: *args та kwargs

Python дозволяє створювати функції, що приймають довільну кількість аргументів.

- *args (arguments) – збирає усі позиційні аргументи в кортеж.
- **kwargs (keyword arguments) – збирає усі іменовані аргументи в словник.

```
def student_info(name, group, *grades, **additional_info):
    print(f"Student: {name}, Group: {group}")
    print(f"Grades: {grades}")
    for key, value in additional_info.items():
        print(f"{key}: {value}")
# Виклик
student_info('Maria', 'KN-101', 85, 92, 78, scholarship=True, dormitory=14)
# Student: Maria, Group: KN-101
# Grades: (85, 92, 78)
# scholarship: True
# dormitory: 14
```

3.4. Рекурсивні функції

Рекурсія – це техніка, коли функція викликає саму себе. Кожна рекурсивна функція повинна мати:

1. **Базовий випадок (умова виходу)** – простий випадок, який можна вирішити без рекурсії, щоб уникнути нескінченного виклику.
2. **Рекурсивний випадок** – виклик тієї самої функції з іншими (зазвичай простішими) аргументами.

Класичні приклади:

```
# 1. Факторіал числа n: n! = 1 * 2 * ... * n, 0! = 1
def factorial(n):
    if n == 0:          # Базовий випадок
        return 1
    else:               # Рекурсивний випадок
        return n * factorial(n - 1)

# 2. Числа Фібоначчі: F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2)
def fibonacci(n):
    if n <= 1:          # Базовий випадок (F(0) або F(1))
```

```
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

print(factorial(5))    # 120
print(fibonacci(7))    # 13
```

3.5. Функції як аргументи та функції, що повертають функції

Оскільки функції – це об'єкти, їх можна використовувати динамічно.

```
 Функція як аргумент
def apply_operation(func, a, b):
    return func(a, b)
def add(x, y):
    return x + y
def multiply(x, y):
    return x * y
result1 = apply_operation(add, 5, 3)        # 8
result2 = apply_operation(multiply, 5, 3)   # 15

# Функція, що повертає функцію (фабрика функцій)
def create_greeter(greeting):
    def greeter(name):
        return f"{greeting}, {name}!"
    return greeter

say_hello = create_greeter("Hello")
say_hi = create_greeter("Hi")

print(say_hello("Oleh"))  # Hello, Oleh!
print(say_hi("Anna"))     # Hi, Anna!
```

Важливо: рекурсія вимагає уваги до глибини викликів (в Python є обмеження) та може бути неефективною для деяких задач (наприклад, наївна реалізація Фібоначчі). Її слід застосовувати там, де вона природно відображає структуру задачі (наприклад, обхід дерев).

4. Методичні рекомендації

Розглянемо розв'язання типових задач, що охоплюють основні теми лабораторної роботи.

Задача 1. Фільтрація списку за умовою

Створіть програму, яка з заданого списку чисел фільтрує лише ті, які більші за середнє арифметичне всього списку. Використайте функції filter() та lambda. Оформити у вигляді функції filter_above_average().

```
from typing import List

def filter_above_average(numbers: List[float]) -> List[float]:
    """
    Фільтрує числа, які більші за середнє арифметичне списку.
    Args:
        numbers: список чисел
    Returns:
        Список чисел, більших за середнє значення вхідного
списку
    """
    # Обчислюємо середнє арифметичне. Уникаємо ділення на
нуль.
    if not numbers:
        return []
    average = sum(numbers) / len(numbers)

    # Використовуємо filter з lambda-умовою
    filtered = list(filter(lambda x: x > average, numbers))
    return filtered

# Приклади використання та тестування
if __name__ == "__main__":
    # Тест 1: Основний випадок
    test1 = [1, 2, 3, 4, 5]
    print(f"Вхід: {test1} → Вихід: {filter_above_average(test1)}")
    # Середнє = 3.0, числа > 3: [4, 5]

    # Тест 2: Всі числа однакові
    test2 = [7, 7, 7, 7]
    print(f"Вхід: {test2} → Вихід: {filter_above_average(test2)}")
    # Середнє = 7.0, чисел > 7 немає: []

    # Тест 3: Дробові числа
    test3 = [1.5, 2.5, 3.5, 4.5]
    print(f"Вхід: {test3} → Вихід: {filter_above_average(test3)}")
    # Середнє = 3.0, числа > 3.0: [3.5, 4.5]

    # Тест 4: Порожній список
    test4 = []
```

```
print(f"Вхід: {test4} → Вихід: {filter_above_average(test4)}")
# []
```

Коментарі:

- Функція анотована типами (List[float], -> List[float]) для кращої читабельності.
- Важливо обробляти крайовий випадок з порожнім списком, щоб уникнути помилки ділення на нуль.
- Функція filter() повертає ітератор, тому для отримання списку необхідно використати list().
- Умова фільтрації x > average чітко відображає логіку задачі.

Задача 2. Обробка довільної кількості аргументів

Напишіть функцію merge_info(), яка приймає довільну кількість позиційних аргументів-рядків (імена) та довільну кількість іменованих аргументів (додаткова інформація у вигляді "характеристика=значення"). Функція повинна повертати список рядків, де кожен рядок – це ім’я разом з усією додатковою інформацією у форматі "Ім’я: характеристика1=значення1, характеристика2=значення2, ...".

```
def merge_info(*names: str, **details) -> list:
    """
    Об’єднує імена з додатковою інформацією.

    Args:
        *names: довільна кількість позиційних аргументів-рядків (імена)
        **details: довільна кількість іменованих аргументів

    Returns:
        Список відформатованих рядків
    """
    result = []
    # Проходимо по кожному імені
    for name in names:
        # Якщо є додаткова інформація, форматуємо її
        if details:
            details_str = ", ".join([f"{key}={value}" for key, value in details.items()])
            formatted_string = f"{name}: {details_str}"
        else:
```

```
                formatted_string = f"{name}: немає додаткової
інформації"
            result.append(formatted_string)
        return result

    # Приклади використання
    if __name__ == "__main__":
        # Тест 1: Звичайний випадок
        output1 = merge_info("Олексій", "Марія", group="KN-101",
year=1)
        print("Вхід: ('Олексій', 'Марія', group='KN-101',
year=1)")
        for line in output1:
            print(f"  Вихід: {line}")

        # Тест 2: Лише імена без додаткової інформації
        output2 = merge_info("Іван")
        print("\nВхід: ('Іван')")
        for line in output2:
            print(f"  Вихід: {line}")

        # Тест 3: Багато додаткової інформації
        output3 = merge_info("Анна", "Богдан", "Світлана",
                             scholarship=True, dormitory=5,
average_grade=85.5)
        print("\nВхід: ('Анна', 'Богдан', 'Світлана',
scholarship=True, dormitory=5, average_grade=85.5)")
        for line in output3:
            print(f"  Вихід: {line}")
```

Коментарі:

- Аргументи *names збираються в кортеж, **details – у словник.
- Використано list comprehension для ефективного створення рядка з деталями.
- Функція коректно працює, навіть якщо іменованих аргументів не передано.
- Генерація списку рядків дозволяє легко виводити або подальше обробляти результат.

Задача 3. Рекурсивне обчислення суми цифр числа

Реалізуйте функцію digital_root(n), яка обчислює *цифровий корінь* натурального числа. Цифровий корінь – це рекурсивна сума всіх цифр

числа до тих пір, поки не вийде однозначне число. Наприклад, digital_root(942) → 9+4+2=15 → 1+5=6. Не використовуйте цикли.

```
    def digital_root(n: int) -> int:
        """
        Обчислює цифровий корінь числа рекурсивним методом.

        Args:
            n: натуральне число

        Returns:
            Цифровий корінь (однозначне число від 1 до 9, або 0 для
n=0)
        """
        # Базовий випадок: якщо число однозначне (включаючи 0)
        if n < 10:
            return n

        # Рекурсивний випадок: сума цифр поточного числа
        # n % 10 - остання цифра, n // 10 - число без останньої цифри
        sum_of_digits = (n % 10) + digital_root(n // 10)
        # Застосовуємо функцію рекурсивно до отриманої суми
        return digital_root(sum_of_digits)

    # Приклади використання
    if __name__ == "__main__":
        # Тест 1
        print(f"Вхід: 942 → Вихід: {digital_root(942)}")  # 6
        # Тест 2
        print(f"Вхід: 493193 → Вихід: {digital_root(493193)}")  #
2
        # Тест 3 (крайовий випадок)
        print(f"Вхід: 0 → Вихід: {digital_root(0)}")  # 0
        # Тест 4 (однозначне число)
        print(f"Вхід: 7 → Вихід: {digital_root(7)}")  # 7
```

Коментарі:

- Базовий випадок – число менше 10. Для n=0 функція також коректно повертає 0.
- У рекурсивному випадку ми спочатку обчислюємо суму цифр, а потім рекурсивно застосовуємо ту саму функцію до цієї суми.
- Це приклад *непрямої* або *подвійної* рекурсії: функція digital_root викликає себе двічі.

• Важливо переконатися, що рекурсія збігається (аргумент стає меншим з кожним кроком). Тут n // 10 гарантовано менше за n для n >= 10.

Задача 4. Функція вищого порядку для застосування операції

Створіть функцію create_processor(), яка приймає на вхід одну функцію-перетворювач (transformer) та повертає *нову функцію*. Ця нова функція має приймати список елементів і застосовувати transformer до кожного елемента, але лише якщо елемент задовольняє умові фільтрації (перевірка на входження до множини allowed). Фільтр також передається в create_processor при її створенні.

```
from typing import Callable, List, Any, Set

def create_processor(transformer: Callable[[Any], Any],
                     allowed:          Set[Any])          ->
Callable[[List[Any]], List[Any]]:
    """
    Фабрика функцій для обробки списків.
    Створює функцію, яка фільтрує список за множиною allowed
    і застосовує transformer до кожного елемента, що пройшов
фільтр.

    Args:
        transformer: функція для перетворення елемента
        allowed: множина дозволених значень для фільтрації

    Returns:
        Функція, яка приймає список і повертає новий список.
    """
    def process_list(items: List[Any]) -> List[Any]:
        # Фільтруємо (перевіряємо на входження в allowed) і
застосовуємо перетворення
        result = [transformer(item) for item in items if item
in allowed]
        return result

    # Повертаємо внутрішню функцію як результат
    return process_list

# Приклади використання
if __name__ == "__main__":
```

```
        # 1. Створюємо процесор для роботи з числами
        # Трансформер: помножити на 2. Фільтр: лише парні числа з множини {2,4,6,8,10}
        double_allowed_evens = create_processor(lambda x: x * 2, {2, 4, 6, 8, 10})

        test_list1 = [1, 2, 3, 4, 5, 6, 10, 12]
        output1 = double_allowed_evens(test_list1)
        print(f"Вхід: {test_list1} → Вихід: {output1}")
        # Пояснення: з списку беруться лише 2,4,6,10 (вони в множині allowed). Кожне з них множиться на 2.
        # 12 не в множині allowed. Результат: [4, 8, 12, 20]

        # 2. Створюємо процесор для роботи з рядками
        # Трансформер: перевести у верхній регістр. Фільтр: лише конкретні міста
        capitalize_cities = create_processor(str.upper, {"kyiv", "lviv", "odesa"})

        test_list2 = ["kyiv", "kharkiv", "lviv", "odesa", "dnipro"]
        output2 = capitalize_cities(test_list2)
        print(f"Вхід: {test_list2} → Вихід: {output2}")
        # Результат: ['KYIV', 'LVIV', 'ODESA']
```

Коментарі:

- create_processor – це *фабрика функцій*: вона створює та повертає нову функцію (process_list).
- Внутрішня функція process_list "запам'ятовує" (closure) аргументи, передані зовнішній функції (transformer і allowed).
- Використано list comprehension, що поєднує фільтрацію (if item in allowed) та трансформацію (transformer(item)) в один елегантний вираз.
- Типізація Callable допомагає зрозуміти, які функції очікуються на вході та виході.

Типові помилки і шляхи їх усунення

1. **Некоректна умова виходу в рекурсії (відсутній базовий випадок):**

   Помилка. Програма викликає RecursionError: maximum recursion depth exceeded.

   Рішення. Завжди першим питанням при написанні рекурсивної функції має бути: "Який найпростіший випадок

(базовий), який можна вирішити без рекурсії?". Чітко прописати умову if для цього випадку і переконатися, що рекурсивний виклик на кожному кроці наближає аргументи до базового випадку.

2. **Зміна глобальних змінних всередині lambda або функцій вищого порядку:**

Помилка. Неочевидні побічні ефекти, коли функція раптом змінює стан програми.

Рішення. Lambda та функції, що передаються в map/filter/reduce, мають бути *чистими* (pure). Вони не повинні змінювати зовнішні змінні або списки, а лише повертати результат обчислення на основі своїх аргументів. Використовуйте return, а не зміну глобального стану.

3. **Нерозуміння, що map() та filter() повертають ітератор, а не список:**

Помилка. Спроба звернутися за індексом до результату map() або повторно використати його після однієї ітерації.

Рішення. Якщо потрібен саме список, завжди явно перетворюйте результат: list(map(...)). Для одноразового перебору перетворення не обов'язкове.

4. **Плутанина між *args (кортеж) та **kwargs (словник):**

Помилка. Спроба звернутися до args['key'] або передати kwargs без **.

Рішення. Пам'ятайте: args – це **кортеж** позиційних аргументів, доступ за індексом. kwargs – це **словник** іменованих аргументів, доступ за ключем. При передачі kwargs в іншу функцію використовуйте розпакування: other_function(**kwargs).

5. **Створення зайвої lambda для простих операцій:**

Помилка. map(lambda x: str(x), numbers) замість map(str, numbers).

Рішення. Багато вбудованих функцій (str, int, len, abs) вже є об'єктами, які можна передавати напряму. Використовуйте їх, код буде чистішим.

Корисні поради

1. **Почніть з базового випадку.** При розробці рекурсивної функції спочатку напишіть умову для найпростішого вхідного значення, а потім думайте, як звести складну задачу до простої.

2. **Декомпозиція.** Якщо задача з функціями вищого порядку здається складною, розбийте її на частини. Спочатку напишіть окремо функцію-перетворювач, окремо протестуйте фільтрацію, а потім об'єднайте їх у map/filter або list comprehension.

3. **Використовуйте type hints.** Анотації типів (def func(a: int) -> str) не впливають на виконання, але значно покращують читабельність коду та допомагають IDE виявляти потенційні помилки.

4. **Дотримуйтесь PEP 8.** Це робить ваш код професійним і зрозумілим для інших. Звертайте увагу на відступи (4 пробіли), пробіли навколо операторів, іменування функцій (snake_case).

5. **Тестуйте на крайових випадках.** Завжди перевіряйте свою функцію на порожньому списку, нулі, від'ємних числах, одному елементі. Це допомагає виявити логічні помилки та зробити код надійним.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Lambda: Чи є число парним

Напишіть lambda-функцію is_even, яка приймає одне ціле число і повертає True, якщо воно парне, і False – якщо ні. Не використовуйте умовні оператори (if/else) в тілі lambda.

```
Вхід: 10 → Вихід: True
Вхід: 7 → Вихід: False
Вхід: 0 → Вихід: True
```

Завдання 1.2. Lambda: Конкатенація з форматуванням

Створіть lambda-функцію greet, яка приймає два аргументи: ім'я (name) та вітання (greeting) за замовчуванням "Hello". Функція повинна повертати рядок у форматі "<greeting>, <name>!".

```
Вхід: ("Anna", "Hi") → Вихід: "Hi, Anna!"
Вхід: ("Petro") → Вихід: "Hello, Petro!"
Вхід: ("Maria", "Good morning") → Вихід: "Good morning, Maria!"
```

Завдання 1.3. Lambda: Обчислення площі прямокутника

Створіть lambda-функцію area, яка приймає довжину та ширину прямокутника (цілі або дробові числа) і повертає його площу. Передбачте, що аргументи можуть бути передані в довільному порядку (спочатку ширина, потім довжина). Використайте вбудовані функції для цього.

```
Вхід: (5, 3) → Вихід: 15.0
Вхід: (2.5, 4) → Вихід: 10.0
Вхід: (3) → Вихід: Помилка (підказка: функція має приймати два аргументи)
```

Завдання 1.4. Lambda: Перевірка на паліндром

Напишіть lambda-функцію is_palindrome, яка приймає рядок, перетворює його на нижній регістр, видаляє всі пробіли і перевіряє, чи дорівнює рядок сам собі у зворотному порядку. Повертає булеве значення.

```
Вхід: "Anna" → Вихід: True
Вхід: "Python" → Вихід: False
Вхід: "A man a plan a canal Panama" → Вихід: True
```

Завдання 1.5. map(): Застосування знижки

Дано список цін товарів prices = [100, 200, 300, 400]. За допомогою map() та lambda створіть новий список discounted_prices, де до кожної ціни застосовано знижку 15%. Результат округліть до двох знаків після коми.

```
Вхід: [100, 200, 300, 400] → Вихід: [85.0, 170.0, 255.0, 340.0]
Вхід: [50, 1500] → Вихід: [42.5, 1275.0]
Вхід: [] → Вихід: []
```

Завдання 1.6. filter(): Знайти слова певної довжини

Дано список слів words = ["cat", "elephant", "dog", "house", "algorithm"]. Використовуючи filter() та lambda, створіть новий список, що містить тільки ті слова, довжина яких більша за 4 символи.

```
Вхід: ["cat", "elephant", "dog", "house", "algorithm"] → Вихід: ["elephant", "house", "algorithm"]
Вхід: ["I", "am"] → Вихід: []
Вхід: ["computer", "science"] → Вихід: ["computer", "science"]
```

Завдання 1.7. map() та filter(): Обробка оцінок

Дано список оцінок студентів (цілі числа від 1 до 100) grades = [45, 85, 90, 67, 33, 78, 92, 100, 55]. Використовуючи filter(), відберіть тільки задовільні оцінки (≥60). До відібраного списку застосуйте map(), щоб перевести кожну оцінку в 12-бальну систему за формулою: new_grade = round((old_grade / 100) * 12). Результат – список цілих чисел.

```
    Вхід: [45, 85, 90, 67, 33, 78, 92, 100, 55] → Вихід: [10, 11, 8, 9, 11, 12]
    *(Пояснення: ≥60 це [85,90,67,78,92,100] → у 12-бальній: [10,11,8,9,11,12])*
    Вхід: [100, 60, 0] → Вихід: [12, 7]
    Вхід: [59, 58, 57] → Вихід: []
```

Завдання 1.8. Рекурсія: Сума чисел від 1 до N

Напишіть рекурсивну функцію sum_to_n(n), яка обчислює суму всіх натуральних чисел від 1 до n включно. Для n <= 0 функція повинна повертати 0.

```
    Вхід: 5 → Вихід: 15 (1+2+3+4+5)
    Вхід: 1 → Вихід: 1
    Вхід: -3 → Вихід: 0
```

Завдання 1.9. Рекурсія: Кількість цифр у числі

Напишіть рекурсивну функцію count_digits(n), яка підраховує кількість цифр у цілому невід'ємному числі n. Не використовуйте цикли та перетворення на рядок.

```
    Вхід: 12345 → Вихід: 5
    Вхід: 7 → Вихід: 1
    Вхід: 0 → Вихід: 1
```

Завдання 1.10. Рекурсія: Зворотній порядок цифр

Напишіть рекурсивну функцію reverse_number(n), яка приймає натуральне число і повертає число, складене з його цифр у зворотному порядку. Не використовуйте цикли, рядки або списки.

```
    Вхід: 1234 → Вихід: 4321
    Вхід: 1000 → Вихід: 1 (обережно з нулями на кінці)
    Вхід: 5 → Вихід: 5
```

Частина 2. Середні завдання

Завдання 2.1. args: середнє значення довільної кількості чисел

Напишіть функцію flexible_average(*args), яка обчислює середнє арифметичне від довільної кількості переданих чисел. Якщо аргументи не передані, функція має повернути None. Ігноруйте нечислові типи.

```
Вхід: flexible_average(10, 20, 30) → Вихід: 20.0
Вхід: flexible_average(5) → Вихід: 5.0
Вхід: flexible_average() → Вихід: None
Вхід: flexible_average(1, 'a', 3) → Вихід: 2.0 (ігнорує 'a')
```

Завдання 2.2. kwargs: створення HTML-тегу

Створіть функцію make_html_tag(tag_name, content, **attributes), яка повертає рядок HTML-тегу. Параметр tag_name – назва тегу (напр., "div", "a"), content – вміст тегу. Всі інші іменовані аргументи (**attributes) повинні бути додані як атрибути тегу (напр., class="menu", href="#").

```
Вхід: make_html_tag('p', 'Hello World') → <p>Hello World</p>
Вхід: make_html_tag('a', 'Click here', href="https://example.com", target="_blank") → <a href="https://example.com" target="_blank">Click here</a>
Вхід: make_html_tag('div', '', id="container", class_="main") → <div id="container" class="main"></div>
(Примітка: class - зарезервоване слово, тому використовуйте class_)
```

Завдання 2.3. Рекурсія: пошук максимального елемента в списку

Напишіть рекурсивну функцію find_max(lst), яка знаходить максимальний елемент у списку чисел (не порожньому). Не використовуйте вбудовану функцію max() або цикли.

```
Вхід: [3, 7, 2, 9, 1] → Вихід: 9
Вхід: [-5, -1, -10] → Вихід: -1
Вхід: [42] → Вихід: 42
```

Завдання 2.4. Функція вищого порядку: логування виклику

Створіть функцію logged(func), яка приймає іншу функцію func як аргумент і повертає нову функцію-обгортку. Ця обгортка має:

1. Виводити у консоль рядок "Calling function <func_name> with arguments <args> <kwargs>".

2. Викликати оригінальну функцію func з переданими аргументами.

3. Виводити у консоль рядок "Function <func_name> returned <result>".

4. Повертати результат виклику func.

Застосуйте logged до простої функції додавання add(a, b) та продемонструйте її роботу.

```
Приклад роботи:
@logged # або add = logged(add)
def add(a, b):
    return a + b
add(5, 3)
# В консолі має з'явитись:
# Calling function add with arguments (5, 3) {}
# Function add returned 8
```

Завдання 2.5. Комбінація map, filter, lambda: аналіз даних

Дано список словників з інформацією про студентів: students = [{'name': 'Anna', 'age': 22, 'avg_grade': 91}, {'name': 'Bob', 'age': 19, 'avg_grade': 78}, ...].

Використовуючи ланцюжок filter() та map(), створіть список імен студентів, які старші за 20 років *і* мають середній бал вище 85. Використовуйте lambda-функції.

```
Приклад роботи:
students = [{'name':'Anna','age':22,'avg_grade':91},
{'name':'Bob','age':19,'avg_grade':78},
{'name':'Charlie','age':23,'avg_grade':88}]
['Anna', 'Charlie'] (Bob не підходить за віком, Charlie має бал >85)
```

Частина 3. Складні завдання

Завдання 3.1. Складна рекурсія: генерація всіх перестановок

Напишіть рекурсивну функцію permutations(items), яка приймає список items і повертає список всіх можливих перестановок (permutations) його елементів. Елементи унікальні. Не використовуйте модуль itertools.

Підказка: Рекурсивний підхід: для кожного елемента списку, відокремте його, згенеруйте всі перестановки решти списку і

приклейте відокремлений елемент на початок кожної з цих перестановок.

```
    Вхід: [1, 2, 3] → Вихід: [[1,2,3], [1,3,2], [2,1,3], [2,3,1],
[3,1,2], [3,2,1]]
    Вхід: ['a'] → Вихід: [['a']]
    Вхід: [] → Вихід: [[]]
```

Завдання 3.2. Функціональна обробка дерева (рекурсія + map)

Реалізуйте структуру дерева за допомогою словників: вузол = {'value': value, 'children': [node1, node2, ...]}. Напишіть **рекурсивну** функцію map_tree(node, func), яка приймає корінь дерева node та функцію-перетворювач func, і повертає **нове дерево** такої ж структури, але в якому значення value кожного вузла замінено на результат виклику func(old_value).

Не змінюйте оригінальне дерево. Використовуйте map() для обробки списку дітей.

```
    Приклад дерева:
    tree = {
        'value': 1,
        'children': [
            {'value': 2, 'children': []},
            {'value': 3, 'children': [
                {'value': 4, 'children': []}
            ]}
        ]
    }
    Вхід (tree, lambda x: x*2) → Вихід (нове дерево):
    {
        'value': 2,
        'children': [
            {'value': 4, 'children': []},
            {'value': 6, 'children': [
                {'value': 8, 'children': []}
            ]}
        ]
    }
```

Завдання 3.3. Мемоізація рекурсивної функції за допомогою декоратора

Напишіть **декоратор** memoize. Він повинен зберігати (кешувати) результати викликів функції, яку він декорує, для кожного унікального набору аргументів. При повторному виклику з тими самими аргументами декоратор має повертати значення з кешу, не викликаючи оригінальну функцію.

Застосуйте цей декоратор до рекурсивної функції обчислення чисел Фібоначчі fib(n) і продемонструйте значне прискорення обчислення для n=35 порівняно з немемоізованою версією.

Вимоги до декоратора:

1. Кеш - словник, ключі - кортежі аргументів.
2. Працює з функціями, що приймають будь-яку кількість позиційних аргументів.
3. Не використовує глобальні змінні для кешу (кожна функція має свій кеш).

Приклад роботи:

```
@memoize
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)
print(fib(35)) # Швидкий результат
```

6. Питання для самоперевірки

1. Поясніть основну відмінність між звичайною функцією (def) та lambda-функцією. Наведіть приклад ситуації, коли використання lambda є більш доцільним.
2. Що повертають функції map() та filter() в Python 3? Як отримати з результатів їх роботи звичайний список?
3. Для чого використовують аргументи *args та **kwargs в оголошенні функції? Який тип даних має args, а який – kwargs всередині функції?
4. Опишіть два обов’язкові елементи кожної коректної рекурсивної функції. Чому відсутність будь-якого з них призведе до помилки?
5. Що таке "базовий випадок" (base case) у рекурсії? Наведіть приклад для рекурсивної функції обчислення факторіалу.
6. Яку проблему може викликати рекурсивна реалізація чисел Фібоначчі (наприклад, fib(40))? Який існує загальний підхід до її Рішення?

7. Що означає, що "функції в Python є об’єктами першого класу"? Наведіть три приклади операцій, які можна робити з функціями, як з об’єктами.

8. Що таке функція вищого порядку (higher-order function)? Назвіть три вбудовані функції вищого порядку в Python та опишіть їхню роботу.

9. Як працює функція reduce() з модуля functools? Наведіть приклад її використання для знаходження добутку елементів списку.

10. Що таке "замикання" (closure) у контексті функцій? Наведіть короткий приклад функції, яка повертає іншу функцію, що "пам’ятає" значення зовнішньої змінної.

11. Яку роль відіграє ключове слово key у функції sorted()? Напишіть приклад сортування списку рядків за довжиною з використанням lambda.

12. Чим може бути небезпечна рекурсія з великою глибиною? Яке стандартне обмеження в Python і як його можна змінити (чому це рідко є хорошою ідеєю)?

13. У чому різниця між lambda x: x**2 та def square(x): return x**2 з точки зору області видимості та можливості налагодження?

14. Як можна імітувати роботу map() за допомогою list comprehension? Наведіть приклад перетворення списку чисел на список їх квадратів двома способами.

15. Що виведе наступний код та чому?

```
def maker(n):
    return lambda x: x ** n
square = maker(2)
cube = maker(3)
print(square(5), cube(5))
```

16. Який результат виконання цього фрагмента? Поясніть логіку роботи filter() з None.

```
data = [1, 0, 'hello', '', True, False, [], [1,2]]
result = list(filter(None, data))
print(result)
```

17. Що таке декоратор функції? Опишіть механізм його роботи на простому прикладі (без використання синтаксису @).

18. При рекурсивному обході дерева чи списку, який підхід зазвичай ефективніший: рекурсія чи ітерація (цикли)? Аргументуйте свою відповідь.

19. Як можна передати список або словник у функцію, яка очікує набір позиційних чи іменованих аргументів (*args та **kwargs)? Наведіть приклад синтаксису розпакування.

20. Розкажіть про потенційну проблему використання mutable об’єктів (наприклад, списків) за замовчуванням в аргументах функції. Наведіть приклад безпечної альтернативи.

ЛАБОРАТОРНА РОБОТА №11
Модулі та робота з файлами

1. Мета

Опанувати принципи модульності в програмуванні та навчитися базовим операціям читання та запису даних у файли в Python. Студенти повинні набути практичних навичок у створенні та використанні власних модулів, використанні вбудованих модулів Python (math, random, datetime), а також ефективній та безпечній роботі з текстовими файлами, включаючи обробку помилок та структурованих даних.

2. Завдання

1. Навчитися створювати власні модулі та імпортувати їх різними способами.
2. Опанувати використання вбудованих модулів math, random та datetime для Рішення прикладних задач.
3. Оволодіти основними методами читання з текстових файлів (read, readline, readlines).
4. Навчитися записувати дані у файли за допомогою методів write та writelines.
5. Зрозуміти та застосовувати контекстний менеджер with для безпечної роботи з файлами.
6. Набути навичок обробки помилок (винятків) при роботі з файловою системою.
7. Закріпити вміння читати та записувати структуровані дані (наприклад, списки, словники) у файли.
8. Ознайомитися з підходами до обробки великих файлів, що не поміщаються в оперативну пам’ять.

3. Короткі теоретичні відомості

3.1. Концепція модульності

Модуль у Python – це просто файл з розширенням .py, що містить визначення та оператори на Python (функції, класи, змінні). Модульність дозволяє розбивати великі програми на логічні, легко керовані частини, що можуть повторно використовуватись.

3.2. Створення та імпорт власних модулів

Створити модуль просто: достатньо написати код у файлі, наприклад, my_utils.py. Щоб використати цей код в іншій програмі, потрібно його імпортувати.

Основні способи імпорту:

1. **Повний імпорт модуля:** import my_utils. Доступ до функцій через крапку: my_utils.my_function().
2. **Імпорт конкретних об'єктів:** from my_utils import my_function, my_variable. Доступ безпрефіксний: my_function().
3. **Імпорт з псевдонімом (alias):** import my_utils as mu або from math import sqrt as square_root.

3.3. Вбудовані модулі Python

- math**:** Надає математичні функції (sin, cos, sqrt, log, ceil, floor, pi, e).

```
import math
radius = 5
area = math.pi * math.pow(radius, 2)
print(f"Площа кола: {area:.2f}")
```

- random**:** Генерація псевдовипадкових чисел та операції випадкового вибору (randint, uniform, choice, shuffle).

```
import random
random_number = random.randint(1, 10)   # Ціле число від 1 до 10
cards = ['Туз', 'Король', 'Дама']
random.shuffle(cards)  # Перемішати список
```

- datetime**:** Робота з датами та часом. Ключовий клас – datetime.

```
from datetime import datetime, timedelta
now = datetime.now()
print(f"Поточна дата та час: {now}")
tomorrow = now + timedelta(days=1)  # Додати 1 день
```

3.4. Робота з файлами в Python

Відкриття файлу: функція open(path, mode) повертає файловий об'єкт.

- **Режими (mode):**
  - 'r' – читання (за замовчуванням).
  - 'w' – запис (стирає існуючий файл або створює новий).

- ‘a’ – дозапис в кінець файлу.
- ‘x’ – ексклюзивне створення (помилка, якщо файл існує).
- ‘t’ – текстовий режим (за замовчуванням).
- ‘b’ – бінарний режим.

Методи читання:

- read([size]) – читає весь файл або size символів/байт.
- readline() – читає одну рядок.
- readlines() – читає всі рядки у список.

Методи запису:

- write(string) – записує рядок у файл.
- writelines(list_of_strings) – записує список рядків.

Контекстний менеджер with

Найкращий спосіб роботи з файлами. Автоматично закриває файл навіть у разі помилки.

```
with open('data.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    # Файл закриється автоматично тут
```

3.5. Обробка помилок (винятків)

При роботі з файлами можуть виникати помилки (файл не знайдено, недостатньо прав). Їх слід обробляти за допомогою try...except.

```
try:
    with open('missing.txt', 'r') as f:
        data = f.read()
except FileNotFoundError:
    print("Файл не знайдено! Перевірте шлях.")
except IOError as e:
    print(f"Сталася помилка вводу-виводу: {e}")
```

3.6. Читання та запис структурованих даних

Часто дані у файлах мають структуру (наприклад, кожен рядок – запис). Для роботи з ними зручно використовувати списки.

```
# Запис списку у файл (кожен елемент з нового рядка)
cities = ['Київ', 'Львів', 'Одеса']
with open('cities.txt', 'w') as f:
    for city in cities:
        f.write(city + '\n')
```

```
# Читання списку з файлу
with open('cities.txt', 'r') as f:
    loaded_cities = [line.strip() for line in f.readlines()]
```

Обробка великих файлів

Щоб не завантажувати великий файл цілком у пам'ять, його читають по частинах: рядок за рядком або блоками.

```
# Читання великого файлу рядок за рядком
with open('huge_log.txt', 'r') as file:
    for line in file:  # Ітерація без readlines()
        process_line(line)  # Обробка одного рядка
```

Таблиця: Порівняння методів читання файлів

Метод	Переваги	Недоліки	Найкраще використовувати для...
read()	Простота, весь вміст у змінній	Використовує багато пам'яті для великих файлів	Невеликих файлів (до декількох МБ)
readline()	Мале використання пам'яті	Необхідно керувати циклом читання	Пошуку конкретної інформації або обробки рядків за умовами
readlines()	Зручність роботи зі списком рядків	Використовує багато пам'яті для великих файлів	Файлів середнього розміру, де потрібен список усіх рядків
Ітерація (for line in file:)	Дуже економічний за пам'яттю, простий синтаксис	Немає прямого доступу до конкретного рядка за індексом	Обробки великих файлів, лог-файлів, посторінкового читання

4. Методичні рекомендації

Задача 1. Створення власного модуля для математичних операцій

Створіть власний модуль geometry.py, який містить функції для обчислення площі та периметра основних геометричних фігур (квадрат, прямокутник, коло). У головній програмі імпортуйте цей модуль різними способами та використайте функції.

```
    # geometry.py
    """Модуль для обчислення геометричних характеристик фігур."""

    import math

    def square_area(side: float) -> float:
        """Повертає площу квадрата за стороною."""
        return side ** 2

    def square_perimeter(side: float) -> float:
        """Повертає периметр квадрата за стороною."""
        return 4 * side

    def rectangle_area(length: float, width: float) -> float:
        """Повертає площу прямокутника за довжиною та шириною."""
        return length * width

    def circle_area(radius: float) -> float:
        """Повертає площу кола за радіусом."""
        return math.pi * radius ** 2

    # main.py
    # Спосіб 1: Повний імпорт
    import geometry

    side = 5
    print(f"Площа квадрата зі стороною {side}:
{geometry.square_area(side)}")

    # Спосіб 2: Імпорт конкретних функцій
    from geometry import rectangle_area, square_perimeter

    length, width = 4, 6
```

```
    print(f"Площа прямокутника {length}x{width}: {rectangle_area(length, width)}")
    print(f"Периметр квадрата зі стороною {side}: {square_perimeter(side)}")

    # Спосіб 3: Імпорт з псевдонімом
    import geometry as geom

    radius = 3.5
    print(f"Площа кола з радіусом {radius}: {geom.circle_area(radius):.2f}")
```

Приклад роботи:

```
Вхід: side = 5, length = 4, width = 6, radius = 3.5
Вихід:
Площа квадрата зі стороною 5: 25
Площа прямокутника 4x6: 24
Периметр квадрата зі стороною 5: 20
Площа кола з радіусом 3.5: 38.48
```

Коментарі:

- Модуль повинен містити докстрінгу з описом призначення
- Типізація (type hints) покращує читаність коду
- Різні способи імпорту демонструють гнучкість Python
- Використання math.pi забезпечує високу точність обчислень

Задача 2. Генерація паролів з використанням модуля random

Напишіть функцію, яка генерує випадковий пароль заданої довжини. Пароль повинен містити літери (великі та малі), цифри та спеціальні символи. Використайте модуль random.

```
    import random
    import string

    def generate_password(length: int = 12) -> str:
        """Генерує випадковий пароль заданої довжини."""
        # Визначаємо символи для паролю
        characters = string.ascii_letters + string.digits + "!@#$%^&*()_+-=[]{}|;:,.<>?"

        # Перевіряємо мінімальну довжину
        if length < 8:
```

```
            raise ValueError("Пароль повинен мати довжину не менше
8 символів")

        # Генеруємо пароль
        password = ''.join(random.choice(characters) for _ in
range(length))

        # Перевіряємо, що пароль містить принаймні по одному
символу кожного типу
        if (any(c.islower() for c in password) and
            any(c.isupper() for c in password) and
            any(c.isdigit() for c in password) and
            any(c in "!@#$%^&*()_+-=[]{}|;:,.<>?" for c in
password)):
            return password
        else:
            # Якщо умови не виконані, генеруємо знову
            return generate_password(length)

    # Демонстрація роботи
    print("Пароль 1:", generate_password(10))
    print("Пароль 2:", generate_password(16))
    print("Пароль 3:", generate_password(8))
```

Приклад роботи:

```
Вхід: generate_password(10)
Вихід: "kL8@jQ#9pM"

Вхід: generate_password(8)
Вихід: "A1b!C2d#"
Вхід: generate_password(12)
Вихід: "xY3$zW8*vP7q"
```

Коментарі:

- Модуль string містить корисні константи для роботи з символами
- Використання random.choice() для випадкового вибору символів
- Рекурсивний виклик забезпечує дотримання вимог до паролю
- Додана перевірка мінімальної довжини паролю

Задача 3. Аналіз текстового файлу з підрахунком статистики

Створіть програму, яка читає текстовий файл та обчислює: загальну кількість слів, кількість унікальних слів, кількість рядків, та найдовше слово. Використайте контекстний менеджер with та обробку помилок.

```
    def analyze_text_file(filename: str) -> dict:
        """
        Аналізує текстовий файл та повертає статистику.

        Returns:
            dict: Словник зі статистикою файлу
        """
        try:
            with open(filename, 'r', encoding='utf-8') as file:
                lines = file.readlines()

                if not lines:  # Файл порожній
                    return {
                        'total_lines': 0,
                        'total_words': 0,
                        'unique_words': 0,
                        'longest_word': ''
                    }

                total_lines = len(lines)
                all_words = []

                # Розбиваємо кожен рядок на слова
                for line in lines:
                    # Видаляємо знаки пунктуації та перетворюємо
на нижній регістр
                    words = line.strip().lower().split()
                    # Видаляємо знаки пунктуації з кожного слова
                    clean_words = [word.strip('.,!?;:"()[]{}') for
word in words]
                    all_words.extend(clean_words)

                total_words = len(all_words)
                unique_words = len(set(all_words))
                longest_word = max(all_words, key=len) if
all_words else ''

                return {
                    'total_lines': total_lines,
                    'total_words': total_words,
```

```
                'unique_words': unique_words,
                'longest_word': longest_word
            }

    except FileNotFoundError:
        print(f"Помилка. файл '{filename}' не знайдено.")
        return {}
    except IOError as e:
        print(f"Помилка вводу-виводу: {e}")
        return {}

# Тестування функції
# Спочатку створимо тестовий файл
test_content = """Python - потужна мова програмування.
Python популярна для веб-розробки та аналізу даних.
Python має простий синтаксис та велику спільноту."""

with open('test_file.txt', 'w', encoding='utf-8') as f:
    f.write(test_content)

# Аналізуємо файл
stats = analyze_text_file('test_file.txt')
if stats:
    print("Статистика файлу:")
    print(f"Рядків: {stats['total_lines']}")
    print(f"Слів: {stats['total_words']}")
    print(f"Унікальних слів: {stats['unique_words']}")
    print(f"Найдовше слово: '{stats['longest_word']}'")
```

Приклад роботи:

```
Вхід (файл test_file.txt):
Python - потужна мова програмування.
Python популярна для веб-розробки та аналізу даних.
Python має простий синтаксис та велику спільноту.

Вихід:
Статистика файлу:
Рядків: 3
Слів: 20
Унікальних слів: 15
Найдовше слово: 'програмування'
```

Коментарі:

- Контекстний менеджер with гарантує коректне закриття файлу

- Кодування utf-8 забезпечує коректну роботу з українськими символами
- Обробка винятків FileNotFoundError та IOError робить програму стійкою
- Очищення слів від пунктуації покращує точність аналізу

Задача 4. Фільтрація та запис структурованих даних у файл

Напишіть програму, яка читає файл зі списком студентів (у кожному рядку: ім'я, прізвище, оцінка), фільтрує студентів з оцінкою вище заданого порогу, сортує їх за прізвищем та записує результат у новий файл.

```
    def filter_students(input_file: str, output_file: str,
min_grade: float) -> None:
        """
        Фільтрує студентів з оцінкою вище min_grade та записує у
новий файл.
        Args:
            input_file: шлях до вхідного файлу
            output_file: шлях до вихідного файлу
            min_grade: мінімальна оцінка для фільтрації
        """
        try:
            students = []

            # Читання та обробка даних
            with open(input_file, 'r', encoding='utf-8') as
infile:
                for line in infile:
                    line = line.strip()
                    if not line:  # Пропускаємо порожні рядки
                        continue

                    # Розбиваємо рядок на частини
                    parts = line.split(',')
                    if len(parts) >= 3:
                        first_name = parts[0].strip()
                        last_name = parts[1].strip()
                        try:
                            grade = float(parts[2].strip())
                            students.append((last_name,
first_name, grade))
                        except ValueError:
```

```
                        print(f"Попередження:        некоректна
оцінка у рядку: {line}")

            # Фільтрація студентів
            filtered_students = [
                student for student in students
                if student[2] >= min_grade
            ]

            # Сортування за прізвищем
            filtered_students.sort(key=lambda x: x[0])

            # Запис результату
            with   open(output_file,   'w',   encoding='utf-8')   as
outfile:
                outfile.write("Прізвище, Ім'я, Оцінка\n")
                outfile.write("-" * 30 + "\n")

                for     last_name,     first_name,     grade     in
filtered_students:
                    outfile.write(f"{last_name},      {first_name},
{grade:.1f}\n")

            print(f"Знайдено  {len(filtered_students)}  студентів  з
оцінкою >= {min_grade}")
            print(f"Результат записано у файл: {output_file}")

        except FileNotFoundError:
            print(f"Помилка. файл '{input_file}' не знайдено.")
        except Exception as e:
            print(f"Сталася неочікувана Помилка. {e}")

    # Створення тестового файлу
    test_data = """Іван, Петренко, 85.5
    Марія, Сидоренко, 92.0
    Олег, Коваленко, 78.0
    Наталія, Іваненко, 95.5
    Андрій, Мельник, 88.0
    Юлія, Шевченко, 73.5
    """

    with open('students.txt', 'w', encoding='utf-8') as f:
        f.write(test_data)
```

```
# Виклик функції
filter_students('students.txt', 'good_students.txt', 85.0)
```

Приклад роботи:

```
Вхід (файл students.txt):
Іван, Петренко, 85.5
Марія, Сидоренко, 92.0
Олег, Коваленко, 78.0
Наталія, Іваненко, 95.5
Андрій, Мельник, 88.0
Юлія, Шевченко, 73.5

Вихід (файл good_students.txt):
Прізвище, Ім'я, Оцінка
------------------------------
Іваненко, Наталія, 95.5
Мельник, Андрій, 88.0
Петренко, Іван, 85.5
Сидоренко, Марія, 92.0
```

Коментарі:

- Використання кортежів для зберігання структурованих даних
- Обробка можливих помилок у вхідних даних (некоректні оцінки)
- Лямбда-функція для сортування за прізвищем
- Форматований вивід у файл з заголовками та роздільниками

Задача 5. Робота з датами та часом (модуль datetime)

Створіть програму для менеджменту завдань, яка дозволяє додавати завдання з дедлайном, відображати завдання на сьогодні та обчислювати залишок часу до дедлайну.

```
from datetime import datetime, timedelta
import json

class TaskManager:
    """Менеджер завдань з дедлайнами."""

    def __init__(self, filename: str = 'tasks.json'):
        self.filename = filename
        self.tasks = self._load_tasks()

    def _load_tasks(self) -> list:
        """Завантажує завдання з JSON-файлу."""
        try:
```

```
                with open(self.filename, 'r', encoding='utf-8') as file:
                    tasks = json.load(file)
                    # Конвертуємо рядки дат назад у об'єкти datetime
                    for task in tasks:
                        task['deadline'] =
datetime.fromisoformat(task['deadline'])
                    return tasks
            except (FileNotFoundError, json.JSONDecodeError):
                return []

        def _save_tasks(self) -> None:
            """Зберігає завдання у JSON-файл."""
            # Конвертуємо datetime у рядок для JSON
            tasks_to_save = []
            for task in self.tasks:
                task_copy = task.copy()
                task_copy['deadline'] = task['deadline'].isoformat()
                tasks_to_save.append(task_copy)

            with open(self.filename, 'w', encoding='utf-8') as file:
                json.dump(tasks_to_save, file, ensure_ascii=False,
indent=2)

        def add_task(self, title: str, description: str,
days_until_deadline: int) -> None:
            """Додає нове завдання."""
            deadline = datetime.now() + timedelta(days=days_until_deadline)

            task = {
                'title': title,
                'description': description,
                'deadline': deadline,
                'created': datetime.now()
            }

            self.tasks.append(task)
            self._save_tasks()
            print(f"Завдання '{title}' додано. Дедлайн:
{deadline.strftime('%d.%m.%Y %H:%M')}")

        def show_today_tasks(self) -> None:
            """Показує завдання, дедлайн яких сьогодні."""
            today = datetime.now().date()
            today_tasks = [
                task for task in self.tasks
                if task['deadline'].date() == today
            ]
```

```
            if not today_tasks:
                print("На сьогодні завдань немає.")
                return

            print(f"\nЗавдання на сьогодні ({today.strftime('%d.%m.%Y')}):")
            print("-" * 50)
            for i, task in enumerate(today_tasks, 1):
                time_left = task['deadline'] - datetime.now()
                hours_left = int(time_left.total_seconds() // 3600)
                minutes_left = int((time_left.total_seconds() % 3600) //
60)

                print(f"{i}. {task['title']}")
                print(f"   Опис: {task['description']}")
                print(f"Залишилось часу: {hours_left} год. {minutes_left}
хв")
                print()

        def show_all_tasks(self) -> None:
            """Показує всі завдання з інформацією про дедлайн."""
            if not self.tasks:
                print("Завдань немає.")
                return

            print("\nВсі завдання:")
            print("-" * 50)

            for i, task in enumerate(self.tasks, 1):
                deadline_str= task['deadline'].strftime('%d.%m.%Y %H:%M')
                time_left = task['deadline'] - datetime.now()

                if time_left.total_seconds() < 0:
                    status = "ПРОСРОЧЕНО"
                elif time_left.total_seconds() < 86400:    # Менше 24
годин
                    status = "ТЕРМІНОВО"
                else:
                    status = "АКТИВНЕ"

                days_left=     max(0,     int(time_left.total_seconds()//
86400))

                print(f"{i}. {task['title']} [{status}]")
                print(f"   Дедлайн: {deadline_str}")
                print(f"   Залишилось днів: {days_left}")
                print()

    # Демонстрація роботи
```

```
if __name__ == "__main__":
    manager = TaskManager()

    # Додаємо тестові завдання
    manager.add_task("Лабораторна робота", "Модулі та файли", 3)
    manager.add_task("Проект", "Розробка програми", 7)
    manager.add_task("Звіт", "Аналіз результатів", 1)

    # Показуємо завдання на сьогодні
    manager.show_today_tasks()

    # Показуємо всі завдання
    manager.show_all_tasks()
```

Приклад роботи:

```
  Завдання 'Лабораторна робота' додано. Дедлайн: 16.01.2026
14:30
  Завдання 'Проект' додано. Дедлайн: 20.01.2026 14:30
  Завдання 'Звіт' додано. Дедлайн: 14.01.2026 14:30

  Завдання на сьогодні (13.01.2026):
  --------------------------------------------------
  1. Звіт
     Опис: Аналіз результатів
     Залишилось часу: 9 год. 30 хв.
  Всі завдання:
  --------------------------------------------------
  1. Лабораторна робота [АКТИВНЕ]
     Дедлайн: 16.01.2026 14:30
     Залишилось днів: 3
  2. Проект [АКТИВНЕ]
     Дедлайн: 20.01.2026 14:30
     Залишилось днів: 7
  3. Звіт [ТЕРМІНОВО]
     Дедлайн: 14.01.2026 14:30
     Залишилось днів: 1
```

Коментарі:

- Використання класу для інкапсуляції логіки роботи з завданнями
- JSON для збереження структурованих даних між запусками програми
- Методи fromisoformat() та isoformat() для роботи з датами в JSON
- Обчислення різниці між датами за допомогою timedelta

Задача 6. Обробка великого файлу по частинах

Напишіть програму, яка читає дуже великий текстовий файл (мінімум 100 МБ) по частинах, знаходить найдовший рядок та підраховує загальну кількість символів, не завантажуючи весь файл в пам'ять.

```
    def process_large_file(filename: str, chunk_size: int = 8192)
-> tuple:
        """
        Обробляє великий файл по частинах.

        Args:
            filename: шлях до файлу
            chunk_size: розмір блоку читання в байтах

        Returns:
            tuple: (довжина найдовшого рядка, загальна кількість
символів)
        """
        try:
            max_line_length = 0
            total_chars = 0
            buffer = ""

            with open(filename, 'r', encoding='utf-8') as file:
                while True:
                    # Читаємо блок даних
                    chunk = file.read(chunk_size)
                    if not chunk:  # Кінець файлу
                        break

                    # Додаємо до буфера
                    buffer += chunk

                    # Шукаємо повні рядки у буфері
                    lines = buffer.splitlines(keepends=True)

                    # Залишаємо останній неповний рядок у буфері
                    if buffer and buffer[-1] != '\n':
                        buffer = lines.pop() if lines else ""
                    else:
                        buffer = ""

                    # Обробляємо повні рядки
                    for line in lines:
```

```
                        line_length = len(line.rstrip('\n'))
                        max_line_length    =    max(max_line_length,
line_length)
                        total_chars += len(line)

            # Обробляємо останній рядок, якщо він є
            if buffer:
                line_length = len(buffer)
                max_line_length        =        max(max_line_length,
line_length)
                total_chars += len(buffer)

            return max_line_length, total_chars

        except FileNotFoundError:
            print(f"Помилка. файл '{filename}' не знайдено.")
            return 0, 0
        except MemoryError:
            print("Помилка пам'яті. Спробуйте зменшити chunk_size.")
            return 0, 0

    # Демонстрація на меншому файлі
    def create_test_file():
        """Створює тестовий файл для демонстрації."""
        import random
        import string
        with open('large_test.txt', 'w', encoding='utf-8') as f:
            # Генеруємо 10000 рядків різної довжини
            for i in range(10000):
                # Випадкова довжина рядка від 10 до 200 символів
                line_length = random.randint(10, 200)
                # Створюємо випадковий рядок
                line=''.join(random.choice(string.ascii_letters+' ')
                             for _ in range(line_length))
                f.write(line + '\n')

                # Додаємо один дуже довгий рядок
                if i == 5000:
                    long_line = 'X' * 500 + '\n'
                    f.write(long_line)

    # Створюємо тестовий файл
    create_test_file()
```

```
    # Обробляємо файл
    max_length, total_chars = process_large_file('large_test.txt',
chunk_size=4096)

    print(f"Найдовший рядок: {max_length} символів")
    print(f"Загальна кількість символів: {total_chars}")
    print(f"Середня довжина рядка: {total_chars / 10001:.1f}
символів")
```

Приклад роботи:

```
Вхід: файл large_test.txt з 10001 рядком
Вихід:
Найдовший рядок: 500 символів
Загальна кількість символів: 1057892
Середня довжина рядка: 105.8 символів
```

Коментарі:

- Читання файлу блоками фіксованого розміру забезпечує економію пам'яті
- Буферизація рядків коректно обробляє рядки, які розділені між блоками
- Параметр keepends=True зберігає символи нового рядка для точного підрахунку
- Метод працює навіть з файлами, які перевищують об'єм оперативної пам'яті

Задача 7. Створення пакету модулів для роботи з файлами

Створіть структуру пакету file_utils з декількома модулями для роботи з файлами: один модуль для читання, один для запису, один для аналізу. Продемонструйте використання пакету у головній програмі.

```
    # Структура пакету:
    # file_utils/
    # |   __init__.py
    # |   file_reader.py
    # |   file_writer.py
    # |   file_analyzer.py

    # file_utils/__init__.py
    """
    Пакет file_utils - утиліти для роботи з файлами.
    """
```

```
    from .file_reader import read_lines, read_chunks
    from .file_writer import write_lines, append_to_file
    from .file_analyzer import count_words, find_longest_line

    __all__ = ['read_lines', 'read_chunks', 'write_lines',
               'append_to_file', 'count_words', 'find_longest_line']

    # file_utils/file_reader.py
    """Модуль для читання файлів."""

    def read_lines(filename: str, encoding: str = 'utf-8') ->
list:
        """Читає всі рядки з файлу."""
        try:
            with open(filename, 'r', encoding=encoding) as file:
                return [line.rstrip('\n') for line in file]
        except FileNotFoundError:
            print(f"Файл {filename} не знайдено")
            return []

    def read_chunks(filename: str, chunk_size: int = 8192,
                    encoding: str = 'utf-8'):
        """Генератор, який читає файл блоками."""
        try:
            with open(filename, 'r', encoding=encoding) as file:
                while True:
                    chunk = file.read(chunk_size)
                    if not chunk:
                        break
                    yield chunk
        except FileNotFoundError:
            print(f"Файл {filename} не знайдено")
            yield ""

    # file_utils/file_writer.py
    """Модуль для запису у файли."""

    def write_lines(filename: str, lines: list, encoding: str =
'utf-8') -> bool:
        """Записує список рядків у файл."""
        try:
            with open(filename, 'w', encoding=encoding) as file:
                for line in lines:
                    file.write(str(line) + '\n')
```

```
            return True
        except IOError as e:
            print(f"Помилка запису у файл {filename}: {e}")
            return False

def append_to_file(filename: str, content: str,
                   encoding: str = 'utf-8') -> bool:
    """Додає вміст у кінець файлу."""
    try:
        with open(filename, 'a', encoding=encoding) as file:
            file.write(content + '\n')
        return True
    except IOError as e:
        print(f"Помилка дозапису у файл {filename}: {e}")
        return False

# file_utils/file_analyzer.py
"""Модуль для аналізу файлів."""

def count_words(text: str) -> int:
    """Підраховує кількість слів у тексті."""
    words = text.split()
    return len(words)

def find_longest_line(lines: list) -> tuple:
    """Знаходить найдовший рядок та його індекс."""
    if not lines:
        return "", -1

    longest_line = max(lines, key=len)
    longest_index = lines.index(longest_line)
    return longest_line, longest_index

# main.py - використання пакету
from file_utils import (
    read_lines, write_lines,
    append_to_file, count_words, find_longest_line
)

# Створюємо тестовий файл
test_data = [
    "Перший рядок файлу",
    "Другий, дещо довший рядок",
```

```
        "Третій рядок - найдовший з усіх наявних рядків у цьому
файлі",
        "Четвертий рядок",
        "П’ятий і останній рядок"
    ]
    write_lines('test_document.txt', test_data)
    # Читаємо та аналізуємо
    lines = read_lines('test_document.txt')
    print(f"Прочитано {len(lines)} рядків:")

    for i, line in enumerate(lines, 1):
        word_count = count_words(line)
        print(f"{i}. {line} (слів: {word_count})")

    # Знаходимо найдовший рядок
    longest_line, line_index = find_longest_line(lines)
    print(f"\nНайдовший     рядок     (№{line_index     +     1}):
{longest_line}")

    # Додаємо новий рядок
    append_to_file('test_document.txt', 'Шостий, доданий рядок')
    print("\nДодано новий рядок. Оновлений вміст:")

    updated_lines = read_lines('test_document.txt')
    for line in updated_lines:
        print(f"  - {line}")
```

Приклад роботи:

```
  Прочитано 5 рядків:
  1. Перший рядок файлу (слів: 3)
  2. Другий, дещо довший рядок (слів: 4)
  3. Третій рядок - найдовший з усіх наявних рядків у цьому
файлі (слів: 10)
  4. Четвертий рядок (слів: 2)
  5. П’ятий і останній рядок (слів: 4)

  Найдовший рядок (№3): Третій рядок - найдовший з усіх наявних
рядків у цьому файлі

  Додано новий рядок. Оновлений вміст:
    - Перший рядок файлу
    - Другий, дещо довший рядок
    - Третій рядок - найдовший з усіх наявних рядків у цьому
файлі
    - Четвертий рядок
```

```
- П'ятий і останній рядок
- Шостий, доданий рядок
```

Коментарі:

- Пакет дозволяє логічно групувати функціональність
- Файл __init__.py визначає публічний інтерфейс пакету
- Генератор read_chunks дозволяє ефективно обробляти великі файли
- Модульна структура полегшує тестування та супровід коду

Типові помилки і шляхи їх усунення

1. Неправильне кодування файлів

Проблема. UnicodeDecodeError при читанні файлів з кириличними символами.

Рішення. Завжди вказуйте кодування при відкритті файлу:

```
# Неправильно
with open('file.txt', 'r') as f:    # Використовується кодування за замовчуванням системи

# Правильно
with open('file.txt', 'r', encoding='utf-8') as f:
```

2. Незакриття файлу

Проблема. Використання файлу без контекстного менеджера може призвести до витоку ресурсів.

Рішення. Завжди використовуйте with:

```
# Ризиковано
file = open('data.txt', 'r')
content = file.read()
file.close()  # Можна забути

# Безпечно
with open('data.txt', 'r') as file:
    content = file.read()
# Файл закриється автоматично
```

3. Перезапис файлу замість додавання

Проблема. Випадкове використання режиму 'w' замість 'a'.

Рішення. Уважно вибирайте режим відкриття файлу:

```
# СТИРАЄ існуючий вміст
with open('log.txt', 'w') as f:
    f.write('Новий запис')
```

```
# ДОДАЄ до існуючого вмісту
with open('log.txt', 'a') as f:
    f.write('Додатковий запис')
```

4. Читання великих файлів цілком

Проблема. MemoryError при спробі прочитати великий файл за допомогою read() або readlines().

Рішення. Використовуйте ітерацію по рядках або читання блоками:

```
# Небезпечно для великих файлів
with open('huge_file.txt', 'r') as f:
    lines = f.readlines()  # Весь файл у пам'яті!

# Безпечно
with open('huge_file.txt', 'r') as f:
    for line in f:  # Читає по одному рядку
        process_line(line)
```

5. Відсутність обробки помилок

Проблема. Програма падає, якщо файл не існує або немає прав доступу.

Рішення. Завжди використовуйте try-except:

```
try:
    with open('important.txt', 'r') as f:
        data = f.read()
except FileNotFoundError:
    print("Файл не знайдено! Створіть файл або перевірте шлях.")
except PermissionError:
    print("Немає прав доступу до файлу.")
except IOError as e:
    print(f"Помилка вводу-виводу: {e}")
```

6. Неправильний шлях до файлу

Проблема. Файл не знаходиться за вказаним шляхом.

Рішення. Використовуйте абсолютні шляхи або перевіряйте поточну директорію:

```
import os

# Перевірка існування файлу
if os.path.exists('data.txt'):
    with open('data.txt', 'r') as f:
        # робота з файлом
else:
```

```
        print("Файл не існує")
```

Корисні поради

1. Використовуйте pathlib для роботи з шляхами

Замість модуля os для роботи з файловими шляхами, використовуйте сучасний pathlib:

```
from pathlib import Path

# Створення об'єкта Path
file_path = Path('data') / 'subfolder' / 'file.txt'

# Перевірка існування
if file_path.exists():
    # Читання
    content = file_path.read_text(encoding='utf-8')
    # Запис
    file_path.write_text('Новий вміст', encoding='utf-8')
```

2. Ефективне читання рядків зі збереженням пробілів

Для обробки файлів зі складним форматуванням:

```
# Збереження кінців рядків
with open('formatted.txt', 'r') as f:
    lines = f.readlines()  # Зберігає \n в кінці кожного рядка

# Видалення тільки правого пробілу
clean_lines = [line.rstrip() for line in lines]
```

3. Використання enumerate() при обробці файлів

При роботі з рядками файлу зручно отримувати номери рядків:

```
with open('data.txt', 'r') as file:
    for line_number, line in enumerate(file, start=1):
        if 'помилка' in line.lower():
            print(f"Потенційна помилка у рядку {line_number}:
{line.strip()}")
```

4. Оптимізація роботи з JSON

Для ефективної роботи з великими JSON-файлами:

```
import json

# Для великих файлів - завантаження потоком
with open('large_data.json', 'r') as f:
    data = json.load(f)  # Весь файл у пам'яті
```

```
    # Альтернатива для дуже великих файлів
    import ijson  # потрібно встановити: pip install ijson
    with open('huge_data.json', 'r') as f:
        # Обробка об'єктів по одному
        for item in ijson.items(f, 'item'):
            process_item(item)
```

5. Логування операцій з файлами

Додавайте логування для відстеження операцій з файлами:

```
    import logging

    logging.basicConfig(level=logging.INFO)
    logger = logging.getLogger(__name__)

    try:
        with open('important.dat', 'rb') as f:
            data = f.read()
        logger.info(f"Успішно прочитано файл important.dat, розмір: {len(data)} байт")
    except Exception as e:
        logger.error(f"Помилка читання файлу: {e}", exc_info=True)
```

6. Використання tempfile для тимчасових файлів

Для роботи з тимчасовими даними:

```
    import tempfile
    # Автоматичне видалення після використання
    with tempfile.NamedTemporaryFile(mode='w', delete=True, suffix='.txt') as tmp:
        tmp.write('Тимчасові дані')
        tmp.flush()
        # Робота з тимчасовим файлом
        # Файл автоматично видалиться після виходу з блоку with
```

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Генерація випадкових паролів

Напишіть програму, яка використовує модуль random для генерації 5 випадкових паролів довжиною 8 символів. Кожен пароль повинен містити принаймні одну велику літеру, одну малу літеру та одну цифру. Виведіть паролі у стовпчик.

```
Вхід: немає
```

```
Вихід:
1. A3bC7dEf
2. xY9zP5qR
3. mN8bV2cX
4. pQ1rS6tU
5. kL4jH0wZ
```

Завдання 1.2. Обчислення геометричних характеристик

Використовуючи модуль math, напишіть функцію, яка приймає радіус кола та повертає кортеж (площа, довжина кола). Округліть результати до двох знаків після коми.

```
Вхід: 5 → Вихід: (78.54, 31.42)
Вхід: 2.5 → Вихід: (19.63, 15.71)
Вхід: 10 → Вихід: (314.16, 62.83)
```

Завдання 1.3. Робота з датою народження

Використовуючи модуль datetime, напишіть програму, яка запитує рік, місяць та день народження користувача та обчислює:

1. Скільки років користувачеві
2. У який день тижня він народився
3. Скільки днів до наступного дня народження

```
Вхід: 2000, 5, 15 (сьогодні 13.01.2026)
Вихід:
Вік: 25 років
День народження: понеділок
Днів до наступного дня народження: 122

Вхід: 1995, 12, 31
Вихід:
Вік: 30 років
День народження: неділя
Днів до наступного дня народження: 352
```

Завдання 1.4. Статистика кидків кубика

Напишіть програму, яка імітує 1000 кидків шестигранного кубика за допомогою модуля random. Підрахуйте та виведіть частоту випадання кожної грані у відсотках.

```
Вхід: немає
Вихід:
Грань 1: 16.7%
```

```
Грань 2: 16.9%
Грань 3: 16.5%
Грань 4: 16.8%
Грань 5: 16.4%
Грань 6: 16.7%
(значення будуть випадковими)
```

Завдання 1.5. Підрахунок рядків у файлі

Створіть текстовий файл poem.txt з 5-7 рядками вірша. Напишіть програму, яка читає файл та підраховує:

1. Загальну кількість рядків
2. Загальну кількість слів

```
Вхід (файл poem.txt):
В небі ясному сонце сяє,
Пташки весело щебечуть.
Квіти пестять нас запахом,
Вітер грає з листочками.
Життя - це дивовижна пісня.

Вихід:
Рядків: 5
Слів: 24
```

Завдання 1.6. Пошук найдовшого слова

Напишіть програму, яка читає файл text.txt та знаходить найдовше слово у файлі. Якщо таких слів декілька, виведіть перше знайдене. Ігноруйте розділові знаки.

```
Вхід (файл text.txt):
Програмування - це мистецтво створення алгоритмів.
Python є потужною та зрозумілою мовою для початківців.
Вихід:
Найдовше слово: "програмування" (13 символів)

Вхід:
Швидка бура лисиця стрибає через лінивого собаку.
Вихід:
Найдовше слово: "стрибає" (7 символів)
```

Завдання 1.7. Читання файлу за допомогою readline()

Створіть файл numbers.txt, де кожен рядок містить одне число. Напишіть програму, яка читає файл за допомогою методу readline() у циклі та обчислює суму всіх чисел.

```
Вхід (файл numbers.txt):
10
25
-5
17
8
Вихід: Сума чисел: 55

Вхід:
3.14
2.71
1.41
Вихід: Сума чисел: 7.26
```

Завдання 1.8. Створення файлу з таблицею множення

Напишіть програму, яка створює файл multiplication_table.txt і записує в нього таблицю множення від 1 до 10 у форматі "a × b = c", по 10 прикладів у рядку.

```
Вихід (фрагмент файлу):
1 × 1 = 1    1 × 2 = 2    1 × 3 = 3    ...    1 × 10 = 10
2 × 1 = 2    2 × 2 = 4    2 × 3 = 6    ...    2 × 10 = 20
...
10 × 1 = 10  10 × 2 = 20  10 × 3 = 30  ...    10 × 10 = 100
```

Завдання 1.9. Додавання записів до щоденника

Створіть програму "Електронний щоденник", яка дозволяє додавати записи з поточною датою та часом. Кожен новий запис додається у файл diary.txt у новий рядок у форматі "YYYY-MM-DD HH:MM:SS - [запис]".

```
Вхід (користувач вводить): "Сьогодні почав вивчати модулі в Python"
Вихід у файлі:
2026-01-13 14:30:15 - Сьогодні почав вивчати модулі в Python

Вхід: "Зробив лабораторну роботу"
Вихід у файлі (додається):
2026-01-13 14:30:15 - Сьогодні почав вивчати модулі в Python
2026-01-13 15:45:22 - Зробив лабораторну роботу
```

Завдання 1.10. Копіювання файлу з фільтрацією

Напишіть програму, яка читає файл source.txt та створює файл filtered.txt, записуючи тільки ті рядки, довжина яких перевищує 20 символів.

```
Вхід (файл source.txt):
Короткий рядок.
Цей рядок досить довгий для копіювання.
Ще один короткий.
Дуже довгий рядок з багатьма символами для тестування.

Вихід (файл filtered.txt):
Цей рядок досить довгий для копіювання.
Дуже довгий рядок з багатьма символами для тестування.
```

Частина 2. Середні завдання

Завдання 2.1. Модуль для обробки текстів

Створіть модуль text_processor.py з такими функціями:

1. count_vowels(text) - повертає кількість голосних літер
2. reverse_words(text) - повертає текст, де слова записані у зворотньому порядку
3. to_pig_latin(text) - перетворює кожне слово на піг-латин (перша літера переміщається в кінець + "ay")

Створіть головну програму, яка демонструє роботу всіх функцій.

```
Вхід: "Python це цікаво"
Вихід:
Голосних: 5
Зворотній порядок слів: "цікаво це Python"
Піг-латин: "ythonPay цеay ікавоцay"

Вхід: "Привіт світ"
Вихід:
Голосних: 3
Зворотній порядок слів: "світ Привіт"
Піг-латин: "рівитП світay"
```

Завдання 2.2. Калькулятор з історією

Створіть модуль calculator.py з класом Calculator, який зберігає історію обчислень. Клас повинен мати методи для основних операцій (+, -, *, /) та методи для збереження історії у файл і завантаження з файлу.

```
Вхідні операції:
calc.add(5, 3) → 8
calc.multiply(4, 2) → 8
calc.divide(10, 2) → 5.0

Вихід (файл history.txt):
2026-01-13 14:30:15 | 5 + 3 = 8
2026-01-13 14:30:16 | 4 * 2 = 8
2026-01-13 14:30:17 | 10 / 2 = 5.0
```

Завдання 2.3. Аналіз CSV-файлу з оцінками

Створіть програму для аналізу файлу grades.csv, де кожен рядок містить: ім'я студента, прізвище та 5 оцінок. Програма повинна:

1. Обчислити середній бал для кожного студента
2. Знайти студента з найвищим середнім балом
3. Обчислити середній бал по кожному предмету
4. Зберегти результати у файл report.txt

```
Вхід (grades.csv):
Іван,Петренко,85,90,78,92,88
Марія,Сидоренко,92,95,89,94,91
Олег,Коваленко,78,82,75,80,79

Вихід (report.txt):
СЕРЕДНІ БАЛИ СТУДЕНТІВ:
Іван Петренко: 86.6
Марія Сидоренко: 92.2
Олег Коваленко: 78.8

КРАЩИЙ СТУДЕНТ: Марія Сидоренко (92.2)

СЕРЕДНІ БАЛИ З ПРЕДМЕТІВ:
Предмет 1: 85.0
Предмет 2: 89.0
Предмет 3: 80.7
Предмет 4: 88.7
Предмет 5: 86.0
```

Завдання 2.4. Менеджер паролів з обробкою помилок

Створіть програму для зберігання паролів у зашифрованому вигляді. Програма повинна:

1. Зберігати пари "сайт - зашифрований пароль" у файл
2. Мати функції додавання, пошуку та видалення записів
3. Обробляти помилки: відсутність файлу, невірний формат даних, проблеми з доступом
4. Використовувати просте шифрування (наприклад, зсув символів)

```
Вхідні дії:
1. Додати запис: "google.com", "myPass123"
2. Додати запис: "github.com", "dev2024"
3. Знайти пароль для "google.com"

Вихід:
Додано запис для google.com
```

```
Додано запис для github.com
Пароль для google.com: myPass123

Вихід у файлі (encrypted):
google.com:nzQbtt234
github.com:efw4568
```

Завдання 2.5. Об'єднання та сортування файлів

Напишіть програму, яка читає два файли з числами (file1.txt та file2.txt), об'єднує їх в один відсортований список, видаляє дублікати та записує результат у файл merged.txt. Використовуйте контекстний менеджер with та обробку помилок.

```
Вхід (file1.txt):
15
3
42
7
15

Вхід (file2.txt):
7
23
42
9

Вихід (merged.txt):
3
7
9
15
23
42
```

Частина 3. Складні завдання

Завдання 3.1. Система моніторингу лог-файлів

Розробіть систему для аналізу веб-сервер логів. Програма повинна:

1. Читати великий лог-файл (server.log) по частинах
2. Підраховувати статистику: найпопулярніші сторінки, коди відповідей, джерела трафіку

3. Виявляти підозрілу активність (багато запитів з одного IP за короткий час)

4. Генерувати звіт у HTML-форматі з графіками (використати модуль datetime для аналізу часу)

5. Зберігати результати у окремі файли для подальшого аналізу

```
Вхід (фрагмент server.log):
192.168.1.1 - - [13/Jan/2026:10:30:15] "GET /index.html HTTP/1.1" 200 1524
192.168.1.2 - - [13/Jan/2026:10:30:16] "GET /about.html HTTP/1.1" 200 2048
192.168.1.1 - - [13/Jan/2026:10:30:17] "POST /login.php HTTP/1.1" 302 -
192.168.1.3 - - [13/Jan/2026:10:30:18] "GET /index.html HTTP/1.1" 200 1524
192.168.1.1 - - [13/Jan/2026:10:30:19] "GET /admin.php HTTP/1.1" 403 512

Вихід (report.html):
ЗВІТ ПРО РОБОТУ СЕРВЕРА
Період: 13.01.2026 10:30:15 - 13.01.2026 10:30:19
Загальна кількість запитів: 5
Найпопулярніша сторінка: /index.html (2 запити)
Розподіл кодів відповідей: 200 (60%), 302 (20%), 403 (20%)
Підозріла активність: IP 192.168.1.1 (3 запити за 4 секунди)
```

Завдання 3.2. Система управління інвентарем магазину

Створіть повноцінну систему управління інвентарем з такими модулями:

1. inventory.py - класи для товарів, категорій та управління запасами

2. file_manager.py - збереження/завантаження даних у JSON/CSV формати

3. reports.py - генерація звітів (популярні товари, майбутні замовлення, статистика)

4. Головна програма з меню для роботи з системою

Система повинна підтримувати:

- Додавання/видалення/редагування товарів
- Продажі та поповнення запасів
- Автоматичне замовлення товарів при низькому залишку
- Збереження всіх операцій у лог-файл
- Роботу з великими обсягами даних (1000+ товарів)

```
Приклад роботи:
```

```
    1. Додати товар: "Ноутбук Dell", категорія "Електроніка", ціна
25000, кількість 10
    2. Продати 2 ноутбуки
    3. Переглянути звіт по категорії "Електроніка"

    Вихід:
    Товар додано. ID: PROD001
    Продаж зареєстрована. Залишок: 8
    ЗВІТ ПО КАТЕГОРІЇ "ЕЛЕКТРОНІКА":
    Товарів: 15, Загальна вартість: 450000 грн
    Найпопулярніші: Ноутбук Dell (8 продажів)
```

Завдання 3.3. Платформа для аналізу даних студентських успішностей

Розробіть платформу для аналізу академічних даних з таких модулів:

1. data_importer.py - імпорт даних з різних форматів (CSV, JSON, Excel)
2. analyzer.py - статистичний аналіз (середні бали, кореляції, прогнозування)
3. visualizer.py - побудова графіків (гістограми, діаграми розподілу)
4. exporter.py - експорт результатів у різних форматах
5. scheduler.py - автоматичне виконання аналізу за розкладом

Функціонал:

- Обробка великих наборів даних (10000+ записів)
- Виявлення аномалій та трендів
- Прогнозування успішності на основі історичних даних
- Багатокористувацький доступ з різними ролями
- Автоматична генерація звітів у кінці семестру

```
    Приклад аналізу:
    Вхідні дані (студенти_семестр1.csv, студенти_семестр2.csv):
    ID,Ім'я,Предмет1,Предмет2,Предмет3
    001,Іван Петренко,85,90,78
    002,Марія Сидоренко,92,95,89

    Вихід (звіт.pdf):
    АНАЛІЗ АКАДЕМІЧНОЇ УСПІШНОСТІ
    Період: Вересень 2025 - Січень 2026
    Загальна успішність: 87.5%
    Найкращий студент: Марія Сидоренко (92.0%)
    Найскладніший предмет: Предмет3 (середній бал 83.5)
```

```
Прогноз на наступний семестр: +2.3% успішності
Рекомендації: Збільшити кількість практичних занять з Предмет3
```

6. Питання для самоперевірки

1. Яка різниця між командами import math та from math import sqrt? Наведіть приклади використання кожної з них.

2. Що таке модуль у Python? Як створити власний модуль і як його імпортувати в іншій програмі?

3. Опишіть різницю між методами read(), readline() та readlines() для роботи з файлами. Коли краще використовувати кожен з них?

4. Що таке контекстний менеджер with і навіщо його використовують при роботі з файлами? Наведіть приклад.

5. Які режими відкриття файлів існують у Python? Поясніть різницю між режимами ‘r’, ‘w’, ‘a’ та ‘x’.

6. Які винятки (exceptions) можуть виникати при роботі з файлами і як їх обробляти? Наведіть приклад обробки FileNotFoundError.

7. Як можна ефективно обробляти дуже великі файли, які не поміщаються в оперативну пам’ять?

8. Які функції надає модуль random? Наведіть приклади використання хоча б трьох з них.

9. Як працювати з датами та часом за допомогою модуля datetime? Наведіть приклад обчислення різниці між двома датами.

10. Що таке кодування файлів (encoding) і чому важливо вказувати правильне кодування (наприклад, utf-8) при роботі з текстовим файлами в Python?

11. Як зберегти структуровані дані (словники, списки) у файл і потім зчитати їх? Наведіть приклад використання JSON для цих цілей.

12. Як можна імпортувати модуль з псевдонімом (alias) і навіщо це робиться?

13. Що таке файл __init__.py у пакеті Python і яку роль він виконує?

14. Які способи запису даних у файл існують у Python? Поясніть різницю між методами write() та writelines().

15. Як можна перевірити, чи існує файл перед його відкриттям? Наведіть приклад з використанням модуля os або pathlib.

16. Що таке відносний та абсолютний шлях до файлу? Наведіть приклади.

17. Як можна організувати читання файлу рядок за рядком без завантаження всього файлу в пам’ять?

18. Які функції модуля math використовуються найчастіше? Наведіть приклади їх застосування в реальних задачах.

19. Як правильно обробляти ситуацію, коли при читанні файлу трапляються порожні рядки або рядки з некоректними даними?

20. Чим відрізняється робота з текстовими та бінарними файлами? Коли потрібно використовувати режим ‘b’ при відкритті файлу?

21. Які переваги та недоліки модульної архітектури програм порівняно з монолітною?

22. Як би ви організували систему логування для програми, яка працює з файлами, щоб відстежувати всі операції та помилки?

23. Чому важливо закривати файли після роботи з ними? Що може статися, якщо цього не робити?

24. Як би ви вирішили проблему одночасної роботи кількох користувачів з одним файлом?

25. Які критерії слід враховувати при виборі формату файлу (текстовий, CSV, JSON, XML) для збереження даних у вашому проекті?

26. Напишіть код, який створює власний модуль з функціями для роботи з рядками, а потім імпортує його у головній програмі трьома різними способами.

27. Створіть програму, яка читає файл зі списком електронних адрес, перевіряє їх коректність (наявність ‘@’ та ‘.’) та записує валідні адреси в окремий файл.

28. Напишіть функцію, яка приймає шлях до директорії та повертає статистику про файли у цій директорії (кількість, загальний розмір, розширення).

ЛАБОРАТОРНА РОБОТА №12
Структури даних та алгоритми: стек, черга, сортування та пошук

1. Мета

Оволодіти практичними навичками реалізації та використання основних лінійних структур даних (стек, черга) в Python, навчитися реалізовувати та аналізувати базові алгоритми сортування (бульбашкою, вибором, вставками) та пошуку (лінійний, бінарний), а також засвоїти принципи аналізу алгоритмічної складності для оцінки ефективності програмних рішень. Розвинути вміння застосовувати інструменти штучного інтелекту для аналізу та покращення власного коду.

2. Завдання

Завдання:

1. Реалізувати стек на основі списку Python та його основні операції (push, pop, peek).
2. Використовуючи власну реалізацію стеку, написати програму для перевірки збалансованості дужок у математичних виразах.
3. Реалізувати чергу на основі списку Python та її основні операції (enqueue, dequeue).
4. Написати програму симуляції простої черги обслуговування (наприклад, в банку або call-центрі).
5. Вивчити та використати стандартну структуру deque з модуля collections.
6. Реалізувати та порівняти алгоритми сортування бульбашкою, вибором та вставками.
7. За допомогою засібів профілювання часу виконання (timeit або time) порівняти практичну швидкість реалізованих алгоритмів сортування на різних наборах даних.
8. Реалізувати алгоритми лінійного та бінарного (ітеративний і рекурсивний варіанти) пошуку.
9. Проаналізувати часову складність (Big O notation) усіх реалізованих у роботі алгоритмів.
10. Використати доступний AI-інструмент (наприклад, GitHub Copilot, ChatGPT, Bard) для отримання рекомендацій щодо оптимізації написаного вами коду та проаналізувати отримані пропозиції.

3. Короткі теоретичні відомості

3.1. Стек (Stack)

Стек – це абстрактна структура даних, яка працює за принципом **LIFO (Last In, First Out)**: останній елемент, який додали в стек, буде першим вилученим. Її можна уявити як стопку тарілок.

Основні операції:

- push(item) – додає елемент item на вершину стеку.
- pop() – видаляє та повертає елемент з вершини стеку. Якщо стек порожній, виникає помилка.
- peek() (або top()) – повертає елемент з вершини стеку без його видалення.
- is_empty() – перевіряє, чи стек порожній.

Реалізація стеку на основі списку Python:

```
class Stack:
    def __init__(self):
        self.items = []    # список для зберігання елементів стеку

    def is_empty(self):
        return len(self.items) == 0

    def push(self, item):
        self.items.append(item)  # Додаємо в кінець списку

    def pop(self):
        if self.is_empty():
            raise IndexError("Pop from an empty stack")
        return self.items.pop()  # Видаляємо останній елемент

    def peek(self):
        if self.is_empty():
            raise IndexError("Peek from an empty stack")
        return self.items[-1]  # Переглядаємо останній елемент

    def size(self):
        return len(self.items)
```

Класичне застосування: перевірка збалансованості дужок. Алгоритм використовує стек для перевірки коректності розміщення круглих, квадратних та фігурних дужок у рядку. Відкриваюча дужка

додається в стек, а коли зустрічається закриваюча – перевіряється відповідність вершини стеку.

3.2. Черга (Queue)

Черга – це абстрактна структура даних, яка працює за принципом **FIFO (First In, First Out)**: перший елемент, який додали в чергу, буде першим вилученим. Її можна уявити як звичайну чергу людей.

Основні операції:

- enqueue(item) – додає елемент item в кінець черги.
- dequeue() – видаляє та повертає елемент з початку черги.
- is_empty() – перевіряє, чи черга порожня.

Реалізація черги на основі списку Python (неефективна для dequeue):

```
class SimpleQueue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def enqueue(self, item):
        self.items.append(item)  # Додаємо в кінець

    def dequeue(self):
        if self.is_empty():
            raise IndexError("Dequeue from an empty queue")
        return self.items.pop(0)  # Видаляємо перший елемент
(ПОВІЛЬНО!)
```

Операція pop(0) має лінійну складність O(n), оскільки потребує зсуву всіх інших елементів списку.

Ефективна черга: використання deque

Модуль collections надає клас deque (double-ended queue), оптимізований для швидких операцій з обох кінців.

```
from collections import deque

class EfficientQueue:
    def __init__(self):
        self.items = deque()  # Створюємо двосторонню чергу
```

```
    def enqueue(self, item):
        self.items.append(item)   # O(1)

    def dequeue(self):
        if not self.items:
            raise IndexError("Dequeue from an empty queue")
        return self.items.popleft()   # O(1) - ефективно!
```

3.3. Алгоритми сортування

Сортування – фундаментальна задача інформатики. Порівняємо три прості, але не найефективніші алгоритми (O(n²) у середньому випадку).

Алгоритм	**Принцип роботи**	**Часова складність (найгірший)**	**Переваги / Недоліки**
Бульбашкою (Bubble Sort)	Послідовно порівнює сусідні елементи та міняє їх місцями, якщо вони в неправильному порядку. Найбільші елементи «спливають» управо.	O(n²)	Простий для розуміння. Дуже повільний на великих даних. Майже не використовується на практиці.
Вибором (Selection Sort)	На кожному кроці знаходить мінімальний елемент з невідсортованої частини і ставить його на поточну позицію.	O(n²)	Простий. Кількість обмінів мінімальна (O(n)). Нестійкий.
Вставками (Insertion Sort)	Масив поділяється на відсортовану та невідсортовану частини. Елементи з невідсортованої частини послідовно вставляються на правильну позицію у відсортованій.	O(n²)	Ефективний на майже відсортованих даних (практично O(n)). Стабільний. Використовується для невеликих масивів.

Приклад. Сортування вставками:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):    # Починаємо з другого елемента
        key = arr[i]  # Елемент, який будемо вставляти
        j = i - 1
        # Зсуваємо елементи відсортованої частини, які більші за key, вправо
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key  # Вставляємо key на знайдену позицію
    return arr
```

3.4. Алгоритми пошуку

Алгоритм	Принцип роботи	Часова складність	Умова застосування
Лінійний (Linear Search)	Послідовне перебирання всіх елементів списку доти, доки не знайдете шуканий або не дійде до кінця.	O(n)	Універсальний. Працює на будь-яких, навіть невідсортованих, даних.
Бінарний (Binary Search)	Ділить відсортований масив навпіл, порівнює шуканий елемент з елементом у середині та відкидає непотрібну половину. Процес повторюється.	O(log n) **ВАЖЛИВО:** Працює **тільки** на відсортованих масивах!	Відсортовані дані

Приклад. Бінарний пошук (ітеративний варіант):

```
def binary_search_iterative(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid  # Знайдено, повертаємо індекс
```

```
        elif arr[mid] < target:
            low = mid + 1  # Шукаємо у правій половині
        else:
            high = mid - 1  # Шукаємо у лівій половині
    return -1  # Елемент не знайдено
```

3.5. Аналіз складності алгоритмів (Big O notation)

«О» велике (Big O) – це математичне позначення, яке описує асимптотичну поведінку функції, зокрема часової чи просторової складності алгоритму при зростанні обсягу вхідних даних (n). Воно дозволяє оцінити ефективність алгоритму, абстрагуючись від конкретної машини.

Найпоширеніші класи складності:

- **O(1)** – константна складність. Час виконання не залежить від n (напр., доступ до елемента масиву за індексом).
- **O(log n)** – логарифмічна. Дуже ефективно (напр., бінарний пошук).
- **O(n)** – лінійна. Час виконання прямо пропорційний n (напр., лінійний пошук).
- **O(n log n)** – лінійно-логарифмічна. Часто зустрічається в ефективних алгоритмах сортування (напр., швидке сортування, сортування злиттям).
- **O(n^2)** – квадратична. Час виконання зростає квадратично зі збільшенням n (напр., сортування бульбашкою, вибором, вставками для великих n).
- **O(2^n)** – експоненційна. Дуже швидко зростає, непрактично для великих n.

4. Методичні рекомендації

Нижче наведено розв’язки типових задач по темі лабораторної роботи. Рекомендується спочатку самостійно спробувати розв’язати задачі, а потім звірятися з запропонованими рішеннями.

Задача 1. Реалізація стеку та перевірка збалансованості дужок

Реалізуйте функції для роботи зі стеком на основі списку: push(stack, item), pop(stack), peek(stack), is_empty(stack). Використовуйте ці функції

для перевірки збалансованості дужок у рядку. Дужки можуть бути трьох видів: круглі (), квадратні [] та фігурні {}.

```
    def push(stack, item):
        """Додає елемент на вершину стеку."""
        stack.append(item)

    def pop(stack):
        """Видаляє та повертає елемент з вершини стеку."""
        if is_empty(stack):
            raise IndexError("Стек порожній")
        return stack.pop()

    def peek(stack):
        """Повертає верхній елемент стеку без видалення."""
        if is_empty(stack):
            raise IndexError("Стек порожній")
        return stack[-1]

    def is_empty(stack):
        """Перевіряє, чи стек порожній."""
        return len(stack) == 0

    def is_balanced(expression):
        """
        Перевіряє збалансованість дужок у виразі.
        Повертає True, якщо дужки збалансовані, інакше False.
        """
        stack = []
        matching_brackets = {')': '(', ']': '[', '}': '{'}

        for char in expression:
            if char in '([{':
                # Додаємо відкриваючу дужку в стек
                push(stack, char)
            elif char in ')]}':
                # Якщо стек порожній або дужка не відповідає
                if is_empty(stack) or peek(stack) !=
matching_brackets[char]:
                    return False
                pop(stack)
        # Вираз збалансований, якщо стек порожній
        return is_empty(stack)

    # Тестування функції is_balanced
```

```
if __name__ == "__main__":
    test_cases = [
        "(((())))",
        "{[()()]}",
        "({[()]})",
        "((())",
        "{[}]",
        ")("
    ]
    for expr in test_cases:
        result = is_balanced(expr)
        print(f"{expr}: {result}")
```

Приклад вхідних та вихідних даних:

```
Вхід: "(((())))" → Вихід: True
Вхід: "{[()()]}" → Вихід: True
Вхід: "((())" → Вихід: False
Вхід: "{[}]" → Вихід: False
Вхід: ")(()" → Вихід: False
```

Коментарі:

Для Рішення задачі використовується стек для відстеження відкриваючих дужок. При зустрічі закриваючої дужки перевіряється відповідність верхнього елемента стеку. Словник matching_brackets допомагає швидко знаходити відповідні пари дужок.

Задача 2. Реалізація черги та симуляція черги обслуговування

Створіть функції для роботи з чергою: enqueue(queue, item), dequeue(queue), is_queue_empty(queue). Напишіть програму для симуляції черги обслуговування клієнтів у банку. Кожен клієнт має номер та час обслуговування. Програма повинна обробляти клієнтів за принципом FIFO.

```
from collections import deque
import time
import random

def enqueue(queue, item):
    """Додає клієнта до черги."""
    queue.append(item)

def dequeue(queue):
```

```
        """Видаляє та повертає першого клієнта з черги."""
        if is_queue_empty(queue):
            raise IndexError("Черга порожня")
        return queue.popleft()

    def is_queue_empty(queue):
        """Перевіряє, чи черга порожня."""
        return len(queue) == 0

    def simulate_bank_queue(num_customers, max_service_time):
        """
        Симулює чергу обслуговування в банку.

        Args:
            num_customers: кількість клієнтів
            max_service_time:  максимальний  час  обслуговування
одного клієнта
        """
        queue = deque()
        total_wait_time = 0
        current_time = 0

        print("Початок роботи банку")
        print("-" * 40)

        # Створюємо клієнтів
        for i in range(num_customers):
            service_time = random.randint(1, max_service_time)
            arrival_time = current_time + random.randint(0, 3)
            current_time = arrival_time

            customer = {
                'id': i + 1,
                'arrival_time': arrival_time,
                'service_time': service_time
            }

            enqueue(queue, customer)
            print(f"Клієнт      {customer['id']}      прибув      о
{arrival_time},"
                  f"час обслуговування: {service_time}")

        print("-" * 40)
        print("Початок обслуговування")
```

```
        print("-" * 40)

        # Обслуговуємо клієнтів
        current_time = 0
        while not is_queue_empty(queue):
            customer = dequeue(queue)

            # Розрахунок часу очікування
            wait_time = max(0, current_time -
customer['arrival_time'])
            total_wait_time += wait_time

            # Оновлення поточного часу
            current_time = max(current_time, customer['arrival_time'])
            start_time = current_time
            end_time = current_time + customer['service_time']

            print(f"Клієнт {customer['id']}: "
                  f"очікування {wait_time}, "
                  f"обслуговування {start_time}-{end_time}")

            # Симуляція часу обслуговування
            current_time = end_time

        average_wait_time = total_wait_time / num_customers
        print("-" * 40)
        print(f"Загальний час роботи: {current_time}")
        print(f"Середній час очікування: {average_wait_time:.2f}")

    # Запуск симуляції
    if __name__ == "__main__":
        simulate_bank_queue(num_customers=5, max_service_time=5)
```

Приклад виконання:

```
Початок роботи банку
----------------------------------------
Клієнт 1 прибув о 0, час обслуговування: 3
Клієнт 2 прибув о 1, час обслуговування: 2
Клієнт 3 прибув о 3, час обслуговування: 4
Клієнт 4 прибув о 4, час обслуговування: 1
Клієнт 5 прибув о 6, час обслуговування: 3
----------------------------------------
Початок обслуговування
----------------------------------------
Клієнт 1: очікування 0, обслуговування 0-3
```

```
Клієнт 2: очікування 2, обслуговування 3-5
Клієнт 3: очікування 0, обслуговування 5-9
Клієнт 4: очікування 5, обслуговування 9-10
Клієнт 5: очікування 4, обслуговування 10-13
----------------------------------------
Загальний час роботи: 13
Середній час очікування: 2.20
```

Коментарі:

У цій задачі використовується deque з модуля collections для ефективної реалізації черги. Кожен клієнт представлений словником з інформацією про час прибуття та тривалість обслуговування. Алгоритм враховує час очікування кожного клієнта та розраховує середній час очікування.

Задача 3. Реалізація сортування бульбашкою

Реалізуйте алгоритм сортування бульбашкою. Функція повинна приймати список чисел та повертати відсортований список у порядку зростання. Додайте можливість підрахунку кількості порівнянь та обмінів.

```python
def bubble_sort(arr, verbose=False):
    """
    Сортування бульбашкою.

    Args:
        arr: список для сортування
        verbose: якщо True, виводить проміжні результати

    Returns:
        Відсортований список
    """
    n = len(arr)
    comparisons = 0
    swaps = 0

    # Робимо копію масиву, щоб не змінювати оригінал
    sorted_arr = arr.copy()
    for i in range(n - 1):
        # Флаг для перевірки, чи були обміни
        swapped = False

        for j in range(n - i - 1):
```

```
                comparisons += 1
                if sorted_arr[j] > sorted_arr[j + 1]:
                    # Обмін елементів
                    sorted_arr[j], sorted_arr[j + 1] = sorted_arr[j + 1], sorted_arr[j]
                    swaps += 1
                    swapped = True

                    if verbose:
                        print(f"Порівняння {comparisons}: обмін {sorted_arr[j+1]} ↔ {sorted_arr[j]}")

            # Якщо обмінів не було, масив відсортовано
            if not swapped:
                break

            if verbose:
                print(f"Після ітерації {i + 1}: {sorted_arr}")

        if verbose:
            print(f"Загальна кількість порівнянь: {comparisons}")
            print(f"Загальна кількість обмінів: {swaps}")

        return sorted_arr

    # Тестування алгоритму
    if __name__ == "__main__":
        test_arrays = [
            [64, 34, 25, 12, 22, 11, 90],
            [5, 1, 4, 2, 8],
            [1, 2, 3, 4, 5],  # вже відсортований
            [10, 9, 8, 7, 6]   # обернений порядок
        ]
        for i, arr in enumerate(test_arrays, 1):
            print(f"\nТест {i}:")
            print(f"Вхідний масив: {arr}")
            sorted_arr = bubble_sort(arr, verbose=False)
            print(f"Відсортований масив: {sorted_arr}")
```

Приклад виконання:

```
Вхід: [64, 34, 25, 12, 22, 11, 90] → Вихід: [11, 12, 22, 25, 34, 64, 90]
Вхід: [5, 1, 4, 2, 8] → Вихід: [1, 2, 4, 5, 8]
Вхід: [1, 2, 3, 4, 5] → Вихід: [1, 2, 3, 4, 5]
```

```
Вхід: [10, 9, 8, 7, 6] → Вихід: [6, 7, 8, 9, 10]
```

Коментарі:

Алгоритм порівнює сусідні елементи та міняє їх місцями, якщо вони в неправильному порядку. Флаг swapped дозволяє оптимізувати алгоритм: якщо на якійсь ітерації не було жодного обміну, масив вже відсортовано. Це покращує продуктивність на майже відсортованих масивах.

Задача 4. Реалізація сортування вибором

Реалізуйте алгоритм сортування вибором. Знаходьте мінімальний елемент у невідсортованій частині масиву та поміщайте його на поточну позицію.

```
def selection_sort(arr, verbose=False):
    """
    Сортування вибором.

    Args:
        arr: список для сортування
        verbose: якщо True, виводить проміжні результати

    Returns:
        Відсортований список
    """
    n = len(arr)
    sorted_arr = arr.copy()

    for i in range(n - 1):
        # Знаходимо індекс мінімального елемента в невідсортованій частині
        min_idx = i

        for j in range(i + 1, n):
            if sorted_arr[j] < sorted_arr[min_idx]:
                min_idx = j

        # Міняємо місцями знайдений мінімальний елемент з першим елементом
        if min_idx != i:
            sorted_arr[i], sorted_arr[min_idx] = sorted_arr[min_idx], sorted_arr[i]
```

```
                if verbose:
                    print(f"Ітерація {i + 1}: мінімум
{sorted_arr[i]} на позицію {i}")
                    print(f"Проміжний результат: {sorted_arr}")

        return sorted_arr

    # Тестування алгоритму
    if __name__ == "__main__":
        test_cases = [
            [29, 10, 14, 37, 13],
            [64, 25, 12, 22, 11],
            [5, 4, 3, 2, 1],
            [1, 2, 3, 4, 5]
        ]

        for i, arr in enumerate(test_cases, 1):
            print(f"\nТест {i}:")
            print(f"Вхідний масив: {arr}")
            result = selection_sort(arr, verbose=False)
            print(f"Відсортований масив: {result}")
```

Приклад виконання:

```
Вхід: [29, 10, 14, 37, 13] → Вихід: [10, 13, 14, 29, 37]
Вхід: [64, 25, 12, 22, 11] → Вихід: [11, 12, 22, 25, 64]
Вхід: [5, 4, 3, 2, 1] → Вихід: [1, 2, 3, 4, 5]
Вхід: [1, 2, 3, 4, 5] → Вихід: [1, 2, 3, 4, 5]
```

Коментарі:

Алгоритм на кожному кроці знаходить мінімальний елемент серед ще не відсортованих елементів і ставить його на правильну позицію. Кількість обмінів у цьому алгоритмі мінімальна (не більше n-1), що може бути перевагою, якщо операція обміну є дорогою.

Задача 5. Реалізація сортування вставками

Реалізуйте алгоритм сортування вставками. Алгоритм повинен працювати подібно до того, як люди складають карти у грі.

```
    def insertion_sort(arr, verbose=False):
        """
        Сортування вставками.
        Args:
```

```
        arr: список для сортування
        verbose: якщо True, виводить проміжні результати
    Returns:
        Відсортований список
    """
    sorted_arr = arr.copy()

    for i in range(1, len(sorted_arr)):
        key = sorted_arr[i]  # Поточний елемент для вставки
        j = i - 1

        if verbose:
            print(f"\nКрок {i}: вставляємо {key}")

        # Зсуваємо елементи, більші за key, вправо
        while j >= 0 and sorted_arr[j] > key:
            sorted_arr[j + 1] = sorted_arr[j]
            j -= 1

        # Вставляємо key на правильну позицію
        sorted_arr[j + 1] = key

        if verbose:
            print(f"Проміжний результат: {sorted_arr}")

    return sorted_arr

# Тестування алгоритму
if __name__ == "__main__":
    test_arrays = [
        [12, 11, 13, 5, 6],
        [31, 41, 59, 26, 41, 58],
        [5, 2, 4, 6, 1, 3],
        [1, 2, 3, 4, 5]
    ]
    for i, arr in enumerate(test_arrays, 1):
        print(f"\nТест {i}:")
        print(f"Вхідний масив: {arr}")
        result = insertion_sort(arr, verbose=False)
        print(f"Відсортований масив: {result}")
```

Приклад виконання:

```
Вхід: [12, 11, 13, 5, 6] → Вихід: [5, 6, 11, 12, 13]
Вхід: [31, 41, 59, 26, 41, 58] → Вихід: [26, 31, 41, 41, 58, 59]
```

```
Вхід: [5, 2, 4, 6, 1, 3] → Вихід: [1, 2, 3, 4, 5, 6]
Вхід: [1, 2, 3, 4, 5] → Вихід: [1, 2, 3, 4, 5]
```

Коментарі:

Цей алгоритм ефективний для малих або майже відсортованих масивів. Він стабільний (зберігає відносний порядок елементів з однаковими значеннями) та може сортувати масив "на місці" без використання додаткової пам'яті.

Задача 6. Порівняння швидкості алгоритмів сортування

Напишіть програму для порівняння часу виконання трьох алгоритмів сортування (бульбашкою, вибором, вставками) на різних наборах даних. Використовуйте модуль time для вимірювання часу.

```
import time
import random
from copy import copy

def generate_test_data(size, data_type='random'):
    """
    Генерує тестові дані різних типів.
    Args:
        size: розмір масиву
        data_type: тип даних ('random', 'sorted', 'reversed',
'almost_sorted')
    Returns:
        Тестовий масив
    """
    if data_type == 'random':
        return [random.randint(1, 1000) for _ in range(size)]
    elif data_type == 'sorted':
        return sorted([random.randint(1, 1000) for _ in
range(size)])
    elif data_type == 'reversed':
        return sorted([random.randint(1, 1000) for _ in
range(size)], reverse=True)
    elif data_type == 'almost_sorted':
        arr = sorted([random.randint(1, 1000) for _ in
range(size)])
        # Змінюємо декілька елементів
        for _ in range(max(1, size // 20)):
            i = random.randint(0, size - 1)
            j = random.randint(0, size - 1)
```

```
                arr[i], arr[j] = arr[j], arr[i]
            return arr
        else:
            raise ValueError(f"Невідомий тип даних: {data_type}")

    def measure_sorting_time(sort_func, arr):
        """
        Вимірює час виконання функції сортування.

        Args:
            sort_func: функція сортування
            arr: масив для сортування
        Returns:
            Час виконання в секундах
        """
        test_arr = copy(arr)  # Створюємо копію, щоб не змінювати
оригінал
        start_time = time.perf_counter()
        sort_func(test_arr)
        end_time = time.perf_counter()
        return end_time - start_time

    def compare_sorting_algorithms():
        """Порівнює швидкість різних алгоритмів сортування."""

        # Алгоритми для порівняння
        algorithms = {
            'Bubble Sort': bubble_sort,
            'Selection Sort': selection_sort,
            'Insertion Sort': insertion_sort
        }

        # Розміри масивів для тестування
        sizes = [100, 500, 1000]

        # Типи даних для тестування
        data_types = ['random', 'sorted', 'reversed',
'almost_sorted']

        print("ПОРІВНЯННЯ АЛГОРИТМІВ СОРТУВАННЯ")
        print("=" * 60)

        for data_type in data_types:
            print(f"\nТип даних: {data_type.upper()}")
```

```
            print("-" * 40)
            print(f"{'Розмір':<10}", end="")
            for algo_name in algorithms.keys():
                print(f"{algo_name:<20}", end="")
            print()

            for size in sizes:
                # Генеруємо тестові дані
                test_data = generate_test_data(size, data_type)

                print(f"{size:<10}", end="")

                for algo_name, sort_func in algorithms.items():
                    try:
                        # Вимірюємо час
                        execution_time =
measure_sorting_time(sort_func, test_data)
                        print(f"{execution_time:<20.6f}", end="")
                    except Exception as e:
                        print(f"{'ERROR':<20}", end="")
                print()

    # Імпорт функцій сортування (вони вже визначені вище)
    if __name__ == "__main__":
        compare_sorting_algorithms()
```

Приклад виконання:

```
ПОРІВНЯННЯ АЛГОРИТМІВ СОРТУВАННЯ
============================================================

Тип даних: RANDOM
----------------------------------------
Розмір    Bubble Sort          Selection Sort       Insertion
Sort
100       0.001234             0.000567             0.000345
500       0.012345             0.005678             0.003456
1000      0.045678             0.023456             0.015678

Тип даних: SORTED
----------------------------------------
Розмір    Bubble Sort          Selection Sort       Insertion
Sort
100       0.000123             0.000456             0.000078
500       0.000345             0.002345             0.000234
1000      0.000678             0.009123             0.000567
```

Коментарі:

Алгоритми демонструють різну продуктивність на різних типах даних. Сортування вставками найефективніше на майже відсортованих масивах. Важливо пам'ятати, що всі три алгоритми мають квадратичну складність $O(n^2)$ у найгіршому випадку, тому для великих масивів краще використовувати більш ефективні алгоритми (швидке сортування, сортування злиттям).

Задача 7. Реалізація лінійного та бінарного пошуку

Реалізуйте алгоритми лінійного та бінарного пошуку (ітеративний та рекурсивний варіанти). Порівняйте їхню ефективність на різних розмірах масивів.

```
def linear_search(arr, target):
    """
    Лінійний пошук.
    Args:
        arr: список для пошуку
        target: шуканий елемент
    Returns:
        Індекс елемента або -1, якщо не знайдено
    """
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1
def binary_search_iterative(arr, target):
    """
    Бінарний пошук (ітеративний варіант).
    Args:
        arr: ВІДСОРТОВАНИЙ список для пошуку
        target: шуканий елемент
    Returns:
        Індекс елемента або -1, якщо не знайдено
    """
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
```

```
            elif arr[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
        return -1

    def binary_search_recursive(arr, target, left=0, right=None):
        """
        Бінарний пошук (рекурсивний варіант).
        Args:
            arr: ВІДСОРТОВАНИЙ список для пошуку
            target: шуканий елемент
            left: ліва межа пошуку
            right: права межа пошуку
        Returns:
            Індекс елемента або -1, якщо не знайдено
        """
        if right is None:
            right = len(arr) - 1

        # Базовий випадок: межі перетнулися
        if left > right:
            return -1
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            # Пошук у правій половині
            return binary_search_recursive(arr, target, mid + 1,
right)
        else:
            # Пошук у лівій половині
            return binary_search_recursive(arr, target, left, mid - 1)
    def compare_search_algorithms():
        """Порівнює ефективність алгоритмів пошуку."""

        import time
        import random

        print("ПОРІВНЯННЯ АЛГОРИТМІВ ПОШУКУ")
        print("=" * 60)

        sizes = [1000, 10000, 100000]
```

```
        for size in sizes:
            print(f"\nРозмір масиву: {size}")
            print("-" * 40)

            # Створюємо відсортований масив
            sorted_arr = sorted([random.randint(1, size * 10) for
_ in range(size)])

            # Вибираємо цільові елементи для пошуку
            targets = [
                sorted_arr[0],  # перший елемент
                sorted_arr[size // 2],  # середній елемент
                sorted_arr[-1],  # останній елемент
                size * 20  # елемент, якого немає в масиві
            ]

            algorithms = [
                ("Лінійний пошук", linear_search),
                ("Бінарний (ітеративний)", binary_search_iterative),
                ("Бінарний                              (рекурсивний)",
binary_search_recursive)
            ]

            print(f"{'Елемент':<15}", end="")
            for algo_name, _ in algorithms:
                print(f"{algo_name:<25}", end="")
            print()

            for target in targets:
                print(f"{target:<15}", end="")

                for algo_name, search_func in algorithms:
                    # Вимірюємо час пошуку
                    start_time = time.perf_counter()

                    if algo_name == "Лінійний пошук":
                        #    Для    лінійного    пошуку    створюємо
невідсортовану копію
                        unsorted_arr = sorted_arr.copy()
                        random.shuffle(unsorted_arr)
                        result = search_func(unsorted_arr, target)
                    else:
```

```
                    # Для бінарного пошуку використовуємо
відсортований масив
                        result = search_func(sorted_arr, target)
                    end_time = time.perf_counter()
                    execution_time = end_time - start_time
                    print(f"{execution_time:<25.8f}", end="")
                print()

    if __name__ == "__main__":
        # Швидкий тест коректності роботи алгоритмів
        test_arr = [1, 3, 5, 7, 9, 11, 13, 15]

        print("Тест коректності:")
        print(f"Масив: {test_arr}")

        test_cases = [7, 1, 15, 10]
        for target in test_cases:
            print(f"\nПошук {target}:")
            print(f"         Лінійний:     {linear_search(test_arr,
target)}")
            print(f"                 Бінарний         (ітеративний):
{binary_search_iterative(test_arr, target)}")
            print(f"                 Бінарний         (рекурсивний):
{binary_search_recursive(test_arr, target)}")

        # Порівняння продуктивності
        compare_search_algorithms()
```

Приклад виконання:

```
Тест коректності:
Масив: [1, 3, 5, 7, 9, 11, 13, 15]

Пошук 7:
  Лінійний: 3
  Бінарний (ітеративний): 3
  Бінарний (рекурсивний): 3

Пошук 10:
  Лінійний: -1
  Бінарний (ітеративний): -1
  Бінарний (рекурсивний): -1
```

Коментарі:

Бінарний пошук значно ефективніший за лінійний (O(log n) проти O(n)), але вимагає попереднього сортування масиву. Рекурсивна реалізація бінарного пошуку може бути менш ефективною через накладні витрати на виклики функцій та ризик переповнення стеку при великих масивах.

Типові помилки і шляхи їх усунення

1. **Неправильна робота з порожніми структурами даних**
   Проблема. IndexError при виклику pop() або peek() на порожньому стеку/черзі
   Рішення. Завжди перевіряйте, чи структура не порожня, перед виконанням операцій видалення або перегляду

```
def pop(stack):
    if is_empty(stack):
        raise IndexError("Стек порожній")
    return stack.pop()
```

2. **Неефективна реалізація черги на списках**
   Проблема. Повільна робота з великими чергами через використання pop(0)
   Рішення. Використовуйте deque з модуля collections або реалізуйте кільцеву чергу

```
from collections import deque
queue = deque()
queue.append('item')  # enqueue
item = queue.popleft()  # dequeue
```

3. **Забули про необхідність відсортованого масиву для бінарного пошуку**
   Проблема. Бінарний пошук повертає некоректні результати або зациклюється
   Рішення. Завжди переконуйтеся, що масив відсортований перед викликом бінарного пошуку

```
def binary_search(arr, target):
    # Можна додати перевірку:
    if arr != sorted(arr):
        raise ValueError("Масив має бути відсортованим")
    # ... решта коду
```

4. **Некоректне оновлення меж у бінарному пошуку**
   Проблема. Алгоритм пропускає елемент або зациклюється
   Рішення. Уважно перевіряйте логіку оновлення left та right

```
        if arr[mid] < target:
            left = mid + 1  # важливо: mid + 1, не mid
        else:
            right = mid - 1  # важливо: mid - 1, не mid
```

5. **Модифікація оригінального масиву при сортуванні**

   Проблема. Вихідний масив змінюється після сортування

   Рішення. Завжди створюйте копію масиву перед сортуванням

```
def bubble_sort(arr):
    sorted_arr = arr.copy()  # створюємо копію
    # ... сортуємо sorted_arr
    return sorted_arr
```

6. **Неправильне обчислення середнього значення у бінарному пошуку**

   Проблема. Можливе переповнення при великих значеннях

   Рішення. Використовуйте безпечну формулу

```
# НЕ БЕЗПЕЧНО для великих чисел:
mid = (left + right) // 2
# БЕЗПЕЧНО:
mid = left + (right - left) // 2
```

Корисні поради

1. **Перед написанням коду** завжди зрозумійте алгоритм на рівні псевдокоду або блок-схеми. Намалюйте алгоритм на папері для складних завдань.
2. **Використовуйте модуль** timeit для точних вимірювань часу виконання, особливо при порівнянні алгоритмів. Він дає більш точні результати, ніж time.time().
3. **Додавайте коментарі ДО написання коду** - це допомагає структурувати думки. Пишіть коментарі, які пояснюють "чому", а не "що".
4. **Тестуйте на крайніх випадках** (boundary cases):
   - порожні масиви/списки
   - масиви з одним елементом
   - вже відсортовані масиви
   - масиви в зворотньому порядку
   - масиви з однаковими елементами
5. **Використовуйте AI-інструменти для навчання, а не для копіювання**:
   - попросіть пояснити складну концепцію

- запропонуйте альтернативні підходи
- знайдіть помилки в вашому коді
- попросіть покращити читабельність коду

6. **Для рекурсивних алгоритмів** завжди переконайтеся, що є базовий випадок, який зупиняє рекурсію, і що кожен рекурсивний виклик наближає до базового випадку.

7. **При порівнянні алгоритмів** звертайте увагу не тільки на час виконання, але й на:

- об’єм використаної пам’яті
- складність у найгіршому, середньому та кращому випадках
- стабільність (чи зберігається порядок однакових елементів)
- можливість сортування "на місці"

8. **Використовуйте візуалізацію** для розуміння роботи алгоритмів. Існують онлайн-інструменти, які анімують роботу алгоритмів сортування та пошуку.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Реверс рядка за допомогою стеку

Напишіть функцію reverse_string(text), яка використовує стек для реверсу рядка. Створіть власну реалізацію стеку (не використовуйте вбудовані функції reversed() або зрізи [::-1]).

```
Вхід: "hello" → Вихід: "olleh"
Вхід: "Python" → Вихід: "nohtyP"
Вхід: "12345" → Вихід: "54321"
Вхід: "" → Вихід: ""
```

Завдання 1.2. Перевірка паліндрому з використанням стеку

Використовуючи стек, напишіть функцію is_palindrome(text), яка перевіряє, чи є рядок паліндромом (читається однаково зліва направо та справа наліво). Ігноруйте регістр букв та пробіли.

```
Вхід: "А роза упала на лапу Азора" → Вихід: True
Вхід: "hello" → Вихід: False
Вхід: "Racecar" → Вихід: True
Вхід: "Python" → Вихід: False
```

Завдання 1.3. Симуляція простої черги заявок

Створіть програму для обробки заявок. Функція process_requests(requests) приймає список заявок (рядки), додає їх у чергу та обробляє по одній, повертаючи список оброблених заявок у порядку їх надходження.

```
Вхід: ["заявка1", "заявка2", "заявка3"] → Вихід: ["заявка1",
"заявка2", "заявка3"]
Вхід: [] → Вихід: []
Вхід: ["task_A", "task_B"] → Вихід: ["task_A", "task_B"]
```

Завдання 1.4. Підрахунок кількості елементів у стеку

Напишіть функцію stack_size(stack), яка обчислює кількість елементів у стеку, не використовуючи вбудовану функцію len() та не змінюючи стек. Можна використовувати додатковий стек.

```
Вхід: [1, 2, 3, 4, 5] → Вихід: 5
Вхід: [] → Вихід: 0
Вхід: ["a", "b", "c"] → Вихід: 3
```

Завдання 1.5. Черга з пріоритетом (спрощена)

Створіть функцію simple_priority_queue(tasks), яка обробляє завдання з пріоритетом. Кожне завдання - це кортеж (пріоритет, назва), де пріоритет - число від 1 (найвищий) до 5 (найнижчий). Функція має повертати завдання в порядку зростання пріоритету (спочатку пріоритет 1, потім 2 і т.д.).

```
Вхід: [(3, "taskC"), (1, "taskA"), (2, "taskB")] → Вихід:
["taskA", "taskB", "taskC"]
Вхід: [(5, "low"), (1, "high"), (1, "urgent")] → Вихід:
["high", "urgent", "low"]
Вхід: [] → Вихід: []
```

Завдання 1.6. Сортування списку парних чисел

Напишіть функцію sort_even_numbers(numbers), яка приймає список цілих чисел і повертає відсортований список тільки парних чисел, непарні числа ігнуються. Використовуйте алгоритм сортування вибором.

```
Вхід: [5, 2, 8, 1, 9, 4] → Вихід: [2, 4, 8]
Вхід: [3, 7, 1] → Вихід: []
Вхід: [10, 2, 6, 4] → Вихід: [2, 4, 6, 10]
```

Завдання 1.7. Лінійний пошук з лічильником

Створіть функцію linear_search_count(arr, target), яка шукає елемент у масиві та повертає кортеж (індекс, кількість_порівнянь). Якщо елемент не знайдено, повертає (-1, кількість_порівнянь).

```
Вхід: arr=[10, 20, 30, 40, 50], target=30 → Вихід: (2, 3)
Вхід: arr=[5, 15, 25], target=10 → Вихід: (-1, 3)
Вхід: arr=[], target=5 → Вихід: (-1, 0)
```

Завдання 1.8. Бінарний пошук з відстеженням кроків

Напишіть функцію binary_search_steps(arr, target), яка виконує бінарний пошук у відсортованому масиві та повертає список індексів, які перевірялися під час пошуку, та фінальний результат.

```
Вхід: arr=[1, 3, 5, 7, 9, 11], target=7 → Вихід: ([2, 4, 3],
3)
Вхід: arr=[2, 4, 6, 8, 10], target=5 → Вихід: ([2, 1], -1)
Вхід: arr=[10, 20, 30], target=10 → Вихід: ([1, 0], 0)
```

Завдання 1.9. Сортування бульбашкою з оптимізацією

Реалізуйте оптимізовану версію сортування бульбашкою, яка враховує, що після k ітерацій останні k елементів вже на своїх місцях. Функція повинна повертати кількість виконаних ітерацій.

```
Вхід: [5, 3, 8, 1, 2] → Вихід: ([1, 2, 3, 5, 8], 4)
Вхід: [1, 2, 3, 4, 5] → Вихід: ([1, 2, 3, 4, 5], 1)
Вхід: [4, 3, 2, 1] → Вихід: ([1, 2, 3, 4], 3)
```

Завдання 1.10. Пошук мінімального та максимального елемента

Напишіть функцію find_min_max(arr), яка знаходить мінімальний та максимальний елемент у масиві, використовуючи принцип алгоритму сортування вибором, але без повного сортування. Функція має повертати кортеж (min, max).

```
Вхід: [7, 2, 9, 1, 5] → Вихід: (1, 9)
Вхід: [-5, -10, -3, -1] → Вихід: (-10, -1)
Вхід: [42] → Вихід: (42, 42)
```

Частина 2. Середні завдання

Завдання 2.1. Калькулятор зворотньої польської нотації

Реалізуйте калькулятор, який обчислює вирази у зворотній польській нотації (RPN) з використанням стеку. Підтримуйте операції: +, -, *, /. Функція rpn_calculator(expression) приймає рядок з виразом у RPN (числа та оператори розділені пробілами).

```
Вхід: "3 4 +" → Вихід: 7
Вхід: "5 1 2 + 4 * + 3 -" → Вихід: 14
Вхід: "10 6 9 3 + -11 * / * 17 + 5 +" → Вихід: 22
```

Завдання 2.2. Симуляція черги з обмеженим часом очікування

Створіть симуляцію черги, де кожен клієнт має час прибуття та максимальний час очікування. Якщо час очікування перевищує максимальний, клієнт залишає чергу. Функція queue_with_timeout(customers, service_time) приймає список клієнтів у форматі (час_прибуття, макс_очікування) та час обслуговування одного клієнта.

```
Вхід: customers=[(0, 5), (2, 3), (4, 2)], service_time=2 →
Вихід: (обслужено: 2, втрачено: 1)
```

```
Вхід: customers=[(0, 10), (1, 10), (2, 10)], service_time=3 →
Вихід: (обслужено: 3, втрачено: 0)
Вхід: customers=[(0, 1), (0, 5), (1, 1)], service_time=2 →
Вихід: (обслужено: 1, втрачено: 2)
```

Завдання 2.3. Сортування за кількома критеріями

Напишіть функцію multi_criteria_sort(students), яка сортує список студентів. Кожен студент - словник з ключами name, grade, attendance. Сортування: спочатку за оцінкою (спадання), потім за відвідуваністю (спадання), потім за іменем (алфавіт).

```
Вхід:
   [{"name": "Alice", "grade": 85, "attendance": 90},
   {"name": "Bob", "grade": 85, "attendance": 95},
   {"name": "Charlie", "grade": 90, "attendance": 80}]
Вихід:
   [{"name": "Charlie", "grade": 90, "attendance": 80},
   {"name": "Bob", "grade": 85, "attendance": 95},
   {"name": "Alice", "grade": 85, "attendance": 90}]
Вхід:
   [{"name": "Anna", "grade": 75, "attendance": 100},
   {"name": "David", "grade": 80, "attendance": 85}]
Вихід:
   [{"name": "David", "grade": 80, "attendance": 85},
   {"name": "Anna", "grade": 75, "attendance": 100}]
```

Завдання 2.4. Гібридний алгоритм сортування

Створіть функцію hybrid_sort(arr, threshold=10), яка використовує сортування вставками для малих підмасивів (розміром <= threshold) та сортування вибором для більших. Розділіть масив на підмасиви розміром threshold та відсортуйте кожен окремо.

```
Вхід: [9, 7, 5, 11, 12, 2, 14, 3, 10, 6], threshold=4 → Вихід:
[2, 3, 5, 6, 7, 9, 10, 11, 12, 14]
Вхід: [3, 1, 4, 1, 5, 9, 2, 6], threshold=3 → Вихід: [1, 1, 2,
3, 4, 5, 6, 9]
Вхід: [10, 9, 8, 7], threshold=5 → Вихід: [7, 8, 9, 10]
```

Завдання 2.5. Аналізатор складності алгоритмів

Напишіть програму, яка аналізує емпіричну складність алгоритмів. Функція complexity_analyzer(sort_func, sizes) приймає функцію сортування та список розмірів масивів, генерує випадкові масиви

кожного розміру, вимірює час сортування та будує графік залежності часу від розміру.

```
Аналіз складності сортування бульбашкою:
Розмір 100: 0.0012 сек
Розмір 200: 0.0048 сек
Розмір 400: 0.0192 сек
Приблизна складність: O(n²)
```

Частина 3. Складні завдання

Завдання 3.1. Система обробки завдань з пріоритетами та залежностями

Розробіть систему обробки завдань, де кожне завдання має: ID, пріоритет (1-10), тривалість виконання та список ID завдань, які повинні бути виконані перед ним. Система має визначати оптимальний порядок виконання завдань, враховуючи пріоритети та залежності. Використовуйте комбінацію черг та стеків.

```
Формат вхідних даних:
tasks = [
    {"id": "A", "priority": 3, "duration": 2, "dependencies": []},
    {"id": "B", "priority": 1, "duration": 1, "dependencies": ["A"]},
    {"id": "C", "priority": 2, "duration": 3, "dependencies": ["A"]},
    {"id": "D", "priority": 4, "duration": 2, "dependencies": ["B", "C"]}
]
Приклад виходу:
Порядок виконання: A → B → C → D
Загальний час: 8
Час очікування: 4
```

Завдання 3.2. Оптимізація маршруту доставки з використанням структур даних

Створіть програму для оптимізації маршруту кур’єра. Кожна доставка має: адресу, час прибуття, час обслуговування та пріоритет. Кур’єр має обслуговувати доставки в оптимальному порядку, враховуючи:

1. Пріоритетні доставки обслуговуються першими
2. Враховується відстань між адресами (спрощено - час переїзду)
3. Не пропускати "вікна" обслуговування

```
Вхідні дані:
```

```
    deliveries = [
        {"id": 1, "address": "A", "arrival": 0, "service": 10,
"priority": 2},
        {"id": 2, "address": "B", "arrival": 5, "service": 15,
"priority": 1},
        {"id": 3, "address": "C", "arrival": 10, "service": 5,
"priority": 3},
    ]
    # Матриця часу переїзду між адресами
    travel_time = {
        ("A", "B"): 3, ("B", "A"): 3,
        ("A", "C"): 5, ("C", "A"): 5,
        ("B", "C"): 4, ("C", "B"): 4
    }
    Очікуваний вихід: Оптимальний порядок обслуговування та
    загальний час.
```

Завдання 3.3. Оптимізація алгоритмів за допомогою AI

Виберіть один з ваших алгоритмів (наприклад, сортування або пошук) та проведіть повний цикл оптимізації з використанням AI:

1. **Аналіз поточного коду:** завантажте свій код у AI-інструмент (ChatGPT, Copilot тощо)
2. **Діагностика:** запропонуйте AI проаналізувати ваш код на наявність неефективностей
3. **Оптимізація:** запросіть 3 альтернативні реалізації з поясненнями
4. **Тестування:** протестуйте всі варіанти на однакових даних
5. **Аналіз результатів:** порівняйте швидкість, читабельність, використання пам’яті
6. **Звіт:** створіть звіт з висновками та рекомендаціями

Вимоги до звіту:

1. Оригінальний код та його аналіз AI
2. Три оптимізовані версії з поясненнями
3. Результати тестування (таблиця порівняння)
4. Висновки: яку версію рекомендувати та чому
5. Рефлексія: що ви навчилися про оптимізацію коду

Приклад фрагменту звіту:

```
Оригінальний алгоритм: Сортування бульбашкою
Час виконання (n=1000): 0.045 сек
```

```
Версія 1: З оптимізацією прапорця
Час: 0.038 сек, покращення: 15%

Версія 2: З обмеженням діапазону
Час: 0.042 сек, покращення: 7%

Версія 3: Гібридна (бульбашка + вставки)
Час: 0.025 сек, покращення: 44%

Рекомендація: Версія 3 як найефективніша
```

Загальні вказівки до виконання завдань:

1. Для кожної задачі створюйте окремий файл .py
2. Код має бути добре документований (коментарі, docstrings)
3. Додавайте тести для перевірки коректності
4. Для задач з AI додайте скріншоти або логі взаємодії
5. Складнощі у виконанні фіксуйте у коментарях - це допоможе в оцінюванні
6. Уникайте плагіату - код має бути написаний самостійно на основі розуміння алгоритмів

Критерії оцінювання:

- Коректність роботи програми (40%)
- Оптимізація та ефективність коду (25%)
- Читабельність та структурування (20%)
- Якість коментарів та документації (15%)

6. Питання для самоперевірки

1. Який принцип роботи стеку (LIFO/FIFO) і наведіть два реальних приклади його застосування?
2. Чому операція pop(0) у звичайному списку Python має лінійну складність O(n), а popleft() у deque - константну O(1)?
3. Поясніть, як алгоритм перевірки збалансованості дужок використовує стек. Що станеться, якщо після обробки всього рядка стек не порожній?
4. Чому бінарний пошук вимагає відсортованого масиву? Що станеться, якщо спробувати виконати його на невідсортованих даних?
5. Порівняйте часову складність лінійного та бінарного пошуку у найкращому, середньому та найгіршому випадках.

6. Чому сортування бульбашкою на вже відсортованому масиві може працювати швидше, ніж у загальному випадку?

7. Яка основна відмінність між сортуванням вибором та сортуванням вставками щодо кількості обмінів елементів?

8. Що таке "стабільність" алгоритму сортування? Чи є сортування бульбашкою, вибором та вставками стабільними?

9. Як працює зворотня польська нотація (RPN) і чому для її обчислення зручно використовувати стек?

10. Що означає "емпірична складність алгоритму" та як її можна виміряти на практиці?

11. Наведіть псевдокод алгоритму перевірки збалансованості дужок трьох видів: (), [], {}.

12. Як би ви реалізували чергу з двома стеками? Оцініть складність операцій enqueue та dequeue.

13. Напишіть рекурсивну версію алгоритму лінійного пошуку. Чи має вона переваги над ітеративною?

14. Як можна модифікувати алгоритм бінарного пошуку, щоб він знаходив не перше, а останнє входження елемента?

15. Поясніть, чому у циклі бінарного пошуку використовується mid = left + (right - left) // 2 замість mid = (left + right) // 2?

16. У яких випадках сортування вставками буде ефективнішим за сортування вибором, і навпаки? Наведіть конкретні приклади даних.

17. Як би ви вибрали між стеком та чергою для наступних завдань:

a) Система "відміни" в текстових редакторах

b) Обробка запитів до сервера

c) Обробка повідомлень в чаті

18. Проаналізуйте, чому всі три реалізовані алгоритми сортування мають квадратичну складність $O(n^2)$ у найгіршому випадку. Що є спільного в їхніх алгоритмах?

19. Як можна використати AI-інструменти не для копіювання коду, а для ефективного навчання програмуванню? Наведіть три конкретні стратегії.

20. Як би ви пояснили різницю між часовою та просторовою складністю алгоритму новачку в програмуванні? Наведіть аналогії з реального життя.

21. Чи можна використати бінарний пошук для пошуку в зв’язному списку? Чому так чи ні?

22. Як би ви модифікували алгоритм сортування бульбашкою, щоб він сортував масив у порядку спадання?

23. Що станеться з алгоритмом перевірки дужок, якщо у виразі зустрінеться символ, який не є дужкою? Чи потрібно його обробляти?

24. Чи можна реалізувати стек за допомогою двох черг? Якою буде складність операцій?

25. Як би ви виміряли ефективність оптимізації, запропонованої AI? Які метрики, крім часу виконання, варто враховувати?

26. Чому, незважаючи на однакову теоретичну складність $O(n^2)$, різні алгоритми сортування показують різну практичну швидкість на однакових даних?

27. У яких реальних сценаріях прості алгоритми сортування (бульбашкою, вибором, вставками) можуть бути кращим вибором, ніж складніші алгоритми з $O(n \log n)$?

28. Які обмеження має рекурсивна реалізація бінарного пошуку порівняно з ітеративною? У яких випадках вона може призвести до помилок?

29. Чи завжди використання AI для оптимізації коду призводить до покращення? Наведіть потенційні ризики та проблеми.

30. Як знання структур даних та алгоритмів може допомогти у вирішенні повсякденних, не пов’язаних з програмуванням завдань? Наведіть приклад.

ЛАБОРАТОРНА РОБОТА №13
Робота з колекціями Python та comprehensions

1. Мета

Оволодіти практичними навичками ефективної роботи з вбудованими колекціями та спеціальними конструкціями мови Python для створення лаконічного, читабельного та продуктивного коду. Закріпити розуміння та навчитися застосовувати comprehensions (генератори списків, словників, множин), ітератори, генератори (за допомогою yield) та корисні типи з модуля collections (namedtuple, defaultdict, Counter) для розв'язання типових задач обробки даних.

2. Завдання

1. Вивчити синтаксис та принципи роботи comprehensions, генераторних виразів, генераторів з yield та обраних типів з модуля collections.
2. Навчитися замінювати традиційні цикли for та конструкції зі збором даних на лаконічні comprehensions там, де це покращує читабельність коду.
3. Опанувати створення та використання ітераторів і генераторів для ефективної роботи з послідовностями даних, особливо великими.
4. Набути досвіду використання namedtuple для створення зручних іменованих кортежів, defaultdict для спрощення угруповання даних та Counter для ефективного підрахунку елементів.
5. Розв'язати набір практичних задач різного рівня складності, спрямованих на консолідацію отриманих знань.

3. Короткі теоретичні відомості

У Python колекції – це структури даних для зберігання та організації груп елементів. Окрім базових типів (list, tuple, dict, set), існують потужні синтаксичні конструкції та спеціалізовані типи, що значно спрощують та прискорюють роботу з даними.

3.1. Comprehensions (генератори колекцій) – це компактний спосіб створення нових колекцій (списків, словників, множин) на основі ітерабельних об'єктів з можливістю фільтрації та трансформації елементів.

- **List Comprehension (генератор списку)**

Створює новий список. Синтаксис: [expression for item in iterable if condition].

```
# Замість:
squares = []
for x in range(5):
    squares.append(x**2)
# Використовуємо:
squares = [x**2 for x in range(5)]  # [0, 1, 4, 9, 16]

# З умовою фільтрації:
even_squares = [x**2 for x in range(10) if x % 2 == 0]  # [0, 4, 16, 36, 64]
```

- **Dictionary Comprehension (генератор словника)**

Створює новий словник. Синтаксис: {key_expression: value_expression for item in iterable if condition}.

```
# Створення словника числа: його куб
cubes_dict = {x: x**3 for x in range(1, 6)}  # {1: 1, 2: 8, 3: 27, 4: 64, 5: 125}

# Інвертування словника (ключі стають значеннями і навпаки)
original = {'a': 1, 'b': 2}
inverted = {v: k for k, v in original.items()}  # {1: 'a', 2: 'b'}
```

- **Set Comprehension (генератор множини)**

Створює нову множину (унікальні елементи). Синтаксис: {expression for item in iterable if condition}.

```
# Множина унікальних довжин слів
words = ["apple", "banana", "cat", "dog", "elephant"]
lengths = {len(word) for word in words}  # {3, 5, 6, 8} (порядок може відрізнятися)
```

3.2. Генераторні вирази (Generator Expressions)

Схожі на list comprehensions, але використовують круглі дужки () і повертають **генератор** – об'єкт, який видає елементи по одному "ліниво" (lazy evaluation), не завантажуючи весь результат у пам'ять одразу. Це ефективно для великих обсягів даних.

```
# List comprehension (завантажує весь список)
list_comp = [x*x for x in range(1000000)]  # Займає багато пам'яті
```

```
    # Генераторний вираз (працює "на льоту")
    gen_exp = (x*x for x in range(1000000))    # Займає мінімум пам'яті
    print(next(gen_exp))  # 0
    print(next(gen_exp))  # 1
    # Можна ітерувати:
    for val in gen_exp:
        if val > 100:
            break
        print(val)
```

3.3. Генератори з yield

Функція-генератор – це функція, яка містить ключове слово yield. При виклику вона повертає об'єкт-генератор і призупиняє виконання до наступного виклику next(). Це ще потужніший інструмент для створення складних послідовностей.

```
    def countdown(n):
        while n > 0:
            yield n    # Повертає значення та "заморожує" стан функції
            n -= 1

    cd = countdown(3)
    print(next(cd))  # 3
    print(next(cd))  # 2
    print(next(cd))  # 1
    # Після цього виклик next(cd) викине StopIteration
```

3.4. Модуль collections – корисні контейнери:

• namedtuple: Створює підклас кортежу з іменованими полями, роблячи код самодокументованим.

```
    from collections import namedtuple
    Point = namedtuple('Point', ['x', 'y'])
    p = Point(10, y=20)
    print(p.x, p.y)  # 10 20 (читабельніше, ніж p[0], p[1])
    print(p._asdict())  # Конвертує в словник: {'x': 10, 'y': 20}
```

• defaultdict: Підклас словника, який автоматично створює значення за замовчуванням для неіснуючих ключів за допомогою фабричної функції.

```
    from collections import defaultdict
    # Угруповання слів за довжиною
```

```
    words_by_length = defaultdict(list)  # list - це функція, що створює порожній список
    for word in ["apple", "bat", "cat", "elephant"]:
        words_by_length[len(word)].append(word)  # Не потрібно перевіряти, чи існує ключ
    # Результат: defaultdict(<class 'list'>, {5: ['apple'], 3: ['bat', 'cat'], 8: ['elephant']})
```

- Counter: Спеціалізований словник для підрахунку хешованих об'єктів.

```
    from collections import Counter
    text = "abracadabra"
    letter_counts = Counter(text)
    print(letter_counts)  # Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})
    print(letter_counts.most_common(2))  # [('a', 5), ('b', 2)]
```

3.5. Ітератори

Будь-який об'єкт, з якого можна отримувати елементи по черзі, реалізує протокол ітератора (методи __iter__() та __next__()). Цикл for неявно використовує ітератори. Comprehensions, генераторні вирази та функції з yield створюють ітератори. Розуміння цього дозволяє створювати власні ефективні послідовності.

```
    my_list = [1, 2, 3]
    my_iter = iter(my_list)  # Отримуємо об'єкт-ітератор
    print(next(my_iter))  # 1
    print(next(my_iter))  # 2
```

4. Методичні рекомендації

Нижче наведено розв'язки типових задач, які демонструють практичне застосування вивчених конструкцій Python. Уважно вивчіть код та коментарі до нього, перш ніж переходити до самостійного виконання завдань.

Задача 1. Створення списку квадратів парних чисел

На основі заданого діапазону цілих чисел (наприклад, від 1 до N) створіть список, що містить квадрати лише тих чисел, які є парними. Використайте list comprehension.

```
    # Задаємо верхню межу діапазону
    n = 10
```

```
    # List comprehension: для кожного x від 1 до n (включно), якщо x парне, додаємо x**2 до списку
    squares_of_evens = [x ** 2 for x in range(1, n + 1) if x % 2 == 0]

    print(squares_of_evens)
```

Приклад вхідних та вихідних даних:

```
Вхід: n = 5 → Вихід: [4, 16]
Вхід: n = 10 → Вихід: [4, 16, 36, 64, 100]
Вхід: n = 1 → Вихід: []
```

Коментарі:

Це класичний приклад list comprehension з умовою фільтрації (if). Код замінює 3-4 рядки з циклом for і умовою if на один лаконічний і читабельний рядок. Діапазон range(1, n+1) гарантує, що число n буде враховано.

Задача 2. Інвертування словника (ключ↔значення)

Дано словник. Створіть новий словник, в якому ключами будуть значення вихідного словника, а значеннями – список ключів, які мали це значення. Використайте defaultdict для зручного формування списків.

```
    from collections import defaultdict

    def invert_dict(original_dict):
        """
        Інвертує словник, збираючи всі ключі з однаковим значенням у список.
        """
        inverted = defaultdict(list)  # Фабрика за замовчуванням - порожній список
        for key, value in original_dict.items():
            # Для кожного значення додаємо ключ у відповідний список
            inverted[value].append(key)
        # Повертаємо звичайний dict для наочності (опціонально)
        return dict(inverted)

    # Приклад використання
    original = {"a": 1, "b": 2, "c": 1, "d": 3, "e": 2}
    result = invert_dict(original)
    print(result)
```

Приклад вхідних та вихідних даних:

```
Вхід: {"a": 1, "b": 2, "c": 1} → Вихід: {1: ['a', 'c'], 2: ['b']}
Вхід: {"apple": "fruit", "carrot": "vegetable", "banana": "fruit"} → Вихід: {'fruit': ['apple', 'banana'], 'vegetable': ['carrot']}
Вхід: {} → Вихід: {}
```

Коментарі:

defaultdict(list) усуває необхідність перевірки if value not in inverted: та ініціалізації порожнього списку. Це стандартний патерн для угруповання даних. Функція dict() в кінці лише для презентації, defaultdict також можна повертати.

Задача 3. Підрахунок частоти символів у рядку

Для заданого рядка підрахуйте, скільки разів кожен символ зустрічається в ньому. Виведіть три найбільш поширені символи. Використайте Counter.

```python
from collections import Counter

def count_chars(text):
    """Підраховує частоти символів і повертає три найпопулярніші."""
    char_counter = Counter(text)  # Counter автоматично підраховує всі символи
    # most_common(n) повертає список кортежів (елемент, кількість)
    return char_counter.most_common(3)

# Приклади
texts = ["abracadabra", "hello world", "aabbcc"]
for t in texts:
    print(f"Текст: '{t}' -> Топ-3: {count_chars(t)}")
```

Приклад вхідних та вихідних даних:

```
Вхід: "abracadabra" → Вихід: [('a', 5), ('b', 2), ('r', 2)]
Вхід: "hello world" → Вихід: [('l', 3), ('o', 2), ('h', 1)] (пробіл також враховується)
Вхід: "aabbcc" → Вихід: [('a', 2), ('b', 2), ('c', 2)]
```

Коментарі:

Counter – найпотужніший інструмент для таких підрахунків. Він також підтримує операції віднімання, об'єднання (& для мінімумів, | для максимумів). Метод .most_common() дуже зручний для аналізу.

Задача 4. Генерація послідовності чисел Фібоначчі

Створіть генератор (за допомогою yield), який буде генерувати послідовність чисел Фібоначчі до певного максимального значення.

```
def fibonacci_gen(max_value):
    """
    Генератор чисел Фібоначчі, який видає числа, не перевищуючи max_value.
    """
    a, b = 0, 1
    while a <= max_value:
        yield a  # Повертаємо поточне число та "заморожуємо" стан
        a, b = b, a + b  # Оновлюємо значення для наступного кроку

# Використання генератора
max_val = 50
fib_gen = fibonacci_gen(max_val)

# Спосіб 1: Ітерація за допомогою for
print(f"Числа Фібоначчі до {max_val}:")
for num in fib_gen:
    print(num, end=" ")
print()
# Спосіб 2: Створення списку з нового генератора
fib_list = list(fibonacci_gen(20))
print(f"Список до 20: {fib_list}")
```

Приклад вхідних та вихідних даних:

```
Вхід: max_value = 10 → Вихід послідовності: 0 1 1 2 3 5 8
Вхід: max_value = 1 → Вихід: 0 1 1
Вхід: max_value = 0 → Вихід: 0
```

Коментарі:

Генератор (yield) ідеально підходить для послідовностей, які обчислюються "на льоту". Він не витрачає пам'ять на зберігання всієї послідовності. Зверніть увагу, що об'єкт-генератор (fib_gen) можна пройти лише один раз. Для повторного використання потрібно створити новий генератор.

Задача 5. Створення іменованих кортежів для зберігання даних про студентів

Використайте namedtuple для створення зручної структури "Студент". Створіть список таких кортежів і відобразіть дані у зручному вигляді.

```
from collections import namedtuple

# 1. Оголошення типу "Student"
Student = namedtuple('Student', ['name', 'age', 'major', 'gpa'])

# 2. Створення списку студентів
students = [
    Student("Анна Коваленко", 20, "Комп'ютерні науки", 4.5),
    Student("Іван Петренко", 22, "Математика", 4.2),
    Student("Марія Сидоренко", 21, "Фізика", 4.8),
]
# 3. Ітерація та доступ до полів за іменем
print("Список студентів:")
for student in students:
    # Доступ через крапку (student.name) набагато зрозуміліший, ніж за індексом (student[0])
    print(f" - {student.name}, {student.age} років, {student.major}, GPA: {student.gpa}")
# 4. Додаткові переваги: конвертація в словник, заміна окремих полів
anna_dict = students[0]._asdict()
print(f"\nПерший студент як словник: {anna_dict}")

# Створення нової версії кортежу з оновленим GPA
updated_ivan = students[1]._replace(gpa=4.4)
print(f"Оновлений GPA Івана: {updated_ivan.gpa}")
```

Приклад вхідних та вихідних даних:

```
Вихід програми:
Список студентів:
  - Анна Коваленко, 20 років, Комп'ютерні науки, GPA: 4.5
  - Іван Петренко, 22 років, Математика, GPA: 4.2
  - Марія Сидоренко, 21 років, Фізика, GPA: 4.8

Перший студент як словник: {'name': 'Анна Коваленко', 'age': 20, 'major': "Комп'ютерні науки", 'gpa': 4.5}
Оновлений GPA Івана: 4.4
```

Коментарі:

namedtuple – це легка альтернатива створенню повноцінного класу для простих контейнерів даних. Код стає самодокументованим (поля мають імена), зберігаючи всі переваги кортежу (незмінність, швидкість, можливість використання як ключ словника).

Задача 6. Фільтрація та трансформація даних за допомогою dictionary comprehension

Дано словник товарів та їх цін. Створіть новий словник, який містить тільки товари з ціною вище певного порогу, і одночасно застосуйте до них знижку 10%.

```
products = {
    "apple": 50,
    "banana": 20,
    "laptop": 25000,
    "mouse": 800,
    "keyboard": 1200,
    "tea": 90
}
price_threshold = 100
discount = 0.10  # 10%

# Dictionary comprehension з умовою (if) та трансформацією значень
filtered_with_discount = {
    item: price * (1 - discount)  # Нова ціна зі знижкою
    for item, price in products.items()
    if price > price_threshold   # Умова фільтрації
}

print("Товари зі знижкою (ціна > 100):")
for item, new_price in filtered_with_discount.items():
    print(f"  {item}: {new_price:.2f} грн")  # Форматування до двох знаків після коми
```

Приклад вхідних та вихідних даних:

Вхід: price_threshold = 100 → Вихід (словник): {'laptop': 22500.0, 'mouse': 720.0, 'keyboard': 1080.0}
Вхід: price_threshold = 1000 → Вихід: {'laptop': 22500.0, 'keyboard': 1080.0}
Вхід: price_threshold = 30000 → Вихід: {}

Коментарі:

Цей приклад показує силу dictionary comprehension, яка дозволяє в одному виразі відфільтрувати вхідні дані, трансформувати значення та створити новий словник. Формула price * (1 - discount) обчислює ціну після знижки.

Задача 7. Використання генераторного виразу для обробки великого файлу "на льоту"

Уявіть, що у вас є великий текстовий файл. Напишіть код, який за допомогою генераторного виразу знаходить усі рядки, що містять певне ключове слово, не завантажуючи весь файл в пам’ять.

```
    def find_lines(filename, keyword):
        """
        Генератор, який повертає рядки з файлу, що містять keyword.
        """
        with open(filename, 'r', encoding='utf-8') as file:
            # Генераторний вираз! Круглі дужки, а не квадратні.
            # Він читає файл по одному рядку за раз.
            matching_lines = (line.strip() for line in file if keyword in line)
            for line in matching_lines:
                yield line

    # Симуляція роботи (замість реального файлу використовуємо список рядків)
    # Уявімо, що це вміст файлу 'data.txt'
    simulated_file_content = [
        "Перший рядок про Python.\n",
        "Другий рядок про програмування.\n",
        "Третій рядок, знову про Python і алгоритми.\n",
        "Четвертий рядок не на тему.\n",
    ]

    # Записуємо симульований вміст у файл для прикладу
    with open('example_log.txt', 'w', encoding='utf-8') as f:
        f.writelines(simulated_file_content)

    # Використання генератора
    print("Рядки, що містять 'Python':")
    for found_line in find_lines('example_log.txt', 'Python'):
        print(f"  - {found_line}")
```

```
# Можна створити список, але тоді він завантажиться в пам'ять
# list_of_lines = list(find_lines('file.txt', 'Python'))
```

Приклад вхідних та вихідних даних:

```
Якщо вміст файлу example_log.txt як у коді вище, то вихід буде:
Рядки, що містять 'Python':
  - Перший рядок про Python.
  - Третій рядок, знову про Python і алгоритми.
```

Коментарі:

Генераторний вираз (...) всередині функції та ключове слово yield роблять цю функцію ефективною для обробки файлів будь-якого розміру. Файл читається по одному рядку, і як тільки знаходиться відповідний рядок, він негайно видається. Це заощаджує пам'ять порівняно з [line for line in file if keyword in line], який створив би список усіх відповідних рядків одразу.

Типові помилки і шляхи їх усунення

1. **Синтаксис comprehensions:** найпоширеніша помилка – неправильний порядок частин у comprehension. Пам'ятайте: [ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАБЕЛЬНИЙ_ОБ'ЄКТ if УМОВА]. Умова (if) завжди в кінці. Для кількох for та if вони йдуть у тому ж порядку, що й у звичайному вкладеному циклі.

   Помилка. [x if x > 0 for x in range(-5, 5)]

   Рішення. [x for x in range(-5, 5) if x > 0]

2. **Заміна defaultdict на звичайний dict:** спроба додати елемент до списку за неіснуючим ключем у звичайному словнику призведе до KeyError.

   Помилка.

   ```
   d = {}
   d['key'].append('value')  # KeyError!
   ```

   Рішення. Ініціалізуйте список або використовуйте defaultdict(list).

3. **Одноразове використання генераторів:** об'єкт-генератор (створений функцією з yield або генераторним виразом) можна пройти лише один раз. Після цього він "спорожнюється".

   Помилка.

   ```
   gen = (x for x in range(3))
   list1 = list(gen)  # [0, 1, 2]
   list2 = list(gen)  # [] - Порожньо!
   ```

Рішення. Створюйте новий генератор для кожного проходження або зберігайте дані в список/кортеж, якщо потрібно багаторазове використання.

4. **Забувають, що namedtuple – незмінний (immutable):** не можна змінити значення поля за допомогою присвоєння.

Помилка. student.name = "Нове ім'я" (викличе AttributeError)

Рішення. Використовуйте метод _replace() для створення нової копії зі зміненими полями.

5. **Неправильне використання Counter для оновлення:** для динамічного підрахунку елементів у циклі потрібно оновлювати лічильник, а не створювати новий.

Помилка (неповний підрахунок):

```
counter = Counter()
for item in data_stream:
    counter = Counter([item])    # Створюється НОВИЙ лічильник кожного разу!
```

Рішення. Використовуйте метод update() або інкрементуйте через ключ: counter[item] += 1.

Корисні поради

1. **Вибір інструменту:**

- list/dict/set comprehension – коли потрібно створити **нову колекцію** відомого розміру та у вас є готовий ітерабельний об'єкт.
- **Генератор (yield)** – коли послідовність обчислюється крок за кроком, є **дуже довгою або нескінченною**, або ви хочете розділити логіку створення даних та їх споживання.
- **Генераторний вираз** – як "легка" версія генератора, коли логіка проста (один вираз) і не потрібна повторна ітерація.
- **collections модуль** – це ваш "швейцарський ніж" для спеціалізованих, але дуже поширених задач: Counter для підрахунків, defaultdict для угруповання, namedtuple для структурованих даних.

2. **Читабельність vs. Лаконічність.** Не перевантажуйте comprehensions занадто складними вкладеними циклами та умовами. Якщо вираз стає довшим за 2-3 рядки, краще використати звичайні цикли for – це зробить код зрозумілішим для майбутніх змін.

3. **Ефективність.** Генераторні вирази та функції з yield економлять пам'ять, але якщо вам потрібен **випадковий доступ** (наприклад, my_data[10]) або потрібно пройти дані **кілька разів**, краще створити список чи кортеж.

4. **Тестування.** Для перевірки роботи генераторів створюйте з них список за допомогою list(my_generator). Це дозволить наочно побачити всі елементи, які видає генератор.

5. **Вивчення документації.** Обов'язково перегляньте офіційну документацію Python для модуля collections. Там ви знайдете й інші корисні типи, як-от deque (двостороння черга) або OrderedDict (словник, що зберігає порядок додавання елементів).

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Фільтрація списку чисел

Створіть за допомогою list comprehension список, що містить лише додатні числа з заданого списку numbers. Від'ємні числа та нуль потрібно ігнорувати.

```
Вхід: [-5, 10, 0, 3, -1, 7] → Вихід: [10, 3, 7]
Вхід: [1, 2, 3, 4, 5] → Вихід: [1, 2, 3, 4, 5]
Вхід: [-10, -20, -30] → Вихід: []
```

Завдання 1.2. Список довжин слів

Дано список слів words. За допомогою list comprehension створіть новий список, що містить довжини цих слів.

```
Вхід: ["Python", "is", "awesome"] → Вихід: [6, 2, 7]
Вхід: ["apple", "banana", "cherry"] → Вихід: [5, 6, 6]
Вхід: [""] → Вихід: [0]
```

Завдання 1.3. Створення словника степенів

Задано список цілих чисел nums. Створіть dictionary comprehension, де ключем буде число зі списку, а значенням – його квадрат.

```
Вхід: [1, 2, 3, 4] → Вихід: {1: 1, 2: 4, 3: 9, 4: 16}
Вхід: [5] → Вихід: {5: 25}
Вхід: [] → Вихід: {}
```

Завдання 1.4. Пошук спільних елементів

Дано два списки list1 та list2. За допомогою list comprehension створіть список, що містить елементи, які є одночасно в обох списках (перетин множин). Результат повинен містити лише унікальні значення.

```
Вхід: [1, 2, 3, 4], [3, 4, 5, 6] → Вихід: [3, 4]
Вхід: ["a", "b", "c"], ["c", "d", "e"] → Вихід: ["c"]
Вхід: [10, 20], [30, 40] → Вихід: []
```

Завдання 1.5. Форматування телефонних номерів

Є список рядків raw_numbers, що містять номери телефонів у різних форматах (наприклад, +380501234567, 0501234567, (050)123-45-67). За допомогою list comprehension створіть новий список, де кожен номер приведено до канонічного вигляду +380501234567. Для спрощення вважайте, що всі номери українські та починаються на 0 або +380. Видаліть усі символи, крім цифр, і додайте префікс +380, якщо його немає.

```
Вхід: ["0501234567", "+380501234567", "(050)123-45-67"] →
Вихід: ["+380501234567", "+380501234567", "+380501234567"]
Вхід: ["0679876543", "0631112233"] → Вихід: ["+380679876543",
"+380631112233"]
Вхід: [] → Вихід: []
```

Завдання 1.6. Унікальні символи в рядку

Дано рядок text. За допомогою set comprehension створіть множину, що містить усі унікальні символи (літери, цифри, знаки), які зустрічаються в цьому рядку.

```
Вхід: "hello world!" → Вихід: {'!', ' ', 'd', 'e', 'h', 'l',
'o', 'r', 'w'} (порядок може бути іншим)
Вхід: "abracadabra" → Вихід: {'a', 'b', 'c', 'd', 'r'}
Вхід: "123123" → Вихід: {'1', '2', '3'}
```

Завдання 1.7. Множина слів, що починаються на певну літеру

Дано список слів word_list. Створіть множину (set comprehension), яка містить усі слова зі списку, що починаються на задану літеру letter. Пошук має бути нечутливим до регістру (тобто, letter='a' має знайти слова, що починаються як на 'a', так і на 'A').

```
Вхід: ["Apple", "banana", "Apricot", "blueberry", "avocado"],
letter='a' → Вихід: {'Apple', 'Apricot', 'avocado'}
Вхід: ["Python", "java", "JavaScript", "C++"], letter='j' →
Вихід: {'java', 'JavaScript'}
Вхід: ["test", "text", "task"], letter='z' → Вихід: set()
(порожня множина)
```

Завдання 1.8. Сума квадратів за допомогою генераторного виразу

Задано список чисел numbers. Обчисліть суму квадратів цих чисел, використовуючи **генераторний вираз** (не list comprehension) та вбудовану функцію sum().

```
Вхід: [1, 2, 3, 4] → Вихід: 30 (1+4+9+16)
Вхід: [10, 20, 30] → Вихід: 1400 (100+400+900)
Вхід: [] → Вихід: 0
```

Завдання 1.9. Генератор парних чисел

Створіть генераторний вираз (не функцію з yield!), який буде генерувати парні числа з діапазону від start до end (включно). Використайте його для створення списку.

```
Вхід: start=2, end=10 → Вихід (список): [2, 4, 6, 8, 10]
Вхід: start=1, end=5 → Вихід: [2, 4]
Вхід: start=7, end=8 → Вихід: [8]
```

Завдання 1.10. Довжина найдовшого слова

Дано список рядків sentences. За допомогою **генераторного виразу** та функції max() знайдіть довжину найдовшого слова у всіх реченнях. Для розбиття речень на слова можна використати str.split() без аргументів.

```
Вхід: ["Python is great", "I love programming"] → Вихід: 11
(довжина слова "programming")
Вхід: ["Short", "Words"] → Вихід: 5
Вхід: ["A B C", "D E"] → Вихід: 1
```

Частина 2. Середні завдання

Завдання 2.1. Робота з даними студентів (namedtuple)

Створіть namedtuple з іменем Student з полями: first_name, last_name, group, average_mark. Напишіть функцію get_best_student(students), яка отримує список таких кортежів і повертає прізвище та ім’я студента з найвищим середнім балом (у форматі "Прізвище І."). Якщо таких студентів кілька, поверніть першого знайденого.

Вхід: [Student("Іван", "Петренко", "КН-101", 4.5), Student("Марія", "Сидоренко", "КН-101", 4.8), Student("Петро", "Іваненко", "КН-102", 4.5)] → Вихід: "Сидоренко М."
Вхід: [Student("Анна", "Коваль", "ПМ-201", 3.2)] → Вихід: "Коваль А."
Вхід: [] → Вихід: None

Завдання 2.2. Групування товарів за категорією (defaultdict)

Є список словників products, кожен з яких описує товар і має ключі "name" та "category". Використовуйте defaultdict, щоб створити словник, де ключами будуть назви категорій, а значеннями – списки назв товарів, що належать до цієї категорії.

Вхід: [{"name": "iPhone", "category": "Електроніка"}, {"name": "Банан", "category": "Продукти"}, {"name": "Ноутбук", "category": "Електроніка"}, {"name": "Хліб", "category": "Продукти"}]
Вихід: defaultdict(<class 'list'>, {'Електроніка': ['iPhone', 'Ноутбук'], 'Продукти': ['Банан', 'Хліб']})
Вхід: [{"name": "Книга", "category": "Навчання"}] → Вихід: defaultdict(<class 'list'>, {'Навчання': ['Книга']})
Вхід: [] → Вихід: defaultdict(<class 'list'>, {})

Завдання 2.3. Аналіз текстів (Counter)

Напишіть функцію analyze_text(text, n), яка приймає рядок text та число n. Функція має повернути список кортежів [(слово1, кількість1), ...] для n найчастіше вживаних слів у тексті. Слова слід розбивати за пробілами, приводити до нижнього регістру та видаляти знаки пунктуації на початку та в кінці слова (можна скористатися str.strip(".,!?;:")).

Вхід: "Привіт, світ! Світ прекрасний. Привіт усім!", n=2 → Вихід: [('привіт', 2), ('світ', 2)]
Вхід: "А й справді, що то за сон? Що за сон?", n=1 → Вихід: [('що', 2)] (або [('сон', 2)], залежно від логіки підрахунку)
Вхід: "Тест", n=5 → Вихід: [('тест', 1)]

Завдання 2.4. Генератор послідовності факторіалів

Створіть функцію-генератор factorial_gen(max_n), яка за допомогою ключового слова yield генерує послідовність факторіалів чисел від 0! до max_n! включно. Нагадування: 0! = 1.

```
Вхід: max_n=5 → Вихід (список з генератора): [1, 1, 2, 6, 24, 120]
Вхід: max_n=0 → Вихід: [1]
Вхід: max_n=3 → Вихід: [1, 1, 2, 6]
```

Завдання 2.5. Генератор "читача" великого файлу по блоках

Створіть функцію-генератор read_in_chunks(file_path, chunk_size=1024), яка симулює читання великого файлу блоками (чанками) фіксованого розміру. На кожній ітерації генератор має видавати (yield) рядок, що представляє один блок симульованих даних (наприклад, блок із chunk_size символів ‘A’). Функція приймає параметр total_size (загальний розмір симульованого файлу в байтах). Генератор повинен видавати блоки, поки не буде "прочитано" total_size байт.

```
Вхід: total_size=2500, chunk_size=1000 → Вихід (список блоків): ['A'*1000, 'A'*1000, 'A'*500] (3 ітерації)
Вхід: total_size=500, chunk_size=1000 → Вихід: ['A'*500] (1 ітерація)
Вхід: total_size=0, chunk_size=1000 → Вихід: [] (0 ітерацій)
```

Частина 3. Складні завдання

Завдання 3.1. Оптимізація обробки великого лог-файлу (Counter + генератор)

Уявіть, що у вас є дуже великий текстовий лог-файл сервера (симулюйте його як генератор, що видає рядки). Кожен рядок містить IP-адресу користувача. Напишіть функцію find_top_ips(log_generator, top_n), яка отримує генератор log_generator (повертає рядки логу по одному) і ціле число top_n. Функція має повернути список top_n найактивніших IP-адрес (найбільша кількість входжень у лозі), використовуючи Counter. Метою є обробка потенційно нескінченного потоку даних без завантаження всього логу в пам’ять.

```
Вхід (генератор видає): ["192.168.1.1", "10.0.0.1", "192.168.1.1", "172.16.0.1", "10.0.0.1", "192.168.1.1"], top_n=2
Вихід: [('192.168.1.1', 3), ('10.0.0.1', 2)]
Вхід (генератор видає): ["127.0.0.1"] * 5, top_n=1 → Вихід: [('127.0.0.1', 5)]
```

```
Вхід: порожній генератор, top_n=5 → Вихід: []
Складність: Потрібно написати як генератор, що симулює файл,
так і функцію аналізу, яка працює з цим генератором.
```

Завдання 3.2. Складний генератор для розпарсування структурованих даних

Напишіть функцію-генератор parse_config(config_lines), яка імітує парсинг конфігураційного файлу. Генератор отримує ітерабельний об'єкт (наприклад, список) рядків. Рядки можуть бути: 1) порожніми, 2) коментарями (починаються з #), 3) секціями (у квадратних дужках, напр. [Database]), 4) параметрами (у форматі key = value). Генератор має ігнорувати порожні рядки та коментарі. Для кожного параметра він має видавати кортеж (section, key, value), де section – назва поточної секції, або "GLOBAL", якщо параметр оголошено до першої секції.

```
Вхід:
# Налаштування
host = localhost
[Database]
name = test
port = 5432
[Logging]
level = INFO
Вихід (послідовність yield):
("GLOBAL", "host", "localhost")
("Database", "name", "test")
("Database", "port", "5432")
("Logging", "level", "INFO")
Вхід: [] → Вихід: нічого (порожня послідовність).
Складність: Треба зберігати стан (поточну секцію) між
викликами yield.
```

Завдання 3.3. Ітератор для обходу вкладеної структури (власний ітератор)

Створіть клас NestedListIterator, який реалізує протокол ітератора (методи __iter__() та __next__()). Цей ітератор має "розгладжувати" довільно вкладені списки, повертаючи кожен елемент по черзі, як якби це був плоский список. Не використовуйте рекурсію в __next__, використайте явний стек (list як LIFO).

```
Вхід: nested_list = [1, [2, 3], [4, [5, 6]], 7]
Вихід (послідовні виклики next()): 1, 2, 3, 4, 5, 6, 7
```

```
Вхід: [["a"], "b"], "c"] → Вихід: "a", "b", "c"
Вхід: [] → Вихід: виняток StopIteration при першому виклику next().
Складність: Реалізація нерекурсивного алгоритму обходу зі збереженням стану в об’єкті ітератора.
```

6. Питання для самоперевірки

1. Які три основні типи comprehensions існують у Python? Наведіть приклад синтаксису кожного з них.

2. Чим відрізняється генераторний вираз (x for x in range(10)) від list comprehension [x for x in range(10)] з точки зору продуктивності та пам’яті?

3. У чому полягає ключова відмінність між функцією, яка повертає список, та функцією-генератором з використанням ключового слова yield?

4. Для чого використовується defaultdict з модуля collections? Наведіть конкретний приклад задачі, де його використання є значно зручнішим, ніж робота зі звичайним dict.

5. Що поверне метод Counter("abracadabra").most_common(2)? Поясніть, що роблять клас Counter та його метод most_common.

6. Які переваги дає використання namedtuple порівняно зі звичайним кортежем або словником для зберігання структурованих даних (наприклад, інформації про студента)?

7. Чи можна використовувати if-else у list comprehension? Якщо так, то яким чином відрізняється синтаксис [x if condition else y for item in iterable] від синтаксису з фільтром [x for item in iterable if condition]?

8. Що станеться, якщо ви спробуєте двічі пройтись циклом for по одному і тому ж об’єкту-генератору? Чому так відбувається?

9. Як за допомогою set comprehension отримати множину усіх унікальних символів, що зустрічаються у двох різних рядках?

10. Яку помилку ви отримаєте, якщо спробуєте змінити значення поля namedtuple безпосередньо через присвоєння (наприклад, student.name = "Нове ім’я")? Як правильно оновити значення?

11. Яким чином можна створити словник за допомогою dictionary comprehension, інвертувавши попередній словник (поміняти ключі та значення місцями)? Чи завжди це буде коректно?

12. Що таке протокол ітератора? Які два методи повинен реалізувати об’єкт, щоб бути ітератором?

13. Навіщо в конструкції [x**2 for x in range(10) if x % 2 == 0] використовується ключове слово if в кінці, а не if-else?

14. Чи може генераторний вираз або функція-генератор бути нескінченною? Наведіть приклад простого нескінченного генератора. Як безпечно отримати з нього кілька перших значень?

15. Припустимо, вам потрібно підрахувати кількість входжень кожного слова в великому тексті, який не влізає в пам'ять. Яку комбінацію інструментів (генератор, Counter) ви б використали для ефективного розв'язання цієї задачі?

16. Чи можна створити defaultdict, де за замовчуванням створювався б, наприклад, int зі значенням 10, а не 0? Як це реалізувати?

17. Як би ви переписали наступний код з використанням list comprehension?

```
result = []
for num in range(1, 11):
    if num % 3 == 0:
        result.append(num * 2)
```

18. Якщо виконати код c = Counter([1, 2, 2, 3, 3, 3]), то чому дорівнюватиме c[5]? Чи викличе це помилку KeyError?

19. У чому практична різниця між використанням yield в генераторі та використанням return зі списком? Коли краще використовувати кожен з підходів?

20. Як би ви пояснили концепцію "лінивих обчислень" (lazy evaluation) на прикладі генераторів у Python?

ЛАБОРАТОРНА РОБОТА №14
Класи та об’єкти

1. Мета

Засвоїти базові принципи об’єктно-орієнтованого програмування (ООП) на Python через практичне створення класів та об’єктів. Студенти навчаться оголошувати класи, визначати атрибути (екземпляра та класу) та методи (екземпляра, класу, статичні), використовувати конструктор __init__ для ініціалізації об’єктів, а також моделювати взаємодію між декількома об’єктами одного класу.

2. Завдання

1. Оволодіти синтаксисом оголошення класу в Python.
2. Навчитися створювати об’єкти (екземпляри) класу.
3. Розрізняти та застосовувати атрибути екземпляра та атрибути класу.
4. Освоїти роботу з конструктором __init__ для ініціалізації початкового стану об’єкта.
5. Навчитися проектувати та викликати методи екземпляра, класу та статичні методи.
6. Розвинути навички моделювання простих реальних сутностей за допомогою класів.
7. Реалізувати взаємодію між декількома об’єктами одного класу в програмі.

3. Короткі теоретичні відомості

Об’єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка використовує об’єкти та їх взаємодію для проектування програм. Основною ідеєю є об’єднання даних та методів для роботи з цими даними в єдину структуру – клас.

Клас – це шаблон або схема для створення об’єктів. Він визначає набір атрибутів (даних) і методів (функцій), які будуть у всіх об’єктів, створених на його основі. Уявіть собі креслення будинку (це клас).

Об’єкт (екземпляр) – це конкретна реалізація класу. Це "будинок", побудований за кресленням. Кожен об’єкт має власний набір даних (атрибутів), хоча методи у них спільні.

3.1. Оголошення класу та створення об’єкта

```
class Student:
    pass

# Створення об’єктів (екземплярів) класу Student
student1 = Student()
student2 = Student()
```

3.2. Конструктор __init__ та атрибути екземпляра

Конструктор __init__ – це спеціальний метод, який автоматично викликається при створенні нового об’єкта. Він використовується для ініціалізації початкових значень атрибутів екземпляра. Атрибути екземпляра – це змінні, що належать конкретному об’єкту. Визначаються зазвичай у конструкторі за допомогою ключового слова self.

```
class Student:
    def __init__(self, name, student_id):
        # Ініціалізація атрибутів екземпляра
        self.name = name            # Атрибут екземпляра
        self.student_id = student_id # Атрибут екземпляра
        self.grades = []            # Атрибут екземпляра

# Створення об’єктів з різними даними
student_alice = Student("Аліна Коваленко", "KN-001")
student_bohdan = Student("Богдан Шевченко", "KN-002")

print(student_alice.name)  # Виведе: Аліна Коваленко
print(student_bohdan.student_id)  # Виведе: KN-002
```

3.3. Методи екземпляра

Метод екземпляра – це функція, визначена всередині класу, яка працює з конкретним об’єктом. Першим параметром завжди є self, який посилається на поточний екземпляр.

```
class Student:
    def __init__(self, name):
        self.name = name
        self.grades = []

    # Метод екземпляра для додавання оцінки
    def add_grade(self, grade):
        self.grades.append(grade)
```

```
        # Метод екземпляра для розрахунку середнього балу
        def get_average(self):
            if not self.grades:
                return 0
            return sum(self.grades) / len(self.grades)

    student = Student("Олексій Мельник")
    student.add_grade(85)
    student.add_grade(92)
    print(f"Середній бал: {student.get_average()}")  # Виведе:
Середній бал: 88.5
```

3.4. Атрибути класу та методи класу (@classmethod)

Атрибут класу – це змінна, що належить самому класу, а не його екземплярам. Вона спільна для всіх об'єктів класу. Метод класу приймає першим параметром cls (посилання на клас, а не об'єкт) і містить декоратор @classmethod. Він часто використовується для створення альтернативних конструкторів.

```
    class Student:
        # Атрибут класу
        university = "Національний Університет"

        def __init__(self, name):
            self.name = name

        @classmethod
        def change_university(cls, new_name):
            # Змінюємо атрибут класу
            cls.university = new_name

    print(Student.university)  # Виведе: Національний Університет
    Student.change_university("Київський Політехнічний Інститут")
    print(Student.university)  # Виведе: Київський Політехнічний
Інститут
```

3.5. Статичні методи (@staticmethod)

Статичний метод – це звичайна функція, яка логічно пов'язана з класом, але не потребує доступу ні до атрибутів екземпляра (self), ні до атрибутів класу (cls). Він має декоратор @staticmethod і може бути викликаний як від імені класу, так і від імені об'єкта.

```
    class Student:
        def __init__(self, name):
            self.name = name

        @staticmethod
        def is_valid_grade(grade):
            # Перевіряє, чи оцінка в допустимому діапазоні
            return 0 <= grade <= 100

    print(Student.is_valid_grade(105))    # Виведе: False (Виклик від імені класу)
    student = Student("Марія")
    print(student.is_valid_grade(85))     # Виведе: True   (Виклик від імені об'єкта)
```

3.6. Взаємодія між об'єктами

Об'єкти можуть взаємодіяти один з одним, передаючи повідомлення (викликаючи методи) або звертаючись до атрибутів іншого об'єкта.

```
    class BankAccount:
        def __init__(self, owner, balance=0):
            self.owner = owner
            self.balance = balance

        def transfer(self, other_account, amount):
            # self - об'єкт, з якого переводять
            # other_account - об'єкт, на який переводять
            if amount > self.balance:
                print("Недостатньо коштів на рахунку.")
                return
            self.balance -= amount
            other_account.balance += amount
            print(f"Переказ {amount} грн. від {self.owner} до {other_account.owner}")
    account1 = BankAccount("Іван", 500)
    account2 = BankAccount("Петро", 300)
    account1.transfer(account2, 200)
    print(account1.balance)  # Виведе: 300
    print(account2.balance)  # Виведе: 500
```

Порівняння методів у класі

Тип методу	Декоратор	Перший параметр	Доступ до атрибутів	Доступ до атрибутів	Основне призначення

			екземпляра	класу	
Метод екземпляра	Немає	self	Так (через self)	Так (через self.__class__)	Робота з даними конкретного об’єкта
Метод класу	@classmethod	cls	Ні	Так (через cls)	Робота з атрибутами класу, альтернативні конструктори
Статичний метод	@staticmethod	Немає	Ні	Ні	Допоміжні функції, які логічно належать класу

4. Методичні рекомендації

Нижче наведено розв’язання типових задач, які допоможуть засвоїти основні концепції роботи з класами та об’єктами в Python. Уважно вивчіть код та коментарі до нього.

Задача 1. Створення класу для представлення книги

Створіть клас Book. Клас має мати конструктор (__init__), який приймає та ініціалізує атрибути екземпляра: title (назва), author (автор), year (рік видання) та is_read (ознака прочитання, за замовчуванням False). Також створіть метод екземпляра mark_as_read(), який змінює стан книги на прочитану, та метод __str__() для кращого текстового представлення об’єкта.

```
class Book:
    """Клас, що представляє книгу в бібліотеці."""

    def __init__(self, title, author, year, is_read=False):
        """Конструктор класу Book.

        Args:
            title (str): Назва книги.
            author (str): Автор книги.
            year (int): Рік видання.
```

```
            is_read (bool, optional): Стан прочитання. За замовчуванням False.
        """
        self.title = title
        self.author = author
        self.year = year
        self.is_read = is_read

    def mark_as_read(self):
        """Позначає книгу як прочитану."""
        self.is_read = True
        print(f'Книгу "{self.title}" позначено як прочитану.')

    def __str__(self):
        """Повертає рядкове представлення інформації про книгу."""
        status = "прочитана" if self.is_read else "не прочитана"
        return (f'Книга: "{self.title}", Автор: {self.author}, '
                f'Рік: {self.year}, Статус: {status}.')

# Приклад використання класу
book1 = Book("1984", "Джордж Орвелл", 1949)
print(book1)  # Викликається book1.__str__()
book1.mark_as_read()
print(book1)

book2 = Book("Кобзар", "Тарас Шевченко", 1840, is_read=True)
print(book2)
```

Приклад вхідних/вихідних даних:

Вхід: Book("1984", "Джордж Орвелл", 1949) → Вихід (після створення): Книга: "1984", Автор: Джордж Орвелл, Рік: 1949, Статус: не прочитана. → Вихід (після mark_as_read()): Книга: "1984", Автор: Джордж Орвелл, Рік: 1949, Статус: прочитана.
Вхід: Book("Маленький принц", "Антуан де Сент-Екзюпері", 1943) → Вихід: Книга: "Маленький принц", Автор: Антуан де Сент-Екзюпері, Рік: 1943, Статус: не прочитана.
Вхід: Book("Кобзар", "Тарас Шевченко", 1840, is_read=True) → Вихід: Книга: "Кобзар", Автор: Тарас Шевченко, Рік: 1840, Статус: прочитана.

Коментарі:

У цій задачі використано конструктор для ініціалізації обов'язкових та опціональних атрибутів, метод екземпляра для зміни стану об'єкта та

"магічний" метод __str__ для зручного виведення інформації. Метод __str__ викликається автоматично функціями print() та str().

Задача 2. Робота з атрибутами класу та екземпляра

Створіть клас Employee (співробітник). Клас має мати атрибут класу company = "TechCorp". У конструкторі ініціалізуйте атрибути екземпляра name (ім'я) та position (посада). Додайте метод класу change_company(new_name), який змінює назву компанії для всіх співробітників. Також створіть статичний метод is_valid_position(position), який перевіряє, чи є передана посада в списку допустимих (наприклад, ["менеджер", "розробник", "тестувальник"]).

```
    class Employee:
        """Клас, що представляє співробітника компанії."""

        # Атрибут класу
        company = "TechCorp"
        # Список допустимих посад (можна використовувати як константу класу)
        VALID_POSITIONS = ["менеджер", "розробник", "тестувальник", "дизайнер"]

        def __init__(self, name, position):
            """Конструктор класу Employee."""
            self.name = name
            self.position = position

        def __str__(self):
            return f"{self.name}, {self.position}, компанія: {Employee.company}"

        @classmethod
        def change_company(cls, new_name):
            """Змінює назву компанії для всіх співробітників.

            Args:
                new_name (str): Нова назва компанії.
            """
            cls.company = new_name
            print(f"Назву компанії змінено на: {new_name}")

        @staticmethod
        def is_valid_position(position):
```

```
            """Перевіряє, чи є посада допустимою.

            Args:
                position (str): Назва посади для перевірки.

            Returns:
                bool: True, якщо посада допустима, інакше False.
            """
            return position.lower() in Employee.VALID_POSITIONS

    # Робота з атрибутом класу
    print(f"Поточна компанія (до створення об'єктів): {Employee.company}")

    emp1 = Employee("Анна Іваненко", "розробник")
    emp2 = Employee("Олег Петренко", "менеджер")

    print(emp1)  # Виведе: Анна Іваненко, розробник, компанія: TechCorp
    print(emp2)  # Виведе: Олег Петренко, менеджер, компанія: TechCorp
    # Виклик методу класу
    Employee.change_company("InnovateLLC")
    print(emp1)  # Тепер компанія для всіх об'єктів змінилася
    print(emp2)  # Виведе: ... компанія: InnovateLLC

    # Виклик статичного методу
    print(Employee.is_valid_position("розробник"))  # True
    print(Employee.is_valid_position("бухгалтер"))  # False
    print(emp1.is_valid_position("дизайнер"))  # True (виклик від об'єкта)
```

Приклад вхідних/вихідних даних:

Вхід: emp1 = Employee("Анна", "розробник") → Вихід (після print(emp1)): Анна, розробник, компанія: TechCorp
Вхід: Employee.change_company("InnovateLLC") → Вихід: Назву компанії змінено на: InnovateLLC. Після цього всі об'єкти покажуть нову компанію.
Вхід: Employee.is_valid_position("бухгалтер") → Вихід: False

Коментарі:

Задача демонструє різницю між атрибутами класу (company) та екземпляра (name, position). Зміна атрибута класу впливає на всі

екземпляри. Статичний метод – це звичайна функція-утиліта, що логічно належить класу.

Задача 3. Взаємодія між об'єктами

Створіть клас BankAccount (банківський рахунок). Клас має мати атрибути owner (власник) та balance (баланс, за замовчуванням 0). Реалізуйте метод deposit(amount) для поповнення рахунку та метод withdraw(amount) для зняття коштів (з перевіркою, чи достатньо коштів). Створіть метод transfer(self, other_account, amount), який дозволяє переказати кошти з поточного рахунку (self) на інший рахунок (other_account).

```
    class BankAccount:
        """Клас, що моделює простий банківський рахунок."""
        def __init__(self, owner, balance=0.0):
            """Конструктор класу BankAccount.

            Args:
                owner (str): Ім'я власника рахунку.
                balance (float, optional): Початковий баланс. За
замовчуванням 0.0.
            """
            self.owner = owner
            self.balance = balance
           print(f"Рахунок для {owner} створено. Баланс: {balance}
грн.")

        def deposit(self, amount):
            """Поповнює рахунок на вказану суму.

            Args:
                amount (float): Сума для поповнення.
            """
            if amount > 0:
                self.balance += amount
                print(f"Поповнено рахунок {self.owner} на {amount}
грн. Новий баланс: {self.balance} грн.")
            else:
                print("Сума поповнення має бути додатною.")

        def withdraw(self, amount):
            """Знімає кошти з рахунку, якщо це можливо.
```

```
            Args:
                amount (float): Сума для зняття.
            """
            if amount > self.balance:
                print(f"Помилка. Недостатньо коштів на рахунку
{self.owner}. Доступно: {self.balance} грн.")
            elif amount > 0:
                self.balance -= amount
                print(f"Знято {amount} грн. з рахунку
{self.owner}. Новий баланс: {self.balance} грн.")
            else:
                print("Сума зняття має бути додатною.")

        def transfer(self, other_account, amount):
            """Переказує кошти з поточного рахунку на інший.

            Args:
                other_account (BankAccount): Об'єкт іншого
рахунку.
                amount (float): Сума для переказу.
            """
            print(f"\nСпроба переказу {amount} грн. від
{self.owner} до {other_account.owner}...")
            if amount > self.balance:
                print(f"Помилка. Недостатньо коштів у
{self.owner}.")
            elif amount <= 0:
                print("Помилка. Сума переказу має бути додатною.")
            else:
                # Знімаємо кошти з поточного рахунку
                self.balance -= amount
                # Додаємо кошти на інший рахунок
                other_account.balance += amount
                print(f"Успішно! Переказ {amount} грн.
завершено.")
                print(f"Баланс {self.owner}: {self.balance} грн.")
                print(f"Баланс {other_account.owner}:
{other_account.balance} грн.")

    # Створення рахунків
    account_alice = BankAccount("Аліса", 1000)
    account_bob = BankAccount("Богдан", 500)

    # Взаємодія між об'єктами
    account_alice.withdraw(200)
```

```
account_bob.deposit(150)

# Переказ коштів
account_alice.transfer(account_bob, 300)
# Спроба переказу при недостатньому балансі
account_bob.transfer(account_alice, 1000)
```

Приклад вхідних/вихідних даних:

Вхід: account_alice = BankAccount("Аліса", 1000) → Вихід: Рахунок для Аліса створено. Баланс: 1000 грн.
Вхід: account_alice.transfer(account_bob, 300) → Вихід: Успішно! Переказ 300 грн. завершено. Баланс Аліса: 500 грн. Баланс Богдан: 950 грн. (за умови, що у Богдана було 500+150=650).
Вхід: account_bob.withdraw(2000) → Вихід: Помилка. Недостатньо коштів на рахунку Богдан. Доступно: 950 грн.

Коментарі:

Ключовий момент тут – метод transfer приймає інший об’єкт того ж класу (other_account) як аргумент. Це класичний приклад взаємодії об’єктів: один об’єкт змінює стан іншого через його публічний інтерфейс (в даному випадку, безпосереднє звернення до атрибута balance, але в реальних системах це робилося б через методи).

Задача 4. Використання методу класу як альтернативного конструктора

Розширте клас Book із Задачі 1. Додайте атрибут класу library_name = "Центральна бібліотека". Створіть метод класу from_string(cls, book_str), який приймає рядок у форматі "Назва; Автор; Рік; Статус" (наприклад, "Степ; Олександр Довженко; 1932; True") і повертає новий об’єкт класу Book, створений на основі цих даних.

```
class Book:
    """Клас, що представляє книгу в бібліотеці."""

    library_name = "Центральна бібліотека"

    def __init__(self, title, author, year, is_read=False):
        self.title = title
        self.author = author
        self.year = year
        self.is_read = is_read
```

```
        def __str__(self):
            status = "прочитана" if self.is_read else "не
прочитана"
            return (f'Книга: "{self.title}". Автор: {self.author}. '
                    f'Рік: {self.year}. Статус: {status}.
Бібліотека: {Book.library_name}')

        @classmethod
        def from_string(cls, book_str):
            """Альтернативний конструктор. Створює об'єкт Book з рядка.

            Формат рядка: "Назва; Автор; Рік; Статус"
            Статус має бути 'True' або 'False' (рядок).

            Args:
                book_str (str): Рядок з даними книги.

            Returns:
                Book: Новий об'єкт класу Book.
            """
            # Розділяємо рядок за роздільником ';'
            parts = book_str.split(';')
            # Видаляємо зайві пробіли навколо кожного елемента
            parts = [part.strip() for part in parts]

            # Розпаковуємо частини. Перетворюємо рік у число, а
статус у булеве значення.
            title, author, year_str, is_read_str = parts
            year = int(year_str)
            # Безпечне перетворення рядка у булеве значення
            is_read = (is_read_str.lower() == 'true')

            # Використовуємо основний конструктор класу через
`cls`
            return cls(title, author, year, is_read)


    # Створення об'єкта звичайним способом
    book1 = Book("Майстер і Маргарита", "Михайло Булгаков", 1966)
    print(book1)

    # Зміна атрибута класу
    Book.library_name = "Міська бібліотека №1"
    print(book1)  # Зміниться назва бібліотеки у всіх об'єктів
```

```
    # Створення об'єкта за допомогою альтернативного конструктора (методу класу)
    book_data_string = "Тіні забутих предків; Михайло Коцюбинський; 1911; False"
    book2 = Book.from_string(book_data_string)
    print(f"\nКнига, створена з рядка:")
    print(book2)

    # Інший приклад
    book3 = Book.from_string("Захар Беркут; Іван Франко; 1883; True")
    print(book3)
```

Приклад вхідних/вихідних даних:

Вхід: Book.from_string("Тіні забутих предків; Михайло Коцюбинський; 1911; False") → Вихід (після print): Книга: "Тіні забутих предків". Автор: Михайло Коцюбинський. Рік: 1911. Статус: не прочитана. Бібліотека: Міська бібліотека №1
Вхід: Book.library_name = "Нова бібліотека" → Впливає на всі об'єкти, включаючи book1, book2, book3.
Вхід: Book.from_string("Камінний хрест; Василь Стефаник; 1900; True") → Вихід: ... Статус: прочитана. ...

Коментарі:

Метод класу from_string є "альтернативним конструктором". Він корисний, коли дані для створення об'єкта надходять у специфічному форматі (наприклад, з файлу CSV або відповіді API). Він інкапсулює логіку парсингу даних всередині класу.

Типові помилки і шляхи їх усунення

1. **Забувають про self у методах екземпляра.**
   Помилка. def get_name(): return name замість def get_name(self): return self.name
   Рішення. Перший параметр у методі екземпляра завжди має бути self. Через нього метод отримує доступ до атрибутів і інших методів конкретного об'єкта.
2. **Плутають атрибути класу та екземпляра.**
   Помилка. Змінюють атрибут класу через self (self.company = "Нова"), що призводить до створення однойменного атрибута екземпляра, який перекриває атрибут класу.
   Рішення. Для зміни атрибута класу всередині методу екземпляра використовуйте self.__class__.company = "Нова" або

НазваКласу.company = "Нова". Для зміни ззовні – використовуйте безпосередньо ім’я класу.

3. **Неправильне використання статичних методів.**

   Помилка. У статичному методі намагаються звернутися до self або cls.

   Рішення. Статичні методи не отримують жодних неявних аргументів. Вони мають бути абсолютно незалежними від стану об’єкта чи класу. Використовуйте їх для утиліт, пов’язаних з предметною областю класу.

4. **Неоголошення атрибутів у __init__.**

   Помилка. Додавання атрибутів об’єкту "на льоту" в інших методах, що ускладнює розуміння структури класу.

   Рішення. Намагайтеся ініціалізувати всі можливі атрибути екземпляра в конструкторі __init__, навіть значеннями за замовчуванням (наприклад, None, [], 0). Це робить код зрозумілішим.

5. **Помилки при взаємодії об’єктів.**

   Помилка. У методі transfer не перевіряється, чи other_account є об’єктом того ж класу, що може призвести до помилок.

   Рішення. Додайте просту перевірку типу: if not isinstance(other_account, BankAccount): raise TypeError("Очікується об’єкт BankAccount"). Це покращує надійність коду.

Корисні поради

1. **Іменування.** Назви класів пишіть у CamelCase (наприклад, BankAccount). Назви методів та атрибутів – у snake_case (наприклад, mark_as_read). Це стандарт PEP 8.

2. **Docstring.** Обов’язково документуйте клас та його методи за допомогою docstring (тривірні лапки). Це допоможе вам та іншим розробникам зрозуміти призначення коду.

3. **Принцип єдиного відповідальності (SRP).** Намагайтеся, щоб кожен клас відповідав за одну конкретну сутність або дію. Наприклад, клас Book відповідає за зберігання даних про книгу, а не за її пошук у базі даних.

4. **Почніть з простих класів.** Не намагайтеся відразу врахувати всі можливі ситуації. Створіть робочу базову версію класу, а потім поступово розширюйте його функціональність, додаючи методи та атрибути.

5. **Використовуйте __str__ для налагодження.** Метод __str__ дуже корисний для швидкого перегляду стану об’єкта під час налагодження програми за допомогою print().

6. **Розмежування методів.** Чітко розумійте, коли використовувати метод екземпляра (для дій з конкретним об’єктом), метод класу (для дій, що стосуються всієї категорії об’єктів) та статичний метод (для допоміжних функцій).

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Клас "Точка"

Створіть клас Point, який представляє точку в двовимірному просторі. Клас має мати конструктор __init__(self, x, y), що приймає координати x та y. Додайте метод __str__(self), який повертає рядок у форматі "(x, y)".

```
Вхід: p = Point(3, 4) → Вихід при print(p): (3, 4)
Вхід: p = Point(-1, 0) → Вихід при print(p): (-1, 0)
Вхід: p = Point(0, 0) → Вихід при print(p): (0, 0)
```

Завдання 1.2. Клас "Книга" (базовий варіант)

Створіть клас Book. Конструктор має приймати параметри title (назва) та author (автор). Додайте метод get_info(self), який повертає рядок: "Книга: [назва], Автор: [автор]".

```
Вхід: b = Book("1984", "Джордж Орвелл") → Вихід при
b.get_info(): "Книга: 1984, Автор: Джордж Орвелл"
Вхід: b = Book("Кобзар", "Тарас Шевченко") → Вихід: "Книга:
Кобзар, Автор: Тарас Шевченко"
Вхід: b = Book("Маленький принц", "Антуан де Сент-Екзюпері") →
Вихід: "Книга: Маленький принц, Автор: Антуан де Сент-
Екзюпері"
```

Завдання 1.3. Клас "Прямокутник"

Створіть клас Rectangle. Конструктор має приймати width (ширина) та height (висота). Додайте метод area(self), який обчислює та повертає площу прямокутника.

```
Вхід: r = Rectangle(5, 3) → Вихід при r.area(): 15
Вхід: r = Rectangle(7, 2) → Вихід: 14
Вхід: r = Rectangle(4, 4) → Вихід: 16
```

Завдання 1.4. Клас "Студент" (ініціалізація)

Створіть клас Student. Конструктор має приймати name (ім'я) та student_id (номер студентського квитка). Атрибути мають зберігатися в self.name та self.student_id. Додайте метод introduce(self), який повертає: "Мене звати [ім'я], мій номер студентського: [номер]".

```
Вхід: s = Student("Олена Петренко", "KN-101") → Вихід при s.introduce(): "Мене звати Олена Петренко, мій номер студентського: KN-101"
Вхід: s = Student("Іван Сидоренко", "KN-102") → Вихід: "Мене звати Іван Сидоренко, мій номер студентського: KN-102"
Вхід: s = Student("Марія Коваль", "KN-103") → Вихід: "Мене звати Марія Коваль, мій номер студентського: KN-103"
```

Завдання 1.5. Клас "Рахунок" з балансом

Створіть клас Account. Конструктор має приймати owner (власник) та опціонально balance (баланс, за замовчуванням 0). Додайте метод show_balance(self), який повертає рядок: "Баланс рахунку [власник]: [баланс] грн."

```
Вхід: a = Account("Аліса") → Вихід при a.show_balance(): "Баланс рахунку Аліса: 0 грн."
Вхід: a = Account("Богдан", 1000) → Вихід: "Баланс рахунку Богдан: 1000 грн."
Вхід: a = Account("Олександр", 250.5) → Вихід: "Баланс рахунку Олександр: 250.5 грн."
```

Завдання 1.6. Клас "Світлофор" зі зміною стану

Створіть клас TrafficLight. Конструктор ініціалізує атрибут color (колір) зі значенням "червоний". Додайте методи: change_to_green(self) (змінює колір на "зелений") та get_color(self) (повертає поточний колір).

```
Вхід: tl = TrafficLight() → Вихід при tl.get_color(): "червоний" → Після tl.change_to_green() → Вихід при tl.get_color(): "зелений"
```

```
Вхід: tl = TrafficLight() → Вихід: tl.change_to_green() →
Вихід при tl.get_color(): "зелений"
Вхід: tl = TrafficLight() → Вихід при tl.get_color() →
tl.change_to_green() → tl.get_color(): "червоний" → "зелений"
```

Завдання 1.7. Клас "Кошик для покупок"

Створіть клас ShoppingCart. Конструктор ініціалізує порожній список items. Додайте метод add_item(self, item_name) (додає товар у кошик) та метод show_items(self) (повертає список товарів у кошику).

```
Вхід: cart = ShoppingCart() → cart.add_item("Яблука") →
cart.add_item("Хліб") → Вихід при cart.show_items():
["Яблука", "Хліб"]
Вхід: cart = ShoppingCart() → cart.add_item("Молоко") → Вихід:
["Молоко"]
Вхід: cart = ShoppingCart() → cart.add_item("Цукор") →
cart.add_item("Кава") → cart.add_item("Чай") → Вихід:
["Цукор", "Кава", "Чай"]
```

Завдання 1.8. Клас "Коло" з обчисленням довжини

Створіть клас Circle. Конструктор приймає radius (радіус). Додайте метод circumference(self), який обчислює та повертає довжину кола за формулою 2 * 3.14 * radius.

```
Вхід: c = Circle(5) → Вихід при c.circumference(): 31.4
Вхід: c = Circle(1) → Вихід: 6.28
Вхід: c = Circle(10) → Вихід: 62.8
```

Завдання 1.9. Клас "Таймер"

Створіть клас Timer. Конструктор ініціалізує атрибут seconds (секунди) зі значенням 0. Додайте методи: tick(self) (збільшує секунди на 1) та reset(self) (скидає секунди до 0).

```
Вхід: t = Timer() → t.tick() → t.tick() → Вихід при t.seconds:
2 → Після t.reset() → Вихід при t.seconds: 0
Вхід: t = Timer() → t.tick() → Вихід: 1
Вхід: t = Timer() → Вихід при t.seconds: 0
```

Завдання 1.10. Клас "Контакт" з номером телефону

Створіть клас Contact. Конструктор приймає name (ім’я) та phone (номер телефону). Додайте метод call(self), який повертає рядок: "Дзвонимо [ім’я] за номером [номер]".

```
Вхід: c = Contact("Олег", "+380501234567") → Вихід при c.call(): "Дзвонимо Олег за номером +380501234567"
Вхід: c = Contact("Анна", "+380671112233") → Вихід: "Дзвонимо Анна за номером +380671112233"
Вхід: c = Contact("Служба порятунку", "101") → Вихід: "Дзвонимо Служба порятунку за номером 101"
```

Частина 2. Середні завдання

Завдання 2.1. Клас "Користувач" з атрибутом класу

Створіть клас User. Додайте атрибут класу user_count = 0, який відстежує загальну кількість створених користувачів. У конструкторі збільшуйте цей лічильник на 1 при кожному створенні нового об'єкта. Додайте метод класу get_total_users(cls), який повертає поточну кількість користувачів.

```
Вхід: User.get_total_users() → Вихід: 0 → Після u1 = User("Аліса") → User.get_total_users(): 1 → Після u2 = User("Богдан") → User.get_total_users(): 2
Вхід: u1 = User("Марія") → User.get_total_users(): 1
Вхід: User.get_total_users() → 0 → u1 = User("Іван") → u2 = User("Оксана") → u3 = User("Петро") → Вихід: User.get_total_users(): 3
```

Завдання 2.2. Клас "Магазин" зі статичним методом

Створіть клас Store. Додайте статичний метод is_valid_product_name(name), який перевіряє, чи назва товару відповідає критеріям: не порожня, довжина не менше 3 символів, не починається з пробілу. Метод повертає True або False. Конструктор класу приймає name (назва магазину). Додайте метод add_product(self, product_name), який додає товар тільки якщо він валідний (використовуючи статичний метод), і повертає відповідне повідомлення.

```
Вхід: Store.is_valid_product_name("Молоко") → Вихід: True
Вхід: Store.is_valid_product_name(" a") → Вихід: False (пробіл на початку та коротка назва)
Вхід: s = Store("Продукти") → s.add_product("Хліб") → Вихід: "Товар 'Хліб' додано" → s.add_product(" ") → Вихід: "Невірна назва товару"
```

Завдання 2.3. Клас "Дата" з методом класу

Створіть клас MyDate. Конструктор приймає day, month, year. Додайте метод класу from_string(cls, date_string), який приймає рядок у форматі "дд.мм.рррр" і повертає новий об’єкт MyDate. Додайте метод екземпляра format_date(self), який повертає дату у форматі "рррр-мм-дд".

```
Вхід: d1 = MyDate(24, 8, 1991) → Вихід при d1.format_date(): "1991-08-24"
Вхід: d2 = MyDate.from_string("01.12.2023") → Вихід при d2.format_date(): "2023-12-01"
Вхід: d3 = MyDate.from_string("15.05.2000") → Вихід при d3.format_date(): "2000-05-15"
```

Завдання 2.4. Клас "Студент" з оцінками та середнім балом

Розширте клас Student (можна з Завдання 1.4 або створити новий). Додайте атрибут grades (список оцінок, ініціалізується порожнім списком у конструкторі). Додайте методи: add_grade(self, grade) (додає оцінку, перевіряючи, що вона від 1 до 12), get_average(self) (повертає середній бал, округлений до 2 знаків після коми, або 0 якщо оцінок немає), is_excellent(self) (повертає True, якщо середній бал ≥ 10.5).

```
Вхід: s = Student("Ірина", "KN-201") → s.add_grade(12) → s.add_grade(11) → s.add_grade(10) → Вихід при s.get_average(): 11.0 → Вихід при s.is_excellent(): True
Вхід: s = Student("Олег", "KN-202") → s.add_grade(8) → s.add_grade(7) → Вихід: s.get_average(): 7.5 → s.is_excellent(): False
Вхід: s = Student("Марія", "KN-203") → Вихід при s.get_average(): 0.0 → s.add_grade(13) (або s.add_grade(0)) → Вихід: "Оцінка має бути від 1 до 12"
```

Завдання 2.5. Клас "Банківський рахунок" з операціями

Розширте клас Account (можна з Завдання 1.5). Додайте методи: deposit(self, amount) (поповнення, сума має бути додатною), withdraw(self, amount) (зняття, тільки якщо достатньо коштів та сума додатна), transfer(self, other_account, amount) (переказ на інший рахунок). Додайте атрибут класу bank_name = "Український Банк".

Вхід: a1 = Account("Аліса", 500) → a2 = Account("Богдан", 300) → a1.transfer(a2, 200) → Вихід: Баланс a1: 300, Баланс a2: 500
Вхід: a = Account("Іван", 100) → a.deposit(50) → a.withdraw(30) → Вихід: Баланс: 120
Вхід: a = Account("Петро", 50) → a.withdraw(60) → Вихід: "Недостатньо коштів" → a.deposit(-10) → Вихід: "Сума має бути додатною"

Частина 3. Складні завдання

Завдання 3.1. Система "Бібліотека" з взаємодією об’єктів

Створіть клас LibraryBook (книга в бібліотеці) та клас LibraryMember (читач).

LibraryBook має: title, author, book_id, is_available (доступність, за замовчуванням True), методи borrow() (позначає книгу як взяту, якщо вона доступна) та return_book() (позначає як доступну).

LibraryMember має: name, member_id, borrowed_books (список id взятих книг), методи borrow_book(book) (додає id книги до свого списку, якщо книга доступна, та викликає метод borrow() книги) та return_book(book) (видаляє id зі списку та викликає return_book() книги). Додайте метод show_borrowed(self) для виведення списку взятих книг.

Вхід:

```
book1 = LibraryBook("1984", "Орвелл", "B001")
book2 = LibraryBook("Кобзар", "Шевченко", "B002")
member = LibraryMember("Олена", "M001")

member.borrow_book(book1)
member.borrow_book(book2)
member.show_borrowed()
```

Вихід: ["B001", "B002"] (статус book1 та book2: is_available = False)
Вхід: (продовження) member.return_book(book1) → book1.is_available: True → member.show_borrowed(): ["B002"]
Вхід: member.borrow_book(book1) → member.borrow_book(book1) (спроба вдруге) → Вихід: "Книга B001 вже взята вами або недоступна"

Завдання 3.2. Система "Інтернет-магазин"

Створіть класи: Product (товар), CartItem (елемент кошика), ShoppingCart (кошик) та Order (замовлення).

- Product: id, name, price.
- CartItem: посилання на product, quantity (кількість), метод get_total() (ціна * кількість).
- ShoppingCart: список CartItem, методи add_item(product, quantity), remove_item(product_id), get_total_price() (загальна сума), clear().
- Order: генерує order_id, приймає ShoppingCart, має метод place_order() (оформляє замовлення: виводить деталі замовлення, очищає кошик, зберігає статус "оплачено").

```
Вхід:
  p1 = Product("P001", "Ноутбук", 30000)
  p2 = Product("P002", "Мишка", 500)
  cart = ShoppingCart()
  cart.add_item(p1, 1)
  cart.add_item(p2, 2)
  order = Order(cart)
  order.place_order()
Вихід: "Замовлення №...: Ноутбук x1 = 30000 грн, Мишка x2 = 1000 грн. Загалом: 31000 грн. Статус: оплачено" (cart має бути очищений)
Вхід: cart.add_item(p1, 2) → cart.get_total_price(): 60000 → cart.remove_item("P001") → cart.get_total_price(): 0
Вхід: cart.add_item(p2, 5) → cart.add_item(p1, 1) → order = Order(cart) → order.place_order() → cart.get_total_price(): 0
```

Завдання 3.3. Реалізація бінарного дерева пошуку (BST)

Створіть клас TreeNode (вузол дерева) та BinarySearchTree (саме дерево).

- TreeNode: value, left (посилання на лівий дочірній вузол), right (посилання на правий дочірній вузол).
- BinarySearchTree: root (корінь), методи:
  - insert(value): додає значення в дерево за правилами BST (менші – ліворуч, більші – праворуч).
  - search(value): повертає True, якщо значення є в дереві.
  - inorder_traversal(): повертає список значень при обході дерева "лівий-корінь-правий" (значення мають бути відсортовані).
  - find_min(): повертає найменше значення в дереві.

```
Вхід:
  bst = BinarySearchTree()
```

```
  bst.insert(5)
  bst.insert(3)
  bst.insert(7)
  bst.insert(1)
  bst.inorder_traversal()
Вихід: [1, 3, 5, 7]
Вхід: (продовження) bst.search(3): True → bst.search(10): False
Вхід: bst.insert(9) → bst.insert(2) → bst.insert(6) → bst.inorder_traversal(): [1, 2, 3, 5, 6, 7, 9] → bst.find_min(): 1
```

6. Питання для самоперевірки

1. Що таке клас в Python і чим він відрізняється від об’єкта?
2. Для чого використовується метод __init__ у класі? Як він називається?
3. Що таке параметр self у методі класу? Чи можна дати йому інше ім’я?
4. Як створити екземпляр (об’єкт) класу Car? Наведіть приклад.
5. Що таке атрибут екземпляра? Наведіть приклад його ініціалізації в конструкторі та доступу до нього.
6. Що таке метод екземпляра? Наведіть приклад оголошення та виклику.
7. Яка різниця між атрибутом екземпляра та атрибутом класу? Наведіть приклад кожного.
8. Як змінити значення атрибута класу, і як ця зміна вплине на вже створені об’єкти?
9. Що таке метод класу (@classmethod)? Який перший аргумент він приймає і для чого використовується?
10. Що таке статичний метод (@staticmethod)? Чим він відрізняється від методу екземпляра та методу класу? Наведіть ситуацію, коли його доцільно використовувати.
11. Як можна реалізувати альтернативний конструктор для класу? Наведіть короткий приклад.
12. Як метод одного об’єкта може взаємодіяти з іншим об’єктом того ж класу? Наведіть концептуальний приклад.
13. Що таке "магічний метод" __str__? Яку проблему він вирішує?
14. Чи можна додавати нові атрибути екземпляру після його створення, не оголошуючи їх у __init__? Чи є це хорошою практикою?

15. Якщо метод класу змінює атрибут класу, чи вплине це на значення цього атрибута, отримане через self у вже існуючих об’єктах? Поясніть.

16. Як перевірити, чи належить певний об’єкт до конкретного класу?

17. Що станеться, якщо ви спробуєте отримати доступ до атрибута екземпляра, який не був ініціалізований?

18. Як можна імітувати "приватні" атрибути в класі Python (навіть якщо це лише угода)?

19. Для чого може бути корисним метод __repr__ і чим він зазвичай відрізняється від __str__?

20. Описавши клас BankAccount, як ви б реалізували логіку, щоб забороняти створення рахунку з від’ємним початковим балансом?

ЛАБОРАТОРНА РОБОТА №15
Наслідування та поліморфізм

1. Мета

Засвоєння принципів об'єктно-орієнтованого програмування (ООП) – наслідування та поліморфізму, формуванні практичних навичок створення ієрархій класів, перевизначення та розширення методів, використання множинного наслідування, абстрактних класів та забезпечення поліморфної поведінки об'єктів у мові Python.

2. Завдання

1. Оволодіти синтаксисом оголошення батьківського (базового) та дочірніх класів.
2. Навчитися використовувати функцію super() для доступу до методів батьківського класу.
3. Опанувати техніки перевизначення (overriding) та розширення (extending) методів.
4. Зрозуміти принципи множинного наслідування та MRO (Method Resolution Order).
5. Навчитися застосовувати функції isinstance() та issubclass() для перевірки типів.
6. Реалізувати поліморфізм через перевизначення методів у різних класах ієрархії.
7. Ознайомитися з абстрактними базовими класами (ABC) та їх призначенням.
8. Створити власну ієрархію класів для розв'язання конкретної предметної задачі, використовуючи отримані знання.

3. Короткі теоретичні відомості

3.1. Наслідування (Inheritance) – це фундаментальний принцип ООП, що дозволяє створювати новий клас (дочірній, похідний) на основі існуючого (батьківського, базового). Дочірній клас автоматично отримує (наслідує) атрибути та методи батьківського класу, маючи можливість додавати нові або змінювати існуючі. Це сприяє повторному використанню коду та створенню логічних ієрархій.

Синтаксис наслідування в Python

```
class ParentClass:
    # Тіло батьківського класу
    pass

class ChildClass(ParentClass):  # В дужках вказується батьківський клас
    # Тіло дочірнього класу
    pass
```

Використання super()

Функція super() повертає тимчасовий об'єкт батьківського класу, що дозволяє викликати його методи. Це найбезпечніший спосіб розширення функціональності батьківського методу в дочірньому класі, особливо в умовах множинного наслідування.

```
class Animal:
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)      # Викликаємо конструктор Animal
        self.breed = breed
```

Перевизначення методів (Overriding)

Якщо дочірній клас визначає метод з тією ж назвою, що й у батьківського, він *перевизначає* його. При виклику цього методу для об'єкта дочірнього класу виконується нова версія.

```
class Bird:
    def make_sound(self):
        return "Chirp!"

class Duck(Bird):
    def make_sound(self):  # Перевизначення методу
        return "Quack!"
```

Розширення функціональності (Extending)

Поєднання виклику методу батьківського класу за допомогою super() та додавання нової логіки.

```
class Duck(Bird):
    def make_sound(self):
        parent_sound = super().make_sound()  # Отримуємо "Chirp!"
        return f"{parent_sound} But actually: Quack!"
```

Множинне наслідування

Клас може наслідувати від двох або більше батьківських класів. У цьому разі порядок пошуку методів визначається MRO (Method Resolution Order), який можна переглянути за допомогою ClassName.__mro__.

```
class A:
    def show(self):
        print("A")

class B:
    def show(self):
        print("B")

class C(A, B):  # Наслідування від A та B
    pass

obj = C()
obj.show()  # Виведе "A", бо A перший у списку наслідування
print(C.__mro__)  # Покаже порядок пошуку: C -> A -> B -> object
```

Перевірка типів

- isinstance(object, Class) – перевіряє, чи є об’єкт екземпляром зазначеного класу або будь-якого з його нащадків.
- issubclass(ChildClass, ParentClass) – перевіряє, чи є ChildClass підкласом (прямим або непрямим) ParentClass.

3.2. Поліморфізм (Polymorphism) – це здатність об’єктів з різною внутрішньою структурою (різних класів) реагувати на один і той самий виклик методу. У Python поліморфізм часто реалізується через перевизначення методів у ієрархії наслідування.

```
def animal_sound(animal):
    print(animal.make_sound())
bird = Bird()
duck = Duck()
animal_sound(bird)  # Виведе "Chirp!"
animal_sound(duck)  # Виведе "Quack!"
```

Функція animal_sound() поліморфна: вона працює з будь-яким об’єктом, який має метод make_sound().

3.3. Абстрактні базові класи (ABC)

Класи, призначені тільки для наслідування. Вони не призначені для створення екземплярів і можуть містити *абстрактні методи* – методи, оголошені без реалізації, які обов’язково повинні бути реалізовані в дочірньому класі. Для роботи з ABC використовується модуль abc.

```
from abc import ABC, abstractmethod

class Shape(ABC):  # Абстрактний базовий клас

    @abstractmethod
    def area(self):
        """Абстрактний метод, повинен бути реалізований у нащадках."""
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):  # Обов’язкова реалізація абстрактного методу
        return self.width * self.height

# shape = Shape()  # Помилка. Неможливо створити екземпляр абстрактного класу
rect = Rectangle(5, 3)
print(rect.area())  # Працює корректно
```

Використання ABC забезпечує чіткий контракт для всіх нащадків та запобігає помилкам на етапі розробки.

4. Методичні рекомендації

Задача 1. Створення базової ієрархії класів "Транспорт"

Створіть базовий клас Vehicle (Транспорт) з атрибутами brand (марка) та max_speed (максимальна швидкість). Створіть два дочірніх класи: Car (Автомобіль), який додає атрибут num_doors (кількість дверей), та Motorcycle (Мотоцикл), який додає атрибут has_sidecar

(наявність коляски). У всіх класах реалізуйте метод display_info(), який виводить інформацію про транспортний засіб.

```
    class Vehicle:
        """Базовий клас для всіх транспортних засобів."""

        def __init__(self, brand, max_speed):
            """Ініціалізація базових атрибутів."""
            self.brand = brand
            self.max_speed = max_speed

        def display_info(self):
            """Виводить базову інформацію про транспорт."""
            return f"Brand: {self.brand}, Max Speed:
{self.max_speed} km/h"

    class Car(Vehicle):
        """Клас автомобіль, наслідує Vehicle."""

        def __init__(self, brand, max_speed, num_doors):
            """Ініціалізація з використанням super()."""
            super().__init__(brand, max_speed)
            self.num_doors = num_doors

        def display_info(self):
            """Перевизначення методу з розширенням."""
            base_info = super().display_info()
            return f"{base_info}, Doors: {self.num_doors}, Type: Car"

    class Motorcycle(Vehicle):
        """Клас мотоцикл, наслідує Vehicle."""
        def __init__(self, brand, max_speed, has_sidecar):
            """Ініціалізація з використанням super()."""
            super().__init__(brand, max_speed)
            self.has_sidecar = has_sidecar

        def display_info(self):
            """Перевизначення методу з розширенням."""
            base_info = super().display_info()
            sidecar_info = "with sidecar" if self.has_sidecar else
"without sidecar"
            return f"{base_info}, {sidecar_info}, Type:
Motorcycle"

    # Демонстрація роботи
```

```
    vehicles = [
        Car("Toyota", 200, 5),
        Motorcycle("Harley-Davidson", 180, False),
        Car("BMW", 250, 3),
        Motorcycle("Ural", 120, True)
    ]
    for vehicle in vehicles:
        print(vehicle.display_info())
```

Приклад вхідних та вихідних даних:

Вхід: Car("Toyota", 200, 5)
Вихід: Brand: Toyota, Max Speed: 200 km/h, Doors: 5, Type: Car

Вхід: Motorcycle("Harley-Davidson", 180, False)
Вихід: Brand: Harley-Davidson, Max Speed: 180 km/h, without sidecar, Type: Motorcycle

Вхід: Motorcycle("Ural", 120, True)
Вихід: Brand: Ural, Max Speed: 120 km/h, with sidecar, Type: Motorcycle

Коментарі:

Ця задача демонструє базове наслідування, використання super() для виклику конструктора батьківського класу, перевизначення методів та їх розширення. Класи Car і Motorcycle наслідують спільну функціональність від Vehicle і додають власні особливості.

Задача 2. Використання поліморфізму для обчислення площ фігур

Створіть абстрактний базовий клас Shape (Фігура) з абстрактним методом area(). Створіть три дочірніх класи: Rectangle (Прямокутник), Circle (Коло) та Triangle (Трикутник). Кожен клас повинен мати відповідні атрибути (довжина, ширина; радіус; сторона та висота) та реалізовувати метод area(). Створіть список різних фігур та обчисліть загальну площу всіх фігур.

```
    from abc import ABC, abstractmethod
    import math

    class Shape(ABC):
        """Абстрактний базовий клас для геометричних фігур."""
```

```
    @abstractmethod
    def area(self):
        """Абстрактний метод для обчислення площі."""
        pass

class Rectangle(Shape):
    """Клас прямокутник."""

    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        """Обчислення площі прямокутника."""
        return self.width * self.height

class Circle(Shape):
    """Клас коло."""

    def __init__(self, radius):
        self.radius = radius

    def area(self):
        """Обчислення площі кола."""
        return math.pi * self.radius ** 2

class Triangle(Shape):
    """Клас трикутник."""

    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        """Обчислення площі трикутника."""
        return 0.5 * self.base * self.height

# Демонстрація поліморфізму
def calculate_total_area(shapes):
    """Обчислює загальну площу списку фігур."""
    total = 0
    for shape in shapes:
        total += shape.area()  # Поліморфний виклик
    return total
```

```
# Створення різних фігур
shapes = [
    Rectangle(4, 5),
    Circle(3),
    Triangle(6, 4),
    Rectangle(2, 3),
    Circle(2.5)
]

# Обчислення площ кожної фігури
print("Areas of individual shapes:")
for i, shape in enumerate(shapes, 1):
    print(f"Shape {i}: {shape.area():.2f}")

# Обчислення загальної площі
total_area = calculate_total_area(shapes)
print(f"\nTotal area of all shapes: {total_area:.2f}")
```

Приклад вхідних та вихідних даних:

```
Вхід: shapes = [Rectangle(4, 5), Circle(3), Triangle(6, 4)]
Вихід:
Shape 1: 20.00
Shape 2: 28.27
Shape 3: 12.00
Total area of all shapes: 60.27

Вхід: shapes = [Circle(2.5), Rectangle(2, 3)]
Вихід:
Shape 1: 19.63
Shape 2: 6.00
Total area of all shapes: 25.63
```

Коментарі:

Задача демонструє концепцію поліморфізму через абстрактні базові класи. Функція calculate_total_area() працює з будь-якими об’єктами, які мають метод area(), незалежно від їх конкретного типу. ABC гарантує, що всі нащадки реалізують необхідний метод.

Задача 3. Множинне наслідування: створення класу "Смартфон"

Створіть два базові класи: Camera (з методом take_photo()) та Phone (з методом make_call()). Створіть клас Smartphone, який наслідує від

обох цих класів. Додайте метод use_app(), унікальний для смартфона. Продемонструйте порядок Рішення методів (MRO).

```
    class Camera:
        """Клас, що представляє камеру."""

        def __init__(self, megapixels):
            self.megapixels = megapixels

        def take_photo(self):
            """Метод для фотографування."""
            return f"Taking photo with {self.megapixels}MP camera"

        def common_method(self):
            """Метод для демонстрації MRO."""
            return "Method from Camera class"
    class Phone:
        """Клас, що представляє телефон."""

        def __init__(self, number):
            self.number = number

        def make_call(self, contact):
            """Метод для здійснення дзвінка."""
            return f"Calling {contact} from {self.number}"

        def common_method(self):
            """Метод для демонстрації MRO."""
            return "Method from Phone class"

    class Smartphone(Camera, Phone):
        """Клас смартфон, наслідує від Camera та Phone."""

        def __init__(self, megapixels, number, os):
            """Ініціалізація з використанням super()."""
            # При множинному наслідуванні super() працює з першим
класом у MRO
            Camera.__init__(self, megapixels)
            Phone.__init__(self, number)
            self.os = os

        def use_app(self, app_name):
            """Унікальний метод смартфону."""
            return f"Using {app_name} on {self.os}"
```

```
        def show_all_features(self):
            """Демонстрація всіх функцій смартфону."""
            features = [
                self.take_photo(),
                self.make_call("Mom"),
                self.use_app("Browser")
            ]
            return features

    # Створення екземпляра смартфону
    smartphone = Smartphone(48, "+380991234567", "Android")

    # Демонстрація функціоналу
    print("Smartphone features:")
    for feature in smartphone.show_all_features():
        print(f"- {feature}")

    # Демонстрація MRO
    print(f"\nMRO for Smartphone: {[cls.__name__ for cls in
Smartphone.__mro__]}")

    # Демонстрація виклику спільного методу
    print(f"\nCalling common_method():
{smartphone.common_method()}")
    print(f"Explanation: It calls from
{Smartphone.__mro__[1].__name__} because of MRO")
```

Приклад вхідних та вихідних даних:

```
Вхід: smartphone = Smartphone(48, "+380991234567", "Android")
Вихід:
Smartphone features:
- Taking photo with 48MP camera
- Calling Mom from +380991234567
- Using Browser on Android

MRO for Smartphone: ['Smartphone', 'Camera', 'Phone', 'object']
Calling common_method(): Method from Camera class
```

Коментарі:

Задача ілюструє множинне наслідування. Важливо розуміти MRO (Method Resolution Order), який визначає порядок пошуку методів. У даному випадку, оскільки Camera вказано першим, його методи мають пріоритет. Для явного виклику методів конкретного батьківського класу використовується синтаксис ClassName.method(self, ...).

Задача 4. Перевірка типів за допомогою isinstance() та issubclass()

Створіть ієрархію класів: Person → Employee → Manager. Клас Person має атрибут name, Employee додає salary, Manager додає department. Напишіть функцію, яка приймає список об'єктів різних типів та:

1. Виводить імена всіх осіб
2. Підраховує загальну зарплату всіх співробітників (Employee та Manager)
3. Виводить кількість менеджерів
4. Перевіряє відносини між класами

```
class Person:
    """Базовий клас для особи."""

    def __init__(self, name):
        self.name = name

    def display_info(self):
        return f"Person: {self.name}"

class Employee(Person):
    """Клас співробітник, наслідує Person."""

    def __init__(self, name, salary):
        super().__init__(name)
        self.salary = salary

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Salary: {self.salary}"

class Manager(Employee):
    """Клас менеджер, наслідує Employee."""

    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}, Department: {self.department}"
```

```
    def process_people(people_list):
        """Обробляє список людей з використанням isinstance()."""

        total_salary = 0
        manager_count = 0

        print("=== Information about all people ===")
        for person in people_list:
            # 1. Виводимо інформацію про всіх
            print(f"- {person.display_info()}")

            # 2. Підраховуємо зарплату співробітників
            if isinstance(person, Employee):  # Працює для
Employee та Manager
                total_salary += person.salary

            # 3. Підраховуємо менеджерів
            if isinstance(person, Manager):
                manager_count += 1

        print(f"\n=== Statistics ===")
        print(f"Total salary of all employees: {total_salary}")
        print(f"Number of managers: {manager_count}")
        print(f"Total people processed: {len(people_list)}")

    # Демонстраційні дані
    people = [
        Person("John Doe"),
        Employee("Alice Smith", 50000),
        Manager("Bob Johnson", 80000, "IT"),
        Employee("Carol Williams", 45000),
        Manager("David Brown", 90000, "Sales"),
        Person("Eva Davis")
    ]
    # Обробка списку
    process_people(people)

    # Демонстрація issubclass()
    print("\n=== Class relationships ===")
    print(f"Is Manager a subclass of Employee?
{issubclass(Manager, Employee)}")
    print(f"Is Employee a subclass of Person?
{issubclass(Employee, Person)}")
```

```
    print(f"Is Manager a subclass of Person? {issubclass(Manager,
Person)}")
    print(f"Is Person a subclass of Employee? {issubclass(Person,
Employee)}")

    # Демонстрація isinstance() з конкретними об'єктами
    print("\n=== Type checking for specific objects ===")
    alice = people[1]
    bob = people[2]

    print(f"Is Alice an Employee? {isinstance(alice, Employee)}")
    print(f"Is Alice a Manager? {isinstance(alice, Manager)}")
    print(f"Is Alice a Person? {isinstance(alice, Person)}")
    print(f"Is Bob both Employee and Manager? {isinstance(bob,
Employee) and isinstance(bob, Manager)}")
```

Приклад вхідних та вихідних даних:

```
Вхід: people = [Person("John"), Employee("Alice", 50000),
Manager("Bob", 80000, "IT")]
Вихід:
=== Information about all people ===
- Person: John
- Person: Alice, Salary: 50000
- Person: Bob, Salary: 80000, Department: IT

=== Statistics ===
Total salary of all employees: 130000
Number of managers: 1
Total people processed: 3
=== Class relationships ===
Is Manager a subclass of Employee? True
Is Employee a subclass of Person? True
Is Manager a subclass of Person? True
Is Person a subclass of Employee? False
```

Коментарі:

Задача демонструє практичне використання isinstance() та issubclass(). isinstance() перевіряє, чи належить об'єкт до певного класу або його нащадків, що важливо при роботі з ієрархіями. issubclass() перевіряє відносини між класами. Ці функції дозволяють писати гнучкий код, який коректно працює з об'єктами різних рівнів ієрархії.

Задача 5. Розширення функціональності через super() у глибокій ієрархії

Створіть ієрархію класів для представлення персоналу університету: UniversityMember → Student → GraduateStudent та UniversityMember → Faculty → Professor. Продемонструйте використання super() для послідовного розширення методів у глибоких ієрархіях.

```
class UniversityMember:
    """Базовий клас для всіх членів університету."""

    def __init__(self, name, member_id):
        self.name = name
        self.member_id = member_id
        self.university = "National University"

    def get_info(self):
        """Базова інформація про члена університету."""
        return f"{self.name} (ID: {self.member_id})"

    def role(self):
        """Повертає роль у університеті."""
        return "University Member"

    def full_description(self):
        """Повний опис члена університету."""
       return f"{self.get_info()} - {self.role()} at {self.university}"
class Student(UniversityMember):
    """Клас студента."""

    def __init__(self, name, member_id, major):
        super().__init__(name, member_id)
        self.major = major
        self.courses = []

    def enroll_course(self, course):
        """Запис на курс."""
        self.courses.append(course)
        return f"{self.name} enrolled in {course}"

    def role(self):
        """Перевизначення ролі."""
        return "Student"

    def get_info(self):
```

```
            """Розширення базової інформації."""
            base_info = super().get_info()
            return f"{base_info}, Major: {self.major}"

        def full_description(self):
            """Розширення повного опису."""
            base_desc = super().full_description()
            courses_info = f", Courses: {', '.join(self.courses)}"
if self.courses else ""
            return f"{base_desc}{courses_info}"

    class GraduateStudent(Student):
        """Клас випускника (магістра/аспіранта)."""

        def __init__(self, name, member_id, major,
research_topic):
            super().__init__(name, member_id, major)
            self.research_topic = research_topic
            self.thesis_progress = 0

        def role(self):
            """Перевизначення ролі."""
            return "Graduate Student"

        def update_thesis_progress(self, progress):
            """Оновлення прогресу дисертації."""
            self.thesis_progress = min(100, max(0,
self.thesis_progress + progress))
            return f"Thesis progress: {self.thesis_progress}%"

        def get_info(self):
            """Розширення інформації."""
            base_info = super().get_info()
            return f"{base_info}, Research: {self.research_topic}"

        def full_description(self):
            """Розширення повного опису."""
            base_desc = super().full_description()
            return f"{base_desc}, Thesis progress: {self.thesis_progress}%"

    class Faculty(UniversityMember):
        """Клас викладача."""

        def __init__(self, name, member_id, department):
```

```
        super().__init__(name, member_id)
        self.department = department

    def role(self):
        """Перевизначення ролі."""
        return "Faculty"

    def get_info(self):
        """Розширення інформації."""
        base_info = super().get_info()
        return f"{base_info}, Department: {self.department}"

class Professor(Faculty):
    """Клас професора."""

    def __init__(self, name, member_id, department, rank):
        super().__init__(name, member_id, department)
        self.rank = rank
        self.research_grants = []

    def role(self):
        """Перевизначення ролі."""
        return f"Professor ({self.rank})"

    def add_grant(self, grant_name, amount):
        """Додавання гранту на дослідження."""
        self.research_grants.append((grant_name, amount))
        return f"Added grant: {grant_name} (${amount})"

    def get_info(self):
        """Розширення інформації."""
        base_info = super().get_info()
        return f"{base_info}, Rank: {self.rank}"

    def full_description(self):
        """Розширення повного опису."""
        base_desc = super().full_description()
        grants_count = len(self.research_grants)
        return f"{base_desc}, Research grants: {grants_count}"

# Демонстрація роботи
print("=== University Members ===")

# Створення об'єктів різних рівнів ієрархії
```

```
    members = [
        Student("Anna Ivanova", "S1001", "Computer Science"),
        GraduateStudent("Petro Sidorov", "GS2001", "Physics",
"Quantum Computing"),
        Faculty("Maria Kovalenko", "F3001", "Mathematics"),
        Professor("Ivan Petrenko", "P4001", "Computer Science",
"Full Professor")
    ]

    # Демонстрація ланцюжкового виклику через super()
    for member in members:
        print(f"\n{member.full_description()}")

    # Демонстрація специфічних методів
    print("\n=== Specialized Actions ===")

    # Студент записується на курси
    student = members[0]
    print(student.enroll_course("Algorithms"))
    print(student.enroll_course("Data Structures"))

    # Випускник працює над дисертацією
    grad_student = members[1]
    print(grad_student.update_thesis_progress(30))
    print(grad_student.update_thesis_progress(25))

    # Професор отримує грант
    professor = members[3]
    print(professor.add_grant("AI Research", 50000))
    print(professor.add_grant("Quantum Algorithms", 75000))

    # Показуємо оновлену інформацію
    print(f"\nUpdated info for {student.name}:
{student.full_description()}")
    print(f"Updated info for {grad_student.name}:
{grad_student.full_description()}")
    print(f"Updated info for {professor.name}:
{professor.full_description()}")
```

Приклад вхідних та вихідних даних:

```
Вхід: members = [Student("Anna", "S1001", "CS"),
GraduateStudent("Petro", "GS2001", "Physics", "Quantum")]
Вихід:
=== University Members ===
```

```
Anna Ivanova (ID: S1001), Major: Computer Science - Student at National University
Petro Sidorov (ID: GS2001), Major: Physics, Research: Quantum Computing - Graduate Student at National University, Thesis progress: 0%

=== Specialized Actions ===
Anna Ivanova enrolled in Algorithms
Anna Ivanova enrolled in Data Structures
Thesis progress: 30%
Thesis progress: 55%
Added grant: AI Research ($50000)
Added grant: Quantum Algorithms ($75000)

Updated info for Anna Ivanova: Anna Ivanova (ID: S1001), Major: Computer Science - Student at National University, Courses: Algorithms, Data Structures
```

Коментарі:

Задача демонструє потужність super() для створення глибоких ієрархій класів. Кожен рівень ієрархії розширює функціональність попереднього, не повторюючи код. Метод full_description() показує, як через ланцюжок викликів super().full_description() можна поступово накопичувати інформацію від базового до похідних класів.

Типові помилки і шляхи їх усунення

1. **Забули викликати super().__init__() у конструкторі дочірнього класу**

Проблема. Атрибути батьківського класу не ініціалізуються
Рішення. Завжди викликайте super().__init__(...) першим у конструкторі дочірнього класу

```python
# Неправильно:
class Child(Parent):
    def __init__(self, x, y):
        self.y = y  # Забули викликати super()

# Правильно:
class Child(Parent):
    def __init__(self, x, y):
        super().__init__(x)  # Викликаємо першим
        self.y = y
```

2. **Неправильний порядок аргументів при виклику super()**

Проблема. TypeError або некоректна ініціалізація
Рішення. Передавайте аргументи у тому порядку, в якому їх очікує конструктор батьківського класу

```
class Parent:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Child(Parent):
    def __init__(self, name, age, grade):
        super().__init__(name, age)  # Правильний порядок
        self.grade = grade
```

3. **Плутанина з MRO при множинному наслідуванні**
Проблема. Викликається не той метод з батьківських класів
Рішення. Перевіряйте MRO за допомогою ClassName.__mro__. Для явного виклику використовуйте ParentClass.method(self, ...)

```
class A:
    def method(self):
        return "A"

class B:
    def method(self):
        return "B"

class C(A, B):
    def call_both(self):
        print(super().method())   # Викличе метод A
        print(B.method(self))     # Явний виклик методу B
```

4. **Створення екземпляра абстрактного класу**
Проблема. TypeError: Can't instantiate abstract class ... with abstract methods ...
Рішення. Абстрактні класи призначені тільки для наслідування. Створюйте екземпляри дочірніх класів, які реалізують усі абстрактні методи

```
from abc import ABC, abstractmethod

class AbstractClass(ABC):
    @abstractmethod
    def must_implement(self):
        pass
```

```
# Неправильно:
# obj = AbstractClass()  # Помилка!

# Правильно:
class ConcreteClass(AbstractClass):
    def must_implement(self):
        return "Implemented"

obj = ConcreteClass()  # Працює
```

5. **Неперевизначення абстрактних методів**

   Проблема. TypeError при спробі створення екземпляра класу
   Рішення. Усі абстрактні методи базового класу повинні бути реалізовані у дочірньому класі

```
class AbstractClass(ABC):
    @abstractmethod
    def method1(self): pass
    @abstractmethod
    def method2(self): pass

# Неправильно (реалізовано тільки один метод):
class Child(AbstractClass):
    def method1(self): return "OK"
    # method2 не реалізовано!

# Правильно:
class Child(AbstractClass):
    def method1(self): return "OK"
    def method2(self): return "Also OK"  # Всі методи реалізовані
```

Корисні поради

1. **Плануйте ієрархію заздалегідь.** Перш ніж писати код, намалюйте схему класів та їх відносин. Це допоможе уникнути плутанини з наслідуванням та MRO.

2. **Використовуйте композицію замість наслідування.** Якщо клас потрібно "наділити" певною функціональністю, іноді краще

використати композицію (включення об’єкта як атрибута), а не наслідування. Наслідування створює тісний зв’язок між класами.

3. **Принцип "is-a" vs "has-a".** Наслідування слід використовувати лише тоді, коли між класами існує відношення "is-a" (є). Наприклад, "Студент є Людиною". Якщо відношення "has-a" (має), краще використовувати композицію.

4. **Дотримуйтесь DRY (Don’t Repeat Yourself).** Якщо бачите однаковий код у кількох класах, ймовірно, його варто винести в базовий клас.

5. **Тестуйте поліморфізм.** При створенні поліморфних методів переконайтесь, що вони коректно працюють з об’єктами різних класів ієрархії.

6. **Документуйте абстрактні методи.** Для абстрактних методів обов’язково пишіть докстрінги, що пояснюють, що повинен робити метод та які параметри очікуються.

7. **Використовуйте type hints.** Для кращої читабельності та підтримки IDE додавайте анотації типів, особливо при роботі з ієрархіями класів.

8. **Перевіряйте типи обережно.** Замість перевірки конкретного типу (type(obj) == Class), використовуйте isinstance(), який враховує наслідування.

9. **Уникайте занадто глибоких ієрархій.** Глибина наслідування більше 3-4 рівнів часто ускладнює розуміння та підтримку коду.

10. **Вивчайте MRO для складних випадків.** При множинному наслідуванні завжди перевіряйте __mro__, щоб зрозуміти порядок пошуку методів.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Створення базового класу "Тварина"

Створіть базовий клас Animal з атрибутами name (ім'я) та species (вид). Додайте метод make_sound() та get_info(). Створіть дочірній клас Dog, який наслідує Animal та перевизначає метод make_sound() для повернення "Гав!". Використовуйте super() для виклику батьківського конструктора.

```
Вхід: dog = Dog("Рекс", "Собака домашній")
Вихід при виклику dog.get_info(): "Рекс - Собака домашній"
Вихід при виклику dog.make_sound(): "Гав!"

Вхід: dog2 = Dog("Бакс", "Лабродор")
Вихід при виклику dog2.get_info(): "Бакс - Лабродор"

Вхід: dog3 = Dog("Шарик", "Вівчарка")
Вихід при виклику dog3.make_sound(): "Гав!"
```

Завдання 1.2. Розширення класу "Студент"

Створіть клас Student з атрибутами name та student_id. Створіть клас ExtendedStudent, який наслідує Student та додає атрибут major (спеціальність). Реалізуйте метод display_info(), який у дочірньому класі повинен показувати всю інформацію про студента.

```
Вхід: s = ExtendedStudent("Олена Петренко", "S12345", "Комп'ютерні науки")
Вихід: s.display_info() → "Студент: Олена Петренко, ID: S12345, Спеціальність: Комп'ютерні науки"

Вхід: s2 = ExtendedStudent("Іван Сидоренко", "S67890", "Математика")
Вихід: s2.display_info() → "Студент: Іван Сидоренко, ID: S67890, Спеціальність: Математика"

Вхід: s3 = ExtendedStudent("Марія Коваленко", "S11111", "Фізика")
Вихід: s3.display_info() → "Студент: Марія Коваленко, ID: S11111, Спеціальність: Фізика"
```

Завдання 1.3. Наслідування класу "Банківський рахунок"

Створіть базовий клас BankAccount з атрибутами owner (власник) та balance (баланс). Додайте методи deposit(amount) (поповнити) та get_balance(). Створіть клас SavingsAccount, який наслідує BankAccount та додає атрибут interest_rate (відсоткова ставка). Використовуйте super() у конструкторі.

```
Вхід: acc = SavingsAccount("Петро Іванов", 1000, 0.05)
acc.deposit(500)
Вихід: acc.get_balance() → 1500

Вхід: acc2 = SavingsAccount("Оксана Сидорова", 2000, 0.03)
Вихід: acc2.get_balance() → 2000

Вхід: acc3 = SavingsAccount("Михайло Петренко", 0, 0.07)
acc3.deposit(3000)
Вихід: acc3.get_balance() → 3000
```

Завдання 1.4. Ієрархія "Фігура → Прямокутник"

Створіть клас Shape з методом area() (повертає 0). Створіть клас Rectangle, який наслідує Shape та додає атрибути width та height. Перевизначте метод area() для обчислення площі прямокутника.

```
Вхід: r = Rectangle(4, 5)
Вихід: r.area() → 20

Вхід: r2 = Rectangle(10, 3)
Вихід: r2.area() → 30

Вхід: r3 = Rectangle(7, 7)
Вихід: r3.area() → 49
```

Завдання 1.5. Клас "Книга" та "Електронна книга"

Створіть клас Book з атрибутами title, author та pages. Створіть клас Ebook, який наслідує Book та додає атрибут file_size (розмір файлу в МБ). Перевизначте метод get_info() для відображення всіх даних.

```
Вхід: book = Ebook("Python для початківців", "Іван Петренко",
300, 2.5)
Вихід: book.get_info() → "Python для початківців, автор: Іван
Петренко, 300 стор., 2.5 МБ"
Вхід: book2 = Ebook("Алгоритми та структури даних", "Олена
Сидорова", 450, 3.2)
Вихід: book2.get_info() → "Алгоритми та структури даних,
автор: Олена Сидорова, 450 стор., 3.2 МБ"
```

```
    Вхід: book3 = Ebook("Основи програмування", "Марія Коваленко",
200, 1.8)
    Вихід: book3.get_info() → "Основи програмування, автор: Марія
Коваленко, 200 стор., 1.8 МБ"
```

Завдання 1.6. Перевизначення методу "Привітання"

Створіть базовий клас Greeter з методом greet(name), який повертає "Привіт, [name]!". Створіть два дочірніх класи: FormalGreeter (повертає "Доброго дня, [name]!") та CasualGreeter (повертає "Привіт, [name]! Як справи?"). Продемонструйте перевизначення методів.

```
    Вхід: g1 = FormalGreeter()
    Вихід: g1.greet("Олена") → "Доброго дня, Олена!"

    Вхід: g2 = CasualGreeter()
    Вихід: g2.greet("Іван") → "Привіт, Іван! Як справи?"

    Вхід: g3 = FormalGreeter()
    Вихід: g3.greet("Марія") → "Доброго дня, Марія!"
```

Завдання 1.7. Класи "Транспорт" з різними звуками

Створіть базовий клас Vehicle з методом make_sound() (повертає "Транспортний засіб видає звук"). Створіть три дочірніх класи: Car ("Біп-біп!"), Bicycle ("Дзень-дзень!"), Train ("Ту-ту!"). Перевизначте метод у кожному класі.

```
    Вхід: v1 = Car()
    Вихід: v1.make_sound() → "Біп-біп!"

    Вхід: v2 = Bicycle()
    Вихід: v2.make_sound() → "Дзень-дзень!"

    Вхід: v3 = Train()
    Вихід: v3.make_sound() → "Ту-ту!"
```

Завдання 1.8. Перевизначення методу для обчислення зарплати

Створіть клас Employee з атрибутами name та base_salary (базова зарплата). Метод calculate_salary() повертає base_salary. Створіть клас Manager, який наслідує Employee та додає атрибут bonus. Перевизначте calculate_salary() для повернення base_salary + bonus.

```
    Вхід: e = Manager("Петро Сидоренко", 20000, 5000)
```

```
Вихід: e.calculate_salary() → 25000

Вхід: e2 = Manager("Олена Іваненко", 25000, 7000)
Вихід: e2.calculate_salary() → 32000

Вхід: e3 = Manager("Іван Петренко", 18000, 3000)
Вихід: e3.calculate_salary() → 21000
```

Завдання 1.9. Класи "Співробітник" з різними ролями

Створіть клас Person з методом get_role() (повертає "Персона"). Створіть класи Student ("Студент"), Teacher ("Викладач"), Admin ("Адміністратор"), які наслідують Person та перевизначають метод get_role().

```
Вхід: p1 = Student()
Вихід: p1.get_role() → "Студент"

Вхід: p2 = Teacher()
Вихід: p2.get_role() → "Викладач"

Вхід: p3 = Admin()
Вихід: p3.get_role() → "Адміністратор"
```

Завдання 1.10. Перевизначення toString-методу

Створіть клас Product з атрибутами name та price. Перевизначте метод __str__() для повернення "Товар: [name], Ціна: [price] грн.". Створіть клас DiscountedProduct, який наслідує Product та додає атрибут discount. Перевизначте __str__() для відображення ціни зі знижкою.

```
Вхід: p = Product("Монітор", 5000)
Вихід: print(p) → "Товар: Монітор, Ціна: 5000 грн."

Вхід: p2 = DiscountedProduct("Клавіатура", 1000, 20)
Вихід: print(p2) → "Товар: Клавіатура, Ціна: 800 грн. (знижка 20%)"

Вхід: p3 = DiscountedProduct("Мишка", 500, 10)
Вихід: print(p3) → "Товар: Мишка, Ціна: 450 грн. (знижка 10%)"
```

Частина 2. Середні завдання

Завдання 2.1. Множинне наслідування: "Розумний будинок"

Створіть три базові класи: LightDevice (метод turn_on(), turn_off()), Thermostat (метод set_temperature(temp)), SecurityDevice (метод activate_alarm()). Створіть клас SmartHomeController, який наслідує всі три класи. Додайте метод evening_mode(), який включає світло, встановлює температуру 22°C та активує сигналізацію. Продемонструйте MRO.

```
Вхід: controller = SmartHomeController()
Вихід при виклику controller.evening_mode() →
"Світло увімкнено"
"Температура встановлена на 22°C"
"Сигналізацію активовано"

Вхід: controller.turn_off() → "Світло вимкнено"
Вхід: controller.set_temperature(25) → "Температура встановлена на 25°C"
Вихід: SmartHomeController.__mro__ → показує порядок пошуку методів
```

Завдання 2.2. Поліморфізм у системі обробки документів

Створіть базовий клас Document з абстрактним методом process(). Створіть три дочірніх класи: TextDocument (повертає "Обробка текстового документу"), SpreadsheetDocument (повертає "Обробка електронної таблиці"), PDFDocument (повертає "Обробка PDF-документу"). Напишіть функцію process_documents(documents), яка обробляє список документів різних типів.

```
Вхід: docs = [TextDocument(), SpreadsheetDocument(), PDFDocument()]
Вихід process_documents(docs):
"Обробка текстового документу"
"Обробка електронної таблиці"
"Обробка PDF-документу"

Вхід: docs2 = [PDFDocument(), PDFDocument(), TextDocument()]
Вихід:
"Обробка PDF-документу"
"Обробка PDF-документу"
"Обробка текстового документу"

Вхід: docs3 = [SpreadsheetDocument(), TextDocument()]
Вихід:
"Обробка електронної таблиці"
"Обробка текстового документу"
```

Завдання 2.3. Система платіжних методів з поліморфізмом

Створіть базовий клас PaymentMethod з методом process_payment(amount). Створіть три дочірніх класи: CreditCard ("Оплата карткою: [amount] грн."), PayPal ("Оплата через PayPal: [amount] грн."), BankTransfer ("Банківський переказ: [amount] грн."). Напишіть функцію checkout(payment_method, amount), яка приймає будь-який платіжний метод.

```
Вхід: card = CreditCard()
checkout(card, 1500) → "Оплата карткою: 1500 грн."

Вхід: paypal = PayPal()
checkout(paypal, 750) → "Оплата через PayPal: 750 грн."

Вхід: transfer = BankTransfer()
checkout(transfer, 3000) → "Банківський переказ: 3000 грн."
```

Завдання 2.4. Практична ієрархія: "Бібліотечна система"

Створіть ієрархію класів для бібліотеки: LibraryItem (атрибути: title, item_id, available; методи: check_out(), return_item()) → Book (додає author, pages) → Magazine (додає issue_number, month) → DVD (додає duration, director). Реалізуйте поліморфізм для методу get_details().

```
Вхід: items = [
    Book("Python Programming", "B001", "John Doe", 400),
    Magazine("Tech Today", "M001", 15, "Січень"),
    DVD("Python Tutorial", "D001", 120, "Jane Smith")
]

Вихід для item.get_details():
Book: "Python Programming" by John Doe, 400 pages
Magazine: "Tech Today", Issue 15, Січень
DVD: "Python Tutorial", 120 min, director: Jane Smith

Вхід: book.check_out() → "Книгу 'Python Programming' видано"
Вхід: book.return_item() → "Книгу 'Python Programming' повернено"
```

Завдання 2.5. Система замовлень з перевіркою типів

Створіть класи Product (атрибути: name, price), OrderItem (атрибути: product, quantity; метод: get_total()), Order (атрибути: order_id, items; методи: add_item(item), get_order_total()). Використовуйте isinstance() для перевірки, що до замовлення додаються тільки об'єкти типу

OrderItem. Реалізуйте метод apply_discount(discount_percent) у класі Order.

```
Вхід:
p1 = Product("Монітор", 5000)
p2 = Product("Клавіатура", 800)
item1 = OrderItem(p1, 2)  # 2 монітори
item2 = OrderItem(p2, 1)  # 1 клавіатура
order = Order("ORD001")
order.add_item(item1)
order.add_item(item2)

Вихід: order.get_order_total() → 10800  # 2*5000 + 800
Вихід після order.apply_discount(10) → 9720  # знижка 10%

Вхід: order.add_item("неправильний об'єкт") → "Помилка. можна додавати тільки OrderItem"

Вхід: order2 = Order("ORD002")
order2.add_item(OrderItem(Product("Мишка", 500), 3))
Вихід: order2.get_order_total() → 1500
```

Частина 3. Складні завдання

Завдання 3.1. Система управління персоналом з абстрактними класами

Створіть абстрактний базовий клас Employee з абстрактними методами calculate_salary() та get_role(). Створіть три конкретних класи: HourlyEmployee (атрибути: hours_worked, hourly_rate), SalariedEmployee (атрибути: monthly_salary, bonus), CommissionEmployee (атрибути: sales_amount, commission_rate). Створіть клас Department для управління співробітниками різних типів. Реалізуйте функціонал додавання співробітників, підрахунку загальних витрат на зарплату та пошуку співробітників за роллю.

```
Вхід:
dept = Department("IT")
dept.add_employee(HourlyEmployee("Іван", "розробник", 160, 250))
dept.add_employee(SalariedEmployee("Олена", "менеджер", 30000, 5000))
dept.add_employee(CommissionEmployee("Петро", "продажі", 500000, 0.05))

Вихід: dept.total_payroll() → сума всіх зарплат
Вихід: dept.find_by_role("розробник") → список співробітників з цією роллю
```

```
    Вихід для кожного співробітника: employee.calculate_salary() →
конкретна зарплата

    Вхід:
    dept2 = Department("Sales")
    dept2.add_employee(CommissionEmployee("Марія",     "продажник",
1000000, 0.07))
    Вихід: dept2.total_payroll() → 70000 (7% від 1 000 000)
```

Завдання 3.2. Симулятор екосистеми з множинним наслідуванням

Створіть систему класів для симуляції екосистеми. Базові класи: LivingOrganism (методи: eat(), reproduce()), Mobile (методи: move(), get_position()), Sensed (методи: see(), hear()). Створіть конкретні класи тварин з множинним наслідуванням: Bird (наслідує LivingOrganism, Mobile, Sensed; додає метод fly()), Fish (наслідує LivingOrganism, Mobile; додає метод swim()), Mammal (наслідує LivingOrganism, Mobile, Sensed; додає метод run()). Реалізуйте систему взаємодії між організмами.

```
    Вхід:
    eagle = Bird("орел", position=(0, 0))
    salmon = Fish("лосось", position=(10, 5))
    lion = Mammal("лев", position=(5, 5))

    Вихід: eagle.move((2, 3)) → "орел перемістився до (2, 3)"
    Вихід: salmon.swim() → "лосось пливе"
    Вихід: lion.see(eagle) → "лев бачить орла на відстані ..."
    Вихід: eagle.fly() → "орел летить"
    Вихід: lion.eat() → "лев їсть"
    Вихід: salmon.reproduce() → "лосось розмножується"

    Симуляція:
    ecosystem = [eagle, salmon, lion]
    for day in range(3):
        for org in ecosystem:
            org.move(random_position())
            org.eat()
```

Завдання 3.3. Система управління навчальним курсом з повною функціональністю

Створіть повну систему для управління навчальним курсом. Абстрактні класи: CourseItem (атрибути: title, content; абстрактні методи:

display(), get_duration()). Конкретні класи: Lecture (додає slides_count, перевизначає методи), Assignment (додає deadline, max_score), Quiz (додає questions_count, passing_score). Клас CourseModule (композиція: список CourseItem). Клас Course (атрибути: name, instructor, список CourseModule). Реалізуйте методи для підрахунку загальної тривалості курсу, відображення всіх матеріалів, додавання студентів, відстеження прогресу. Використовуйте isinstance() та issubclass() для перевірки типів.

```
    Вхід:
    python_course = Course("Програмування на Python", "Іван Петренко")

    module1 = CourseModule("Основи Python")
    module1.add_item(Lecture("Вступ до Python", "Основи синтаксису", 15))
    module1.add_item(Assignment("Перша програма", "Створіть Hello World", "2024-12-01", 10))
    module1.add_item(Quiz("Основи", 10, 7))

    python_course.add_module(module1)

    student1 = Student("Олена Сидорова")
    student2 = Student("Петро Іваненко")
    python_course.enroll_student(student1)
    python_course.enroll_student(student2)

    Вихід:
    python_course.display_course() → показує всю структуру курсу
    python_course.total_duration() → загальна тривалість курсу
    python_course.get_student_progress(student1) → прогрес студента

    Вхід: python_course.mark_completed(student1, lecture1) → "Лекцію 'Вступ до Python' завершено"
    Вихід: isinstance(module1, CourseModule) → True
    Вихід: issubclass(Lecture, CourseItem) → True
    Додатково: система оцінювання завдань, обчислення середнього балу, генерація звітів.
```

6. Питання для самоперевірки

1. Що таке наслідування в ООП і які переваги воно дає?

2. Поясніть різницю між батьківським (базовим) та дочірнім (похідним) класом.

3. Для чого використовується функція super() і в яких випадках її обов'язково викликати?

4. Що таке перевизначення методів (method overriding) і чим воно відрізняється від перевантаження?

5. Поясніть поняття поліморфізму в ООП. Наведіть приклад з життя.

6. Що таке множинне наслідування і які проблеми може викликати його використання?

7. Що таке MRO (Method Resolution Order) і як його можна перевірити в Python?

8. Для чого призначені абстрактні базові класи (ABC) та абстрактні методи?

9. Яка різниця між функціями isinstance() та type()?

10. Чим корисна функція issubclass() та у яких ситуаціях її варто використовувати?

11. Проаналізуйте код:

```
class A:
    def show(self):
        print("A")

class B(A):
    def show(self):
        print("B")

class C(B):
    def show(self):
        super().show()
        print("C")

obj = C()
obj.show()
```

Що буде виведено в результаті виконання цього коду і чому?

12. Що не так з цим кодом та як це виправити?

```
class Parent:
    def __init__(self, x):
        self.x = x

class Child(Parent):
    def __init__(self, x, y):
        self.y = y

obj = Child(10, 20)
print(obj.x)  # Викличе помилку
```

13. Проаналізуйте порядок виклику конструкторів:

```
class A:
    def __init__(self):
        print("A")

class B(A):
    def __init__(self):
        super().__init__()
        print("B")
class C(B):
    def __init__(self):
        super().__init__()
        print("C")

obj = C()
```

У якому порядку будуть виведені букви A, B, C?

14. Чому цей код викличе помилку та як його виправити?

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start_engine(self):
        pass

class Car(Vehicle):
    def drive(self):
        return "Driving"

car = Car()  # Помилка!
```

15. Який буде результат виконання цього коду?

```
class X: pass
class Y: pass
class Z: pass
class A(X, Y): pass
class B(Y, Z): pass
class M(A, B, Z): pass
print([c.__name__ for c in M.__mro__])
```

16. Коли краще використовувати композицію замість наслідування? Наведіть приклад.

17. У вас є класи Rectangle та Square. Чи правильно з точки зору ООП зробити Square підкласом Rectangle? Обґрунтуйте відповідь.

18. Як ви б розробили ієрархію класів для системи обліку тварин у зоопарку? Назвіть основні класи та їх відносини.

19. У чому різниця між інтерфейсом та абстрактним класом? Коли краще використовувати кожен підхід?

20. Як би ви реалізували систему платіжних методів (картка, PayPal, банківський переказ) з використанням поліморфізму?

21. Чи можна в Python наслідувати від вбудованих типів (наприклад, list, dict)? Наведіть приклад корисного застосування.

22. Що таке "diamond problem" (проблема ромба) у множинному наслідуванні та як Python її вирішує?

23. Чи може абстрактний клас містити неабстрактні методи? Наведіть приклад.

24. Як перевизначити оператор + для власного класу з використанням наслідування?

25. Чи можна змінити MRO класу після його створення? Якщо так, то як?

26. Питання для самоперевірки практичних навичок

27. Напишіть приклад коду, де клас Student наслідує від Person і додає атрибут student_id, використовуючи super().

28. Створіть абстрактний клас Shape з абстрактним методом area() та два конкретних класи Circle і Rectangle, які його реалізують.

29. Продемонструйте використання isinstance() для перевірки, чи об’єкт належить до певної ієрархії класів.

30. Напишіть приклад множинного наслідування з трьома батьківськими класами та продемонструйте MRO.

31. Створіть клас, який перевизначає методи __str__() та __repr__(), і поясніть різницю між ними.

32. Дайте визначення наступним термінам:

- інкапсуляція
- наслідування
- поліморфізм
- ієрархія класів
- абстракція
- міксин (mixin)
- "is-a" відношення
- "has-a" відношення

33. Які можуть бути негативні наслідки надмірного використання наслідування (проблема "глибокої ієрархії")?

34. Чому в деяких мовах програмування (наприклад, Java) немає множинного наслідування, а в Python воно є?

35. Коли слід використовувати абстрактні методи, а коли просто порожню реалізацію (pass)?

36. Як поліморфізм сприяє створенню гнучкого та розширюваного коду?

ЛАБОРАТОРНА РОБОТА №16
Інкапсуляція та спеціальні методи

1. Мета

Засвоїти принципи інкапсуляції в об'єктно-орієнтованому програмуванні на Python та навчитися використовувати спеціальні (магічні) методи для створення зручних, безпечних та інтуїтивно зрозумілих класів. У ході роботи студенти навчаться контролювати доступ до атрибутів об'єкта, перевизначати поведінку операторів і вбудованих функцій, а також створювати об'єкти, які можуть взаємодіяти з контекстними менеджерами та протоколами ітерації.

2. Завдання

1. Оволодіти концепцією інкапсуляції та її синтаксичним вираженням у Python через приватні (_, __) атрибути.
2. Навчитися керувати доступом до атрибутів класу за допомогою властивостей (@property) та методів-сеттерів і геттерів.
3. Зрозуміти призначення та реалізувати магічні методи для рядкового представлення об'єктів (__str__, __repr__).
4. Освоїти перевантаження основних арифметичних операторів (наприклад, __add__, __sub__) та операторів порівняння (наприклад, __eq__, __lt__).
5. Навчитися створювати класи-контейнери, реалізуючи методи __len__, __getitem__, __setitem__.
6. Розібратися з механізмом роботи контекстних менеджерів через методи __enter__ та __exit__.
7. Зрозуміти принцип роботи ітераторів та реалізувати їх за допомогою методів __iter__ та __next__.

3. Короткі теоретичні відомості

3.1. Інкапсуляція – це один із трьох основних принципів ООП, який полягає в об'єднанні даних (атрибутів) та методів для роботи з ними в єдину структуру (клас) та обмеженні прямого доступу до внутрішнього стану об'єкта. Це дозволяє захистити дані від некоректного використання та забезпечити контроль над їх зміною.

У Python інкапсуляція має переважно **конвенційний (договірний) характер**.

• _attribute **(захисний атрибут):** Одна зікса _ вказує розробникам, що атрибут чи метод призначений для внутрішнього використання в класі або його нащадках. Python не блокує доступ до таких атрибутів зовні, але це вважається поганим тоном.

```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance  # Захищений атрибут

account = BankAccount(100)
print(account._balance)  # Працює, але не рекомендується
```

• __attribute **(приватний атрибут):** Дві зікси __ на початку імені запускають механізм **name mangling** (спотворення імен). Інтерпретатор змінює ім'я атрибута на _ClassName__attribute, ускладнюючи випадковий доступ до нього ззовні. Це не є повною захищеністю, але сильним сигналом для розробника.

```
class BankAccount:
    def __init__(self, balance, pin):
        self.__balance = balance  # Приватний атрибут
        self.__pin = pin

    def check_balance(self, pin):
        if pin == self.__pin:
            return self.__balance
        else:
            return "Wrong PIN"

account = BankAccount(100, 1234)
# print(account.__balance)  # AttributeError: no attribute '__balance'
print(account._BankAccount__balance)  # Доступ є, але так робити не слід
print(account.check_balance(1234))  # Правильний спосіб доступу: 100
```

3.2. Властивості (Properties)

Для контрольованого доступу до приватних атрибутів використовують механізм **властивостей**. Декоратор @property дозволяє звертатися до методу як до атрибута для читання, а декоратори @attr.setter та @attr.deleter – для зміни та видалення значення з відповідною валідацією.

```
class Temperature:
```

```
        def __init__(self, celsius):
            self._celsius = celsius  # "Приховане" значення

        @property
        def celsius(self):
            """Геттер: повертає температуру в Цельсіях."""
            return self._celsius

        @celsius.setter
        def celsius(self, value):
            """Сеттер: встановлює температуру з перевіркою."""
            if value < -273.15:
                raise ValueError("Temperature below absolute zero
is not possible")
            self._celsius = value

        @property
        def fahrenheit(self):
            """Властивість лише для читання: обчислює
Фаренгейти."""
            return (self._celsius * 9/5) + 32

    temp = Temperature(25)
    print(temp.celsius)        # 25 (працює геттер)
    temp.celsius = 30          # Працює сеттер
    print(temp.fahrenheit)          # 86.0 (працює обчислювальна
властивість)
    # temp.fahrenheit = 100      # AttributeError: can't set
attribute
```

3.3. Спеціальні (магічні) методи

Це методи з подвійним підкресленням на початку та в кінці, які Python викликає автоматично у певних ситуаціях. Вони дозволяють нашим об'єктам працювати з вбудованими операторами та функціями.

- __str__(self) та __repr__(self)**:** Використовуються для рядкового представлення об'єкта.
  - __str__ – для "гарного", неформального виводу (наприклад, для користувача). Викликається функціями print() та str().
  - __repr__ – для однозначного, формального представлення, яке зазвичай є дійсним кодом Python для створення такого об'єкта. Викликається функцією repr() та у режимі відладки.

```
    class Book:
```

```
        def __init__(self, title, author):
            self.title = title
            self.author = author

        def __str__(self):
            return f"'{self.title}' by {self.author}"

        def __repr__(self):
            return f"Book('{self.title}', '{self.author}')"

    book = Book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams")
    print(book)             # Виведе: 'The Hitchhiker's Guide to the Galaxy' by Douglas Adams
    print(repr(book))       # Виведе: Book('The Hitchhiker's Guide to the Galaxy', 'Douglas Adams')
```

- **Перевантаження операторів:** Дозволяє визначити поведінку об'єкта при використанні стандартних операторів.
  - **Арифметичні:** __add__(self, other) (+), __sub__(self, other) (-), __mul__(self, other) (*), тощо.
  - **Порівняння:** __eq__(self, other) (==), __lt__(self, other) (<), __le__(self, other) (<=), __gt__(self, other) (>), __ge__(self, other) (>=).

```
    class Vector2D:
        def __init__(self, x, y):
            self.x = x
            self.y = y

        def __add__(self, other):
            """Додавання двох векторів."""
            return Vector2D(self.x + other.x, self.y + other.y)

        def __eq__(self, other):
            """Перевірка на рівність."""
            return self.x == other.x and self.y == other.y
        def __lt__(self, other):
            """Порівняння за довжиною вектора (для сортування)."""
            return (self.x**2 + self.y**2) < (other.x**2 + other.y**2)

    v1 = Vector2D(2, 3)
    v2 = Vector2D(1, 1)
    v3 = v1 + v2  # Викликається v1.__add__(v2)
    print(v3.x, v3.y)  # 3 4
```

```
print(v1 == v2)     # False
print(v1 > v2)      # True (бо 13 > 2)
```

- **Методи для контейнерів.** Роблять об’єкт схожим на послідовність (list) чи словник (dict).
  - __len__(self): Викликається функцією len().
  - __getitem__(self, key): Викликається для отримання значення за індексом/ключем obj[key].
  - __setitem__(self, key, value): Викликається для присвоєння значення obj[key] = value.

```
class SimpleStack:
    def __init__(self):
        self._items = []

    def push(self, item):
        self._items.append(item)

    def __len__(self):
        return len(self._items)

    def __getitem__(self, index):
        return self._items[index]

stack = SimpleStack()
stack.push('a')
stack.push('b')
print(len(stack))   # 2
print(stack[1])     # 'b'
```

- **Контекстні менеджери (__enter__, __exit__).** Дозволяють створювати об’єкти для використання в конструкції with. Це забезпечує автоматичне отримання та звільнення ресурсів (наприклад, відкриття/закриття файлу).

```
class FileLogger:
    def __init__(self, filename):
        self.filename = filename
        self.file = None

    def __enter__(self):
        self.file = open(self.filename, 'a')
        return self.file

    def __exit__(self, exc_type, exc_val, exc_tb):
```

```
            if self.file:
                self.file.close()

    # Використання
    with FileLogger('log.txt') as log_file:
        log_file.write("Operation started\n")
    # Файл гарантовано закритий навіть у разі помилки
```

- **Ітератори (__iter__, __next__).** Дозволяють об'єкту підтримувати ітерацію в циклі for.
  - __iter__(self): Повинен повертати сам ітератор (зазвичай self).
  - __next__(self): Повинен повертати наступне значення або викидати виняток StopIteration, коли елементи закінчилися.

```
class Countdown:
    def __init__(self, start):
        self.current = start

    def __iter__(self):
        return self

    def __next__(self):
        if self.current <= 0:
            raise StopIteration
        value = self.current
        self.current -= 1
        return value

for number in Countdown(5):
    print(number)  # Виведе: 5 4 3 2 1
```

Опанування цих механізмів є ключовим для створення елегантних, надійних та "пітонічних" (pythonic) класів, які інтегруються в екосистему мови.

4. Методичні рекомендації

Нижче наведено розв'язки типових задач, які допоможуть засвоїти основні концепції інкапсуляції та спеціальних методів. Уважно вивчайте код та коментарі до нього.

Задача 1. Контроль доступу до балансу рахунку

Створіть клас BankAccount, який інкапсулює баланс рахунку як приватний атрибут. Реалізуйте властивість balance з геттером та сеттером. Сеттер повинен заборонити встановлення від'ємного балансу.

```
    class BankAccount:
        """Клас банківського рахунку з контролем балансу."""

        def __init__(self, initial_balance: float = 0.0):
            # Ініціалізуємо приватний атрибут з подвійним
підкресленням
            self.__balance = 0.0
            # Використовуємо сеттер для ініціалізації з валідацією
            self.balance = initial_balance

        @property
        def balance(self) -> float:
            """Геттер для балансу."""
            return self.__balance

        @balance.setter
        def balance(self, new_balance: float) -> None:

            """Сеттер для балансу з перевіркою на від'ємне
значення."""
            if new_balance < 0:
                raise ValueError("Баланс не може бути від'ємним")
            self.__balance = new_balance

        def deposit(self, amount: float) -> None:
            """Поповнити рахунок."""
            if amount <= 0:
                raise ValueError("Сума поповнення має бути
додатною")
            self.balance += amount

        def withdraw(self, amount: float) -> None:
            """Зняти кошти з рахунку."""
            if amount <= 0:
                raise ValueError("Сума зняття має бути додатною")
            # Використовуємо сеттер через self.balance для
автоматичної валідації
            self.balance -= amount
```

Приклад вхідних та вихідних даних:

```
# Вхід: створення рахунку з балансом 100
```

```
account1 = BankAccount(100)
print(account1.balance)  # Вихід: 100

# Вхід: спроба встановити від'ємний баланс
try:
    account1.balance = -50
except ValueError as e:
    print(e)  # Вихід: "Баланс не може бути від'ємним"

# Вхід: поповнення на 50 та зняття 30
account1.deposit(50)
account1.withdraw(30)
print(account1.balance)  # Вихід: 120

# Вхід: створення рахунку зі стандартним значенням
account2 = BankAccount()
print(account2.balance)  # Вихід: 0.0
```

Коментарі:

1. Використання __balance робить атрибут приватним (name mangling).

2. Сеттер використовується навіть в __init__ для забезпечення валідації при будь-якому присвоєнні.

3. Методи deposit та withdraw використовують властивість balance, що автоматично забезпечує валідацію.

Задача 2. Клас для роботи з температурами

Створіть клас Temperature, який зберігає температуру в градусах Цельсія. Реалізуйте властивості для доступу до температури у Цельсіях та Фаренгейтах (лише для читання). Температура не може бути нижче абсолютного нуля (-273.15°C).

```
class Temperature:
    """Клас для роботи з температурами."""

    ABSOLUTE_ZERO = -273.15  # Константа абсолютного нуля

    def __init__(self, celsius: float):
        self._celsius = celsius  # Захищений атрибут
        self._validate(celsius)

    def _validate(self, celsius: float) -> None:
```

```
            """Приватний метод валідації температури."""
            if celsius < self.ABSOLUTE_ZERO:
                raise ValueError(f"Температура не може бути нижче
{self.ABSOLUTE_ZERO}°C")

        @property
        def celsius(self) -> float:
            """Температура в градусах Цельсія."""
            return self._celsius

        @celsius.setter
        def celsius(self, value: float) -> None:
            """Встановити температуру в Цельсіях з валідацією."""
            self._validate(value)
            self._celsius = value
        @property
        def fahrenheit(self) -> float:
            """Температура в градусах Фаренгейта (read-only)."""
            return (self._celsius * 9/5) + 32

        def __str__(self) -> str:
            """Неформальне строкове представлення."""
            return                              f"{self._celsius:.1f}°C
({self.fahrenheit:.1f}°F)"

        def __repr__(self) -> str:
            """Формальне представлення для відладки."""
            return f"Temperature({self._celsius})"
```

Приклад вхідних та вихідних даних:

```
# Вхід: створення об'єкта з температурою 25°C
temp1 = Temperature(25)
print(temp1.celsius)      # Вихід: 25
print(temp1.fahrenheit)   # Вихід: 77.0
print(temp1)              # Вихід: 25.0°C (77.0°F)

# Вхід: зміна температури через сеттер
temp1.celsius = 30
print(temp1.fahrenheit)   # Вихід: 86.0

# Вхід: спроба створити температуру нижче абсолютного нуля
try:
    temp2 = Temperature(-300)
except ValueError as e:
```

```
        print(e)   # Вихід: "Температура не може бути нижче -
273.15°C"

    # Вхід: перевірка repr
    print(repr(temp1))   # Вихід: Temperature(30)
```

Коментарі:

1. Використовується захищений атрибут _celsius за домовленістю.
2. Валідація винесена в окремий приватний метод _validate.
3. Властивість fahrenheit не має сеттера, тому доступна лише для читання.
4. Реалізовані обидва магічні методи для строкового представлення.

Задача 3. Перевантаження операторів для вектора

Створіть клас Vector2D, який реалізує двовимірний вектор. Перевантажте оператори +, -, == та * (множення на скаляр).

```
class Vector2D:
    """Клас двовимірного вектора з перевантаженими
операторами."""

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def __add__(self, other: 'Vector2D') -> 'Vector2D':
        """Додавання двох векторів: v1 + v2."""
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other: 'Vector2D') -> 'Vector2D':
        """Віднімання векторів: v1 - v2."""
        return Vector2D(self.x - other.x, self.y - other.y)

    def __mul__(self, scalar: float) -> 'Vector2D':
        """Множення вектора на скаляр: v * k."""
        if not isinstance(scalar, (int, float)):
            raise TypeError("Множник має бути числом")
        return Vector2D(self.x * scalar, self.y * scalar)

    def __rmul__(self, scalar: float) -> 'Vector2D':
        """Праве множення: k * v."""
        return self.__mul__(scalar)

    def __eq__(self, other: object) -> bool:
```

```
        """Перевірка на рівність: v1 == v2."""
        if not isinstance(other, Vector2D):
            return False
        return self.x == other.x and self.y == other.y

    def __str__(self) -> str:
        """Зручне представлення вектора."""
        return f"Vector2D({self.x}, {self.y})"
    def length(self) -> float:
        """Обчислення довжини вектора."""
        return (self.x**2 + self.y**2) ** 0.5
```

Приклад вхідних та вихідних даних:

```
# Вхід: створення векторів
v1 = Vector2D(2, 3)
v2 = Vector2D(1, 1)

# Вхід: додавання та віднімання
v3 = v1 + v2
print(v3)  # Вихід: Vector2D(3, 4)

v4 = v1 - v2
print(v4)  # Вихід: Vector2D(1, 2)

# Вхід: множення на скаляр (зліва та справа)
v5 = v1 * 2
print(v5)  # Вихід: Vector2D(4, 6)

v6 = 3 * v1
print(v6)  # Вихід: Vector2D(6, 9)

# Вхід: порівняння
print(v1 == v2)           # Вихід: False
print(v1 == Vector2D(2, 3))  # Вихід: True

# Вхід: обчислення довжини
print(v1.length())  # Вихід: 3.605551275463989
```

Коментарі:

1. Метод __rmul__ дозволяє виконувати операцію 3 * v1 коли v1 * 3 не підходить.

2. У методі __eq__ перевіряється тип other, щоб уникнути помилок.

3. Метод __mul__ містить перевірку типу для безпечного множення.

Задача 4. Клас-контейнер для зберігання студентів

Створіть клас StudentList, який поводиться як послідовність. Реалізуйте методи __len__, __getitem__, __setitem__ та __contains__.

```
class StudentList:
    """Клас-контейнер для зберігання студентів."""

    def __init__(self):
        self._students = []   # Прихований список студентів
        self._index = 0       # Для підтримки ітерації

    def add_student(self, name: str, grade: float) -> None:
        """Додати студента до списку."""
        self._students.append({"name": name, "grade": grade})

    def __len__(self) -> int:
        """Повертає кількість студентів."""
        return len(self._students)

    def __getitem__(self, index):
        """Доступ до студента за індексом або зрізом."""
        if isinstance(index, slice):
            # Обробка зрізів: students[1:3]
            return self._students[index]
        return self._students[index]

    def __setitem__(self, index, value):
        """Змінити дані студента за індексом."""
        self._students[index] = value

    def __contains__(self, name: str) -> bool:
        """Перевірка наявності студента за ім'ям."""
        return any(student["name"] == name for student in
self._students)

    def __iter__(self):
        """Повертає ітератор."""
        self._index = 0
        return self
    def __next__(self):
```

```
            """Повертає наступного студента."""
            if self._index < len(self._students):
                result = self._students[self._index]
                self._index += 1
                return result
            raise StopIteration

        def __str__(self):
            """Представлення у вигляді списку."""
            return str([f"{s['name']}: {s['grade']}" for s in
self._students])
```

Приклад вхідних та вихідних даних:

```
    # Вхід: створення списку студентів
    students = StudentList()
    students.add_student("Іван Петренко", 85.5)
    students.add_student("Марія Коваль", 92.0)
    students.add_student("Олексій Шевченко", 78.5)

    # Вхід: перевірка довжини
    print(len(students))  # Вихід: 3

    # Вхід: доступ за індексом
    print(students[1])  # Вихід: {'name': 'Марія Коваль',
'grade': 92.0}

    # Вхід: зміна оцінки через індексацію
    students[2] = {"name": "Олексій Шевченко", "grade": 80.0}
    print(students[2])  # Вихід: {'name': 'Олексій Шевченко',
'grade': 80.0}

    # Вхід: перевірка наявності студента
    print("Марія Коваль" in students)  # Вихід: True
    print("Петро Сидоренко" in students)  # Вихід: False

    # Вхід: ітерація по студентах
    for student in students:
        print(f"{student['name']} - {student['grade']}")
    # Вихід:
    # Іван Петренко - 85.5
    # Марія Коваль - 92.0
    # Олексій Шевченко - 80.0

    # Вхід: робота зі зрізом
```

```
print(students[0:2])   # Вихід: [{'name': 'Іван Петренко',
'grade': 85.5}, {...}]
```

Коментарі:

1. Клас інкапсулює внутрішній список _students.
2. Метод __getitem__ обробляє як індекси, так і зрізи.
3. Метод __contains__ реалізований через генераторний вираз.
4. Реалізований повний протокол ітератора через __iter__ та __next__.

Задача 5. Простий контекстний менеджер для таймера

Створіть контекстний менеджер Timer, який вимірює час виконання блоку коду.

```
import time

class Timer:
    """Контекстний менеджер для вимірювання часу виконання."""

    def __init__(self, name: str = "Операція"):
        self.name = name

    def __enter__(self):
        """Викликається при вході в контекст."""
        self.start_time = time.perf_counter()
        print(f"Початок {self.name}...")
        return self  # Можна повернути об'єкт для використання
в блоці with

    def __exit__(self, exc_type, exc_val, exc_tb):
        """Викликається при виході з контексту."""
        self.end_time = time.perf_counter()
        self.elapsed = self.end_time - self.start_time
        print(f"Кінець {self.name}. Час виконання:
{self.elapsed:.4f} секунд")

        # Якщо повернути True, виняток буде придушений
        # Якщо False або None - виняток буде передано далі
        return False
```

Приклад вхідних та вихідних даних:

```
# Вхід: вимірювання часу виконання циклу
with Timer("Обчислення суми"):
```

```
        total = 0
        for i in range(1_000_000):
            total += i
# Вихід:
# Початок Обчислення суми...
# Кінець Обчислення суми. Час виконання: 0.0453 секунд

# Вхід: вимірювання з іменем та доступом до об'єкта
with Timer("Створення списку") as timer:
    my_list = [x**2 for x in range(10000)]
    # Можна отримати проміжний час всередині блоку
    current_time = time.perf_counter() - timer.start_time
    print(f"Проміжний час: {current_time:.4f} секунд")
# Вихід:
# Початок Створення списку...
# Проміжний час: 0.0012 секунд
# Кінець Створення списку. Час виконання: 0.0015 секунд

# Вхід: контекстний менеджер з обробкою помилок
try:
    with Timer("Операція з помилкою"):
        result = 10 / 0
except ZeroDivisionError:
    print("Виникла помилка ділення на нуль")
# Вихід:
# Початок Операція з помилкою...
# Кінець Операція з помилкою. Час виконання: 0.0001 секунд
# Виникла помилка ділення на нуль
```

Коментарі:

1. Метод __enter__ повертає self, що дозволяє отримати доступ до таймера всередині блоку.

2. Метод __exit__ отримує інформацію про виняток (якщо він виник).

3. Використання time.perf_counter() забезпечує максимальну точність.

Задача 6. Ітератор для послідовності Фібоначчі

Створіть ітератор Fibonacci, який генерує послідовність чисел Фібоначчі.

```
class Fibonacci:
    """Ітератор для послідовності Фібоначчі."""
```

```
        def __init__(self, limit: int = 10):
            self.limit = limit  # Максимальна кількість чисел
            self.count = 0      # Лічильник згенерованих чисел
            self.a, self.b = 0, 1  # Перші два числа

        def __iter__(self):
            """Повертає самого себе як ітератор."""
            return self

        def __next__(self):
            """Генерує наступне число Фібоначчі."""
            if self.count >= self.limit:
                raise StopIteration

            self.count += 1
            current = self.a
            self.a, self.b = self.b, self.a + self.b
            return current

        def __len__(self):
            """Повертає кількість чисел, які будуть
згенеровані."""
            return self.limit

        def __str__(self):
            """Повертає рядок з усією послідовністю."""
            # Створюємо новий ітератор, щоб не порушити поточний
стан
            temp_list = list(Fibonacci(self.limit))
            return f"Fibonacci({self.limit}): {temp_list}"
```

Приклад вхідних та вихідних даних:

```
    # Вхід: ітерація по перших 10 числах Фібоначчі
    fib = Fibonacci(10)
    for num in fib:
        print(num, end=" ")
    # Вихід: 0 1 1 2 3 5 8 13 21 34

    print()  # Новий рядок

# Вхід: повторна спроба ітерації (ітератор вичерпано)
    print("Повторна ітерація:")
    for num in fib:
```

```
        print(num, end=" ")    # Нічого не виведе - ітератор вичерпано

    # Вхід: створення нового ітератора з обмеженням 5
    fib_short = Fibonacci(5)
    print(f"\nДовжина: {len(fib_short)}")  # Вихід: Довжина: 5
    print(fib_short)  # Вихід: Fibonacci(5): [0, 1, 1, 2, 3]

    # Вхід: перетворення в список
    fib_list = list(Fibonacci(7))
    print(fib_list)  # Вихід: [0, 1, 1, 2, 3, 5, 8]

    # Вхід: використання next() вручну
    fib_manual = Fibonacci(3)
    print(next(fib_manual))  # Вихід: 0
    print(next(fib_manual))  # Вихід: 1
    print(next(fib_manual))  # Вихід: 1
    # print(next(fib_manual))  # StopIteration
```

Коментарі:

1. Ітератор зберігає свій стан (a, b, count) між викликами __next__.
2. Після вичерпання (count >= limit) ітератор завжди викликає StopIteration.
3. Метод __str__ створює новий об’єкт, щоб не змінювати стан поточного.
4. Такий ітератор можна використовувати лише один раз (single-use).

Задача 7. Комбінований клас з інкапсуляцією та спеціальними методами

Створіть клас SmartArray, який поєднує інкапсуляцію, властивості, перевантаження операторів та методи контейнера.

```
class SmartArray:
    """Розумний масив з інкапсуляцією та спеціальними методами."""

    def __init__(self, *args):
        # Зберігаємо дані в приватному списку
        self.__data = list(args)

    # Інкапсуляція через властивості
    @property
    def data(self):
```

```
        """Повертає копію даних для захисту від змін."""
        return self.__data.copy()

    @property
    def sum(self):
        """Сума всіх елементів (read-only)."""
        return sum(self.__data)

    @property
    def average(self):
        """Середнє арифметичне (read-only)."""
        if len(self.__data) == 0:
            return 0
        return self.sum / len(self.__data)

    # Методи контейнера
    def __len__(self):
        return len(self.__data)

    def __getitem__(self, index):
        return self.__data[index]
    def __setitem__(self, index, value):
        self.__data[index] = value

    def __contains__(self, item):
        return item in self.__data

    # Арифметичні операції
    def __add__(self, other):
        """Конкатенація двох SmartArray або додавання числа."""
        if isinstance(other, SmartArray):
            return SmartArray(*(self.__data + other.__data))
        elif isinstance(other, (int, float)):
            return SmartArray(*[x + other for x in self.__data])
        else:
            raise TypeError("Непідтримуваний тип для додавання")

    # Порівняння
    def __eq__(self, other):
        if not isinstance(other, SmartArray):
            return False
        return self.__data == other.__data

    def __lt__(self, other):
```

```
        """Порівняння за сумою елементів."""
        if not isinstance(other, SmartArray):
            raise TypeError("Можна порівнювати лише з SmartArray")
        return self.sum < other.sum

    # Строкові представлення
    def __str__(self):
        return f"SmartArray{tuple(self.__data)}"

    def __repr__(self):
        return f"SmartArray({', '.join(map(str, self.__data))})"

    # Корисні методи
    def append(self, value):
        """Додати значення в кінець."""
        self.__data.append(value)
    def clear(self):
        """Очистити масив."""
        self.__data.clear()
```

Приклад вхідних та вихідних даних:

```
# Вхід: створення об'єкта
arr1 = SmartArray(1, 2, 3, 4, 5)
arr2 = SmartArray(6, 7, 8)

# Вхід: доступ через властивості
print(f"Дані: {arr1.data}")       # Вихід: Дані: [1, 2, 3, 4, 5]
print(f"Сума: {arr1.sum}")        # Вихід: Сума: 15
print(f"Середнє: {arr1.average}") # Вихід: Середнє: 3.0

# Вхід: методи контейнера
print(f"Довжина: {len(arr1)}")    # Вихід: Довжина: 5
print(f"arr1[2]: {arr1[2]}")      # Вихід: arr1[2]: 3
print(f"3 в arr1: {3 in arr1}")   # Вихід: 3 в arr1: True

# Вхід: арифметичні операції
arr3 = arr1 + arr2
print(f"arr1 + arr2: {arr3}")     # Вихід: arr1 + arr2: SmartArray(1, 2, 3, 4, 5, 6, 7, 8)

arr4 = arr1 + 10
print(f"arr1 + 10: {arr4}")       # Вихід: arr1 + 10: SmartArray(11, 12, 13, 14, 15)
```

```
# Вхід: порівняння
print(f"arr1 == arr2: {arr1 == arr2}")   # Вихід: arr1 == arr2: False
print(f"arr1 < arr2: {arr1 < arr2}")     # Вихід: arr1 < arr2: True (15 < 21)

# Вхід: строкові представлення
print(str(arr1))   # Вихід: SmartArray(1, 2, 3, 4, 5)
print(repr(arr1))  # Вихід: SmartArray(1, 2, 3, 4, 5)

# Вхід: перевірка інкапсуляції
arr1.append(100)
print(arr1.data)   # Вихід: [1, 2, 3, 4, 5, 100]
# arr1.__data = []  # AttributeError: no attribute '__data'
```

Коментарі:

1. Клас демонструє комплексне використання ООП-принципів.
2. Приватний атрибут __data захищений від прямого доступу.
3. Властивість data повертає копію, щоб захистити внутрішній список.
4. Реалізовано перевантаження для різних типів в __add__.
5. Метод __lt__ порівнює об’єкти за власною логікою (сумою елементів).

Типові помилки і шляхи їх усунення

1. **Забувають реалізувати __next__ для ітератора**
   Проблема. TypeError: iter() returned non-iterator
   Рішення. Клас-ітератор повинен мати ОБИДВА методи: __iter__ (повертає self) та __next__ (повертає наступне значення).
2. **Неправильне використання @property без сеттера**
   Проблема. AttributeError: can’t set attribute
   Рішення. Якщо властивість має бути змінюваною, обов’язково реалізуйте сеттер: @attr.setter.
3. **__eq__ не повертає False для несумісних типів**
   Проблема. AttributeError при порівнянні з об’єктом іншого типу
   Рішення. Завжди перевіряйте isinstance(other, MyClass) на початку __eq__.
4. **Зміна приватних атрибутів через _ClassName__attr**
   Проблема. Обхід інкапсуляції, непередбачувана поведінка

Рішення. Ніколи не використовуйте спотворені імена напряму. Це порушує принципи ООП.

5. **Забувають викликати __init__ батьківського класу**

Проблема. Атрибути базового класу не ініціалізуються

Рішення. У __init__ похідного класу завжди викликайте super().__init__().

6. **Неправильна реалізація __exit__**

Проблема. Винятки в контекстному менеджері не обробляються

Рішення. Метод __exit__ має приймати три аргументи про виняток і повертати True/False для контролю над винятком.

7. **Ітератор не скидає стан при повторній ітерації**

Проблема. Друга ітерація не працює

Рішення. У __iter__ скидайте лічильники або створюйте новий ітератор.

Корисні поради

1. **Принцип "Не питай, а повідомляй".** Краще реалізувати властивості (@property), ніж методи типу get_balance(). Це робить код чистішим: account.balance замість account.get_balance().

2. **Використовуйте __slots__ для оптимізації.** Для класів з фіксованим набором атрибутів визначте __slots__ для зменшення використання пам'яті та пришвидшення доступу.

```
class Point:
    __slots__ = ('x', 'y')  # Забороняє створення __dict__
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

3. **Розділяйте відповідальність.** Не навантажуйте один клас усіма магічними методами. Створюйте окремі класи для окремих завдань: один для контейнера, інший для ітератора тощо.

4. **Дотримуйтесь конвенцій**

- _single_leading_underscore: "захищений", для внутрішнього використання
- __double_leading_underscore: приватний (name mangling)
- __double_underscore__: магічний метод
- single_trailing_underscore_: щоб уникнути конфлікту з ключовими словами

5. **Тестуйте спеціальні методи:** Переконайтеся, що ваші магічні методи працюють з вбудованими функціями:

```
obj = MyClass()
print(len(obj))       # Перевірка __len__
print(obj[0])         # Перевірка __getitem__
print(obj == obj2)    # Перевірка __eq__
with obj as context: # Перевірка __enter__/__exit__
    pass
```

6. **Документуйте магічні методи:** Оскільки вони викликаються неявно, чіткі docstrings допоможуть іншим розробникам зрозуміти логіку.

7. **Використовуйте абстрактні базові класи (ABC):** Для формалізації інтерфейсів використовуйте модуль collections.abc.

```
from collections.abc import Sequence, Iterator

class MySequence(Sequence):
    # Потрібно реалізувати __len__ та __getitem__
    pass
```

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Клас для зберігання пароля

Створіть клас PasswordManager, який зберігає пароль у приватному атрибуті. Реалізуйте властивість password з геттером та сеттером. Сеттер повинен перевіряти, що пароль містить принаймні 8 символів та хоча б одну цифру. При ініціалізації пароль встановлюється у пустий рядок.

```
Вхід:
pm = PasswordManager()
pm.password = "qwerty"
Вихід: ValueError("Пароль має містити принаймні 8 символів та хоча б одну цифру")

Вхід:
pm = PasswordManager()
pm.password = "securepass123"
print(pm.password)
Вихід: "securepass123"
```

```
Вхід:
pm = PasswordManager()
pm.password = "short1"
Вихід: ValueError("Пароль має містити принаймні 8 символів та хоча б одну цифру")
```

Завдання 1.2. Клас для обмеження віку

Створіть клас Person, який зберігає ім'я та вік особи. Вік має бути приватним атрибутом. Реалізуйте властивість age з геттером та сеттером. Сеттер не дозволяє встановити вік менше 0 або більше 120 років. При спробі встановити некоректне значення генерується виняток ValueError.

```
Вхід:
person = Person("Іван", 25)
print(person.age)
Вихід: 25

Вхід:
person = Person("Марія", 30)
person.age = 150
Вихід: ValueError("Вік має бути від 0 до 120 років")

Вхід:
person = Person("Петро", -5)
Вихід: ValueError("Вік має бути від 0 до 120 років")
```

Завдання 1.3. Клас для роботи з колом

Створіть клас Circle, який зберігає радіус як приватний атрибут. Реалізуйте властивості: radius (геттер та сеттер з перевіркою на додатне значення), diameter (тільки геттер, обчислюється як 2*радіус),* area *(тільки геттер, обчислюється як* πr^2).

```
Вхід:
circle = Circle(5)
print(circle.radius, circle.diameter, circle.area)
Вихід: 5 10 78.53981633974483

Вхід:
circle = Circle(10)
circle.radius = -3
Вихід: ValueError("Радіус має бути додатним числом")

Вхід:
circle = Circle(7)
```

```
circle.radius = 2.5
print(round(circle.area, 2))
Вихід: 19.63
```

Завдання 1.4. Клас для зберігання налаштувань

Створіть клас Settings, який має приватні атрибути для зберігання налаштувань: мова, тема, сповіщення. Реалізуйте властивості для кожного з них з перевіркою допустимих значень: мова - "uk", "en", "de"; тема - "light", "dark"; сповіщення - True/False.

```
Вхід:
settings = Settings()
settings.language = "en"
settings.theme = "dark"
settings.notifications = False
print(settings.language, settings.theme, settings.notifications)
Вихід: "en dark False"

Вхід:
settings = Settings()
settings.language = "fr"
Вихід: ValueError("Мова має бути однією з: 'uk', 'en', 'de'")

Вхід:
settings = Settings()
settings.theme = "blue"
Вихід: ValueError("Тема має бути 'light' або 'dark'")
```

Завдання 1.5. Клас для товару зі знижкою

Створіть клас Product, який зберігає назву товару, ціну та відсоток знижки (приватні атрибути). Реалізуйте властивість final_price, яка повертає ціну з урахуванням знижки. Відсоток знижки може бути від 0 до 50.

```
Вхід:
product = Product("Книга", 200, 10)
print(product.final_price)
Вихід: 180

Вхід:
product = Product("Телефон", 10000, 60)
Вихід: ValueError("Знижка має бути від 0 до 50%")

Вхід:
```

```
product = Product("Кава", 150, 0)
product.discount = 20
print(product.final_price)
Вихід: 120
```

Завдання 1.6. Клас для представлення дроби

Створіть клас Fraction, який зберігає чисельник та знаменник. Реалізуйте методи __str__ та __repr__. Метод __str__ повинен повертати дріб у форматі "чисельник/знаменник", а __repr__ - у форматі "Fraction(чисельник, знаменник)".

```
Вхід:
f = Fraction(3, 4)
print(str(f))
Вихід: "3/4"

Вхід:
f = Fraction(1, 2)
print(repr(f))
Вихід: "Fraction(1, 2)"

Вхід:
f = Fraction(5, 1)
print(f)  # викликається __str__
Вихід: "5/1"
```

Завдання 1.7. Клас для координатної точки

Створіть клас Point, який зберігає координати x та y. Реалізуйте методи __str__ (повертає "Point(x, y)") та __repr__ (повертає рядок, який можна використати для створення нового об’єкта). Також реалізуйте метод __len__, який повертає відстань від точки до початку координат (округлену до цілого).

```
Вхід:
p = Point(3, 4)
print(p)
Вихід: "Point(3, 4)"

Вхід:
p = Point(1, 2)
print(repr(p))
Вихід: "Point(1, 2)"

Вхід:
```

```
p = Point(3, 4)
print(len(p))
Вихід: 5
```

Завдання 1.8. Клас для представлення часу

Створіть клас Time, який зберігає години, хвилини та секунди. Реалізуйте метод __str__, який повертає час у форматі "HH:MM:SS", та метод __repr__, який повертає "Time(години, хвилини, секунди)". Значення годин мають бути від 0 до 23, хвилин і секунд - від 0 до 59.

```
Вхід:
t = Time(14, 30, 45)
print(str(t))
Вихід: "14:30:45"

Вхід:
t = Time(8, 5, 0)
print(repr(t))
Вихід: "Time(8, 5, 0)"

Вхід:
t = Time(23, 59, 59)
print(t)
Вихід: "23:59:59"
```

Завдання 1.9. Клас для представлення колекції книг

Створіть клас BookCollection, який зберігає список книг (кожна книга - це рядок). Реалізуйте метод __str__, який повертає список книг, розділених комами, та метод __repr__, який повертає "BookCollection([‘книга1’, ‘книга2’, ...])". Також реалізуйте __len__, який повертає кількість книг.

```
Вхід:
bc = BookCollection(["1984", "Гаррі Поттер", "Мастер і Маргарита"])
print(str(bc))
Вихід: "1984, Гаррі Поттер, Мастер і Маргарита"

Вхід:
bc = BookCollection(["Python Crash Course"])
print(repr(bc))
Вихід: "BookCollection([‘Python Crash Course’])"

Вхід:
```

```
bc = BookCollection(["a", "b", "c", "d"])
print(len(bc))
Вихід: 4
```

Завдання 1.10. Клас для представлення RGB-кольору

Створіть клас Color, який зберігає значення червоного, зеленого та синього кольорів (від 0 до 255). Реалізуйте метод __str__, який повертає колір у форматі "RGB(r, g, b)", та метод __repr__, який повертає "Color(r, g, b)". Реалізуйте також метод __bool__, який повертає False, якщо всі компоненти дорівнюють 0 (чорний колір), інакше True.

```
Вхід:
c = Color(255, 0, 0)
print(str(c))
Вихід: "RGB(255, 0, 0)"

Вхід:
c = Color(0, 128, 64)
print(repr(c))
Вихід: "Color(0, 128, 64)"

Вхід:
c1 = Color(0, 0, 0)
c2 = Color(1, 0, 0)
print(bool(c1), bool(c2))
Вихід: "False True"
```

Частина 2. Середні завдання

Завдання 2.1. Клас для комплексних чисел

Створіть клас ComplexNumber, який реалізує комплексні числа. Перевантажте оператори +, -, * та /. Також реалізуйте метод __abs__, який повертає модуль комплексного числа. Кожна операція має повертати новий об’єкт ComplexNumber.

```
Вхід:
a = ComplexNumber(1, 2)
b = ComplexNumber(3, 4)
print(a + b)
Вихід: ComplexNumber(4, 6)

Вхід:
a = ComplexNumber(5, 7)
```

```
    b = ComplexNumber(2, 3)
    print(a * b)
    Вихід:   ComplexNumber(-11,   29)      #   (5+7i)*(2+3i)   =
10+15i+14i+21i² = -11+29i

    Вхід:
    a = ComplexNumber(3, 4)
    print(abs(a))
    Вихід: 5.0
```

Завдання 2.2. Клас для матриць 2x2

Створіть клас Matrix2x2, який представляє матрицю 2x2. Перевантажте оператори +, -, * (множення на іншу матрицю та на скаляр). Реалізуйте метод __invert__ для обчислення оберненої матриці (якщо вона існує).

```
    Вхід:
    m1 = Matrix2x2(1, 2, 3, 4)
    m2 = Matrix2x2(5, 6, 7, 8)
    print(m1 + m2)
    Вихід: Matrix2x2(6, 8, 10, 12)

    Вхід:
    m1 = Matrix2x2(1, 2, 3, 4)
    m2 = Matrix2x2(2, 0, 1, 2)
    print(m1 * m2)
    Вихід: Matrix2x2(4, 4, 10, 8)  # матричне множення

    Вхід:
    m = Matrix2x2(4, 7, 2, 6)
    print(~m)  # обернена матриця
    Вихід: Matrix2x2(0.6, -0.7, -0.2, 0.4)
```

Завдання 2.3. Клас для поліномів

Створіть клас Polynomial, який зберігає коефіцієнти поліному (наприклад, [3, 0, 2] для $3x^2 + 2$). Перевантажте оператори +, -, * для додавання, віднімання та множення поліномів. Реалізуйте метод __call__, який обчислює значення поліному для заданого x.

```
    Вхід:
    p1 = Polynomial([1, 2, 3])  # 3x² + 2x + 1
    p2 = Polynomial([3, 1])     # x + 3
    print(p1 + p2)
    Вихід: Polynomial([4, 3, 3])
```

```
Вхід:
p1 = Polynomial([2, 0, 1])  # x² + 2
p2 = Polynomial([1, 1])     # x + 1
print(p1 * p2)
Вихід: Polynomial([2, 2, 1, 1])  # x³ + x² + 2x + 2

Вхід:
p = Polynomial([1, 0, 2])  # 2x² + 1
print(p(3))
Вихід: 19
```

Завдання 2.4. Клас для роботи з діапазонами дат

Створіть клас DateRange, який представляє діапазон дат. Реалізуйте методи порівняння (__lt__, __le__, __eq__, __ne__, __gt__, __ge__), де порівняння відбувається за довжиною діапазону (кількістю днів). Також реалізуйте метод __contains__, який перевіряє, чи потрапляє задана дата в діапазон.

```
Вхід:
import datetime
dr1 = DateRange(datetime.date(2024, 1, 1), datetime.date(2024, 1,
10))
dr2 = DateRange(datetime.date(2024, 1, 1), datetime.date(2024, 1,
5))
print(dr1 > dr2)
Вихід: True

Вхід:
dr = DateRange(datetime.date(2024, 5, 1), datetime.date(2024, 5,
31))
print(datetime.date(2024, 5, 15) in dr)
Вихід: True

Вхід:
dr1 = DateRange(datetime.date(2024, 6, 1), datetime.date(2024, 6,
10))
dr2 = DateRange(datetime.date(2024, 6, 5), datetime.date(2024, 6,
15))
print(dr1 < dr2)
Вихід: False (обидва по 10 днів)
```

Завдання 2.5. Клас-словник з обмеженням розміру

Створіть клас LRUCache (Least Recently Used Cache), який поводиться як словник, але має обмежену ємність. При перевищенні ємності видаляється найдавніше використаний елемент. Реалізуйте методи __getitem__, __setitem__, __len__, __contains__, __delitem__. При звертанні до елемента через __getitem__ він вважається "використаним".

```
Вхід:
cache = LRUCache(3)
cache["a"] = 1
cache["b"] = 2
cache["c"] = 3
cache["d"] = 4  # має видалити "a"
print("a" in cache, "b" in cache)
Вихід: False True

Вхід:
cache = LRUCache(2)
cache["x"] = 10
cache["y"] = 20
val = cache["x"]  # "x" стає нещодавно використаним
cache["z"] = 30   # видалить "y", а не "x"
print(cache.keys())
Вихід: ["x", "z"]

Вхід:
cache = LRUCache(3)
cache[1] = "one"
cache[2] = "two"
cache[3] = "three"
del cache[2]
print(len(cache))
Вихід: 2
```

Частина 3. Складні завдання

Завдання 3.1. Контекстний менеджер для транзакцій бази даних

Створіть контекстний менеджер Transaction, який імітує роботу з транзакціями бази даних. При вході в контекст створюється "точка збереження", при успішному виході - "фіксація", при виникненні винятку - "відкат". Клас має зберігати список операцій (рядки), які

виконуються в транзакції. Реалізуйте методи add_operation(operation) для додавання операції до транзакції.

```
    Вхід:
    with Transaction() as t:
        t.add_operation("INSERT INTO users VALUES (1, 'John')")
        t.add_operation("UPDATE accounts SET balance = 1000")
    print("Транзакція успішна")
    Вихід:
    Початок транзакції
    Транзакція успішна
    Фіксація транзакції: ['INSERT INTO users VALUES (1, 'John')',
'UPDATE accounts SET balance = 1000']

    Вхід:
    try:
        with Transaction() as t:
            t.add_operation("DELETE FROM logs WHERE date < '2024-01-
01'")
            raise ValueError("Помилка під час виконання")
    except ValueError:
        print("Транзакція скасована")
    Вихід:
    Початок транзакції
    Відкат транзакції через помилку
    Транзакція скасована

    Вхід:
    with Transaction() as t:
        t.add_operation("CREATE TABLE test (id INT)")
        # Ніяких помилок
    print("Створено таблицю")
    Вихід:
    Початок транзакції
    Створено таблицю
    Фіксація транзакції: ['CREATE TABLE test (id INT)']
```

Завдання 3.2. Розумний ітератор для обходу графа

Створіть клас GraphIterator, який реалізує ітератор для обходу графа в ширину (BFS). Граф представлений у вигляді словника суміжності. Ітератор має приймати початкову вершину та повертати вершини в порядку обходу BFS. Реалізуйте методи __iter__, __next__, а також

__len__ (кількість пройдених вершин) та __str__ (поточний стан черги).

```
Вхід:
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
bfs = GraphIterator(graph, 'A')
for vertex in bfs:
    print(vertex, end=' ')
Вихід: A B C D E F
Вхід:
graph = {1: [2, 3], 2: [1, 4], 3: [1], 4: [2]}
bfs = GraphIterator(graph, 1)
print(next(bfs))
print(next(bfs))
print(next(bfs))
print(f"Пройдено вершин: {len(bfs)}")
Вихід:
1
2
3
Пройдено вершин: 3

Вхід:
graph = {'X': ['Y'], 'Y': ['X', 'Z'], 'Z': ['Y']}
bfs = GraphIterator(graph, 'X')
print(list(bfs))
Вихід: ['X', 'Y', 'Z']
```

Завдання 3.3. Повнофункціональний клас для роботи з матрицями

Створіть клас Matrix, який реалізує повноцінну роботу з матрицями довільного розміру. Клас повинен включати:

1. Інкапсуляцію даних (приватне зберігання матриці)

2. Властивості для розмірів матриці

3. Перевантаження арифметичних операторів (+, -, *, @ для матричного множення)

4. Методи порівняння (==, !=)
5. Методи контейнера (доступ до елементів через [i][j], [i, j] або [i, j:])
6. Контекстний менеджер для тимчасової зміни матриці
7. Ітератор для обходу елементів по рядках

```
Вхід:
m1 = Matrix([[1, 2], [3, 4]])
m2 = Matrix([[5, 6], [7, 8]])
print(m1 + m2)
Вихід: Matrix([[6, 8], [10, 12]])


Вхід:
m = Matrix([[1, 2, 3], [4, 5, 6]])
print(m.rows, m.cols)  # властивості
print(m[1, 2])  # доступ до елемента
m[0, 0] = 10    # зміна елемента
print(m)
Вихід:
2 3
6
Matrix([[10, 2, 3], [4, 5, 6]])

Вхід:
m = Matrix([[1, 2], [3, 4]])
with m.temporary_change() as tmp:
    tmp[0, 0] = 100
    tmp[1, 1] = 400
    print("Всередині контексту:", tmp)
print("Поза контекстом:", m)
Вихід:
Всередині контексту: Matrix([[100, 2], [3, 400]])
Поза контекстом: Matrix([[1, 2], [3, 4]])

Вхід:
m = Matrix([[1, 2], [3, 4]])
for row in m:
    print(row)
Вихід:
[1, 2]
[3, 4]
```

6. Питання для самоперевірки

1. Що таке інкапсуляція в об’єктно-орієнтованому програмуванні та які механізми її реалізації існують у Python?

2. Яка різниця між захищеними (_attribute) та приватними (__attribute) атрибутами в Python? Навіщо потрібен механізм name mangling?

3. Поясніть призначення декоратора @property. Чим властивості кращі за звичайні методи getter/setter?

4. Які переваги дає використання властивостей (@property) порівняно з прямим доступом до атрибутів класу?

5. Для чого використовуються магічні методи __str__ та __repr__? Яка між ними різниця та коли кожен з них викликається?

6. Як реалізувати перевантаження оператора додавання (+) для власного класу? Наведіть приклад сигнатури методу.

7. Що повинен повертати метод __len__? У яких випадках його реалізація є корисною?

8. Які магічні методи потрібно реалізувати, щоб об’єкт вашого класу міг використовуватися в конструкції for item in my_object?

9. Які методи необхідно реалізувати для створення контекстного менеджера? Поясніть призначення параметрів методу __exit__.

10. Чому в Python не можна створити справді приватні атрибути? Який підхід до інкапсуляції прийнято в Python-спільноті?

11. Як реалізувати властивість, доступну тільки для читання? А тільки для запису?

12. Що станеться, якщо для класу реалізувати тільки __getitem__, але не реалізувати __setitem__? Чи можна буде змінювати елементи за індексом?

13. Як реалізувати операцію obj1 == obj2, якщо об’єкти належать до різних класів? Що має повертати метод __eq__ у цьому випадку?

14. Чи можна створити клас, який буде одночасно ітератором та ітерабельним об’єктом? Як це реалізувати?

15. Що таке протокол менеджера контексту та які альтернативні способи його реалізації існують (окрім __enter__/__exit__)?

16. Як працює оператор in з об’єктами користувацьких класів? Який магічний метод для цього потрібно реалізувати?

17. Що таке "винятково-безпечний" контекстний менеджер? Як правильно обробляти помилки в методі __exit__?

18. Як реалізувати підтримку зрізів (slices) у методі __getitem__? Наведіть приклад.

19. Чому методи порівняння (__lt__, __gt__ тощо) часто реалізують через @functools.total_ordering? Які переваги цього підходу?

20. Які можуть виникнути проблеми з продуктивністю при надмірному використанні магічних методів? Наведіть приклади.

21. У яких реальних задачах використання контекстних менеджерів є найбільш доречним? Наведіть приклади з ваших лабораторних робіт.

22. Як би ви реалізували клас для роботи з грошовими сумами, щоб уникнути проблем з округленням та забезпечити коректні арифметичні операції?

23. Чи є сенс використовувати інкапсуляцію у невеликих скриптах? Коли її використання стає обов'язковим?

24. Як би ви пояснили різницю між __str__ та __repr__ початківцю програмісту на прикладі з реального життя?

25. Які тести варто писати для класів, які реалізують магічні методи? Наведіть приклади тестових випадків.

ЛАБОРАТОРНА РОБОТА №17
Графічна бібліотека Turtle та візуалізація даних з Matplotlib

1. Мета

Опанувати навички візуалізації в Python з використанням графічної бібліотеки Turtle для створення програмних векторних зображень та бібліотеки Matplotlib для побудови наукових графіків і діаграм. Мета включає розуміння основних принципів графічного програмування (система координат, керування пером, колірні моделі) та навички представлення та аналізу даних у графічній формі.

2. Завдання

1. Навчитися створювати прості та складні геометричні фігури за допомогою Turtle.
2. Застосувати цикли та функції для генерації графічних патернів і візерунків.
3. Оволодіти техніками заповнення фігур кольором.
4. Навчитися будувати основні типи графіків (лінійні, стовпчасті, кругові, діаграми розсіювання) з використанням Matplotlib.
5. Опанувати налаштування зовнішнього вигляду графіків (осі, легенда, заголовки, мітки).
6. Навчитися створювати складні композиції з декількох графіків (підграфіки).
7. Отримати навички читання даних із файлів та їх подальшої візуалізації.

3. Короткі теоретичні відомості

Візуалізація є ключовим етапом як у пізнанні основ програмування, так і в аналізі даних. У Python для цих цілей існує безліч бібліотек, серед яких для початкового навчання ідеально підходять **Turtle** (для процедурної графіки) та **Matplotlib** (для побудови діаграм).

3.1. Графічна бібліотека Turtle

Turtle – це графічна бібліотека, яка імітує малювання «черепашкою», що пересувається екраном. Керуючи її рухами, можна створювати складні зображення. Це чудовий інструмент для розуміння циклів, функцій та геометричних перетворень.

Основні концепції:

- **Полотно (Canvas):** графічна область для малювання. Має декартову систему координат із центром (0,0).
- **Черепашка (Turtle):** об'єкт, який має позицію (x, y) та напрямок (кут). Має два стани: перо піднято (penup()) – переміщення без сліду, перо опущено (pendown()) – малювання лінії.
- **Керування пером:** властивості пера: колір (color()), товщина (width()), швидкість (speed()).

```
import turtle

# Ініціалізація екрану та черепашки
screen = turtle.Screen()
screen.title("Лабораторна робота")
t = turtle.Turtle()
t.speed(5) # швидкість від 1 (повільно) до 10 (швидко)

# Малюємо квадрат
for _ in range(4):
    t.forward(100) # Рух вперед на 100 пікселів
    t.left(90)     # Поворот наліво на 90 градусів

# Малюємо коло
t.penup()
t.goto(150, 0) # Переміщаємо черепашку без малювання
t.pendown()
t.color("blue")
t.circle(50) # Радіус 50 пікселів

# Завершення
t.hideturtle()
screen.mainloop()
```

Використання циклів для складних фігур:

Багатокутники та зірки легко створюються шляхом повторення однакових дій.

```
# Малювання шестикутника
for _ in range(6):
    t.forward(70)
    t.left(60)
# Малювання п'ятипроменевої зірки
t.color("red")
for _ in range(5):
    t.forward(100)
```

```
    t.right(144) # Кут, що утворює зірку
```

Заповнення кольором: фігуру можна заповнити кольором за допомогою команд begin_fill() та end_fill().

```
t.color("green", "yellow") # колір лінії, колір заповнення
t.begin_fill()
for _ in range(4):
    t.forward(80)
    t.right(90)
t.end_fill()
```

3.2. Бібліотека Matplotlib

Matplotlib – це потужна бібліотека для створення статичних, анімованих та інтерактивних візуалізацій. Основним модулем для побудови графіків є pyplot.

Основні концепції:

- **Фігура (Figure):** вікно або сторінка, на якій розміщуються графіки. Може містити кілька незалежних областей малювання.
- **Вісь (Axes):** область малювання на фігурі, що містить сам графік, осі, легенду тощо. Одна фігура може містити кілька осей (підграфіків).
- **Об'єктно-орієнтований vs функціональний підхід:** Matplotlib дозволяє працювати як через функції у стилі MATLAB (plt.plot()), так і через явне створення об'єктів Figure та Axes, що є більш гнучким.

Приклади базових графіків:

```
import matplotlib.pyplot as plt
import numpy as np

# Лінійний графік
x = np.linspace(0, 10, 100) # 100 точок від 0 до 10
y = np.sin(x)
plt.figure(figsize=(8, 5)) # Створюємо фігуру розміром 8x5 дюймів
plt.plot(x, y, label='sin(x)', color='orange', linewidth=2)
plt.title('Лінійний графік функції sin(x)')
plt.xlabel('Вісь X')
plt.ylabel('Вісь Y')
plt.grid(True)
plt.legend()
plt.show()
```

```
    # Стовпчаста діаграма
    categories = ['A', 'B', 'C', 'D']
    values = [15, 24, 12, 30]
    plt.bar(categories, values, color='skyblue')
    plt.title('Стовпчаста діаграма')
    plt.show()

    # Кругова діаграма
    sizes = [25, 35, 20, 20]
    labels = ['Категорія 1', 'Категорія 2', 'Категорія 3', 'Категорія
4']
    plt.pie(sizes,        labels=labels,        autopct='%1.1f%%',
startangle=90)
    plt.axis('equal') # Робить діаграму круглою
    plt.title('Кругова діаграма')
    plt.show()

    # Діаграма розсіювання
    x_scatter = np.random.rand(50)
    y_scatter = np.random.rand(50)
    colors = np.random.rand(50)
    sizes_scatter = 1000 * np.random.rand(50)
    plt.scatter(x_scatter,  y_scatter,  c=colors,  s=sizes_scatter,
alpha=0.6)
    plt.title('Діаграма розсіювання')
    plt.show()
```

Підграфіки (subplots):

Підграфіки дозволяють розмістити кілька графіків на одній фігурі.

```
    fig, axes = plt.subplots(2, 2, figsize=(10, 8)) # Сітка 2x2

    axes[0, 0].plot(x, y)
    axes[0, 0].set_title('Графік 1')
    axes[0, 1].bar(categories, values)
    axes[0, 1].set_title('Графік 2')

    axes[1, 0].pie(sizes, labels=labels)
    axes[1, 0].set_title('Графік 3')

    axes[1, 1].scatter(x_scatter, y_scatter)
    axes[1, 1].set_title('Графік 4')

    plt.tight_layout() # Автоматичне відстачання
```

```
plt.show()
```

Ці теоретичні основи є мінімально необхідним фундаментом для успішного виконання завдань лабораторної роботи.

4. Методичні рекомендації

Нижче наведено розв'язок типових задач, які демонструють застосування бібліотек Turtle та Matplotlib. Рекомендується вивчити кожен приклад, зрозуміти логіку, а потім переходити до самостійного виконання завдань лабораторної роботи.

Задача 1. Малювання кольорового багатокутника за заданою кількістю сторін

Написати програму, яка запитує у користувача ціле число n (кількість сторін, n >= 3) та довжину сторони length. Програма має намалювати правильний n-кутник випадковим кольором.

```
import turtle
import random

# Ініціалізація
screen = turtle.Screen()
screen.title("Багатокутник")
t = turtle.Turtle()
t.speed(7)
t.width(2)

# Введення даних
n = int(turtle.textinput("Введення", "Введіть кількість сторін (>=3): "))
length = float(turtle.textinput("Введення", "Введіть довжину сторони (пікселі): "))

# Генерація випадкового кольору для лінії
t.color(random.random(), random.random(), random.random())  # RGB у діапазоні 0-1

# Обчислення кута повороту для правильного n-кутника
angle = 360 / n

# Малювання фігури
for _ in range(n):
    t.forward(length)
```

```
        t.right(angle)

    # Завершення
    t.hideturtle()
    screen.mainloop()
```

Приклад вхідних та вихідних даних:

Вхід: n = 5, length = 80 → Вихід: На екрані малюється п’ятикутник зі стороною 80 пікселів випадковим кольором.

Вхід: n = 8, length = 50 → Вихід: На екрані малюється восьмикутник зі стороною 50 пікселів випадковим кольором.

Вхід: n = 3, length = 120 → Вихід: На екрані малюється трикутник зі стороною 120 пікселів випадковим кольором.

Коментарі:

1. Функція random.random() повертає випадкове дробове число від 0 до 1, що дозволяє генерувати мільйони різних відтінків.

2. Ключова формула для малювання правильного багатокутника: angle = 360 / n. Це зовнішній кут повороту черепашки.

3. Використання turtle.textinput() є зручним та інтуїтивним способом організації введення даних у графічних програмах з Turtle.

Задача 2. Малювання суцільної (filled) зірки з заданою кількістю променів

Написати програму, яка запитує у користувача непарну кількість променів points (наприклад, 5, 7, 9) та довжину променя length. Програма має намалювати та заповнити одним кольором зірку з вказаною кількістю променів.

```
import turtle

# Ініціалізація
screen = turtle.Screen()
t = turtle.Turtle()
t.speed(8)
t.width(1)

# Введення даних
```

```
    points = int(turtle.textinput("Зірка", "Введіть непарну кількість променів (напр., 5, 7, 9): "))
    length = float(turtle.textinput("Зірка", "Введіть довжину променя: "))

    # Встановлення кольорів: темно-синя лінія, світло-блакитне заповнення
    t.color("navy", "lightcyan")

    # Обчислення кута для побудови зірки
    angle = 180 - (180 / points)

    # Малювання з заповненням
    t.begin_fill()
    for _ in range(points):
        t.forward(length)
        t.right(angle)
    t.end_fill()

    t.hideturtle()
    screen.mainloop()
```

Приклад вхідних та вихідних даних:

Вхід: points = 5, length = 150 → Вихід: На екрані малюється та заповнюється класична п’ятипроменева зірка.

Вхід: points = 7, length = 100 → Вихід: На екрані малюється семипроменева зірка.

Вхід: points = 9, length = 80 → Вихід: На екрані малюється дев’ятипроменева зірка.

Коментарі:

1. Зірка можлива лише з непарною кількістю променів. Для парного числа вийде просто з’єднаний багатокутник.

2. Формула для кута повороту angle = 180 - (180 / points) є стандартною для таких зірок.

3. Команди begin_fill() та end_fill() завжди використовуються разом. Усе, що намальовано між ними, буде заповнено кольором, вказаним другим аргументом у t.color().

Задача 3. Малювання концентричних квадратів за допомогою циклу.

Написати програму, яка малює 10 квадратів зі спільним центром. Сторона кожного наступного квадрата повинна бути на 20 пікселів

більшою за попередній. Кольри квадратів повинні чергуватися між двома заданими.

```
import turtle

# Ініціалізація
screen = turtle.Screen()
t = turtle.Turtle()
t.speed(10)

# Початкові параметри
start_size = 20
step = 20
num_squares = 10
colors = ["indigo", "gold"]  # Два кольори для чергування

# Цикл для малювання квадратів
for i in range(num_squares):
    # Вибір кольору залежно від парності номера квадрата (i)
    current_color = colors[i % len(colors)]
    t.color(current_color)
    # Зміщення черепашки у стартову позицію для нового
квадрата
    offset = (start_size + i * step) // 2
    t.penup()
    t.goto(-offset, -offset)  # Нижній лівий кут квадрата
    t.pendown()

    # Малювання одного квадрата
    side = start_size + i * step
    for _ in range(4):
        t.forward(side)
        t.left(90)

t.hideturtle()
screen.mainloop()
```

Приклад вхідних та вихідних даних:

Вхід: start_size = 20, step = 20, colors = ["indigo", "gold"] → Вихід: На екрані з’являється 10 квадратів з центром у (0,0), зростаючого розміру з кольорами indigo та gold, що чергуються.

Вхід: start_size = 10, step = 15, colors = ["red", "black"] → Вихід: 10 квадратів, що ростуть швидше, чергуючи червоний та чорний кольори.

Коментарі:

1. Ключовий момент – розрахунок позиції (goto) для кожного нового квадрата так, щоб їх центри співпадали. Для квадрата, що малюється з лівого нижнього кута, це (-side/2, -side/2).

2. Оператор і % len(colors) (остача від ділення) – елегантний спосіб циклічного чергування елементів списку.

3. Обов'язково використовуйте penup() та pendown() при переміщенні до стартової точки, щоб не залишати зайвих ліній.

Задача 4. Побудова лінійного графіка з двох функцій та легендою

Побудувати графіки функцій y = sin(x) та y = cos(x) на інтервалі [0, 4π]. Додати легенду, сітку, підписи осей та заголовок.

```
import matplotlib.pyplot as plt
import numpy as np
# Генерація даних
x = np.linspace(0, 4 * np.pi, 200)  # 200 точок від 0 до 4π
y1 = np.sin(x)
y2 = np.cos(x)

# Створення графіка
plt.figure(figsize=(10, 6))

# Побудова двох ліній з різними кольорами та мітками
plt.plot(x, y1, label='y = sin(x)', color='blue', linewidth=2)
plt.plot(x, y2, label='y = cos(x)', color='red', linestyle='--
', linewidth=2)

# Форматування
plt.title('Графіки функцій синуса та косинуса', fontsize=14)
plt.xlabel('x (радіани)', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, linestyle=':', alpha=0.7)  # Пунктирна сітка з
прозорістю
plt.legend(fontsize=12, loc='upper right')  # Додавання
легенди

# Встановлення меж по осі X для кращої читабельності
plt.xlim(0, 4 * np.pi)

# Відображення графіка
```

```
plt.tight_layout()
plt.show()
```

Приклад вхідних та вихідних даних:

Вхід: Інтервал [0, 4π], функції sin(x), cos(x) → Вихід: З'являється вікно з чітко окресленим графіком, де синя суцільна лінія – sin(x), червона пунктирна – cos(x).

Коментарі:

1. np.linspace() – фундаментальна функція для створення масиву рівномірно розподілених значень аргументу.

2. Аргумент label у plt.plot() є обов'язковим для появи легенди. Сама легенда додається командою plt.legend().

3. Параметри linestyle (стиль лінії) та alpha (прозорість) значно покращують вигляд та читабельність графіка.

4. plt.tight_layout() автоматично підлаштовує розташування елементів графіка, щоб уникнути накладень.

Задача 5. Створення групуваннях стовпчастих діаграм для порівняння даних

Побудувати групування стовпчасту діаграму, яка порівнює успішність студентів (балли) з двох груп ("КН-101", "КН-102") за трьома предметами ("Алгоритмізація", "Математика", "Дискретні структури").

```
import matplotlib.pyplot as plt
import numpy as np

# Дані
subjects = ['Алгоритмізація', 'Математика', 'Дискретні структури']
group_kn101 = [85, 78, 92]
group_kn102 = [88, 82, 80]

bar_width = 0.35  # Ширина одного стовпця
index = np.arange(len(subjects))  # Позиції груп предметів на осі X: [0, 1, 2]

# Створення фігури та осей
fig, ax = plt.subplots(figsize=(9, 6))

# Побудова двох наборів стовпців, зсунутих один від одного
bars1 = ax.bar(index - bar_width/2, group_kn101, bar_width,
```

```
                    label='КН-101',                    color='steelblue',
edgecolor='black')
    bars2 = ax.bar(index + bar_width/2, group_kn102, bar_width,
                   label='КН-102', color='lightcoral', edgecolor='black')

    # Налаштування осей та вигляду
    ax.set_title('Порівняння успішності студентських груп',
fontsize=14, pad=15)
    ax.set_xlabel('Предмети', fontsize=12)
    ax.set_ylabel('Середній бал', fontsize=12)
    ax.set_xticks(index)   # Встановлюємо мітки на осі X у позиціях
index
    ax.set_xticklabels(subjects)      # Підписуємо їх назвами
предметів
    ax.legend()
    ax.grid(axis='y', linestyle='--', alpha=0.6)

    # Додавання числових значень над стовпцями
    def autolabel(bars):
        for bar in bars:
            height = bar.get_height()
            ax.annotate(f'{height}',
                        xy=(bar.get_x()+bar.get_width()/2,
height),
                        xytext=(0, 3),    # Зміщення тексту на 3
пункти вгору
                        textcoords="offset points",
                        ha='center', va='bottom')
    autolabel(bars1)
    autolabel(bars2)

    plt.tight_layout()
    plt.show()
```

Приклад вхідних та вихідних даних:

Вхід: Дані для двох груп з трьох предметів → Вихід: З'являється діаграма, де для кожного предмета стоять поруч два стовпці різного кольору для кожної групи. Над кожним стовпцем вказано числове значення.

Коментарі:

1. np.arange(len(subjects)) створює базові індекси для позиціювання стовпців.

2. Стовпці груп зсуваються на -bar_width/2 та +bar_width/2 відносно центра кожної категорії, щоб стояти поруч.

3. Функція autolabel демонструє, як можна ітерувати по об'єктах стовпців (bars) та додавати до них текст за допомогою annotate.

4. grid(axis='y') додає сітку тільки по вертикальній осі, що є стандартом для стовпчастих діаграм.

Задача 6. Читання даних з CSV-файлу та побудова діаграми розсіювання

Припустимо, у файлі student_data.csv містяться стовпчики hours_studied та exam_score. Необхідно прочитати ці дані та побудувати діаграму розсіювання, щоб візуалізувати залежність між годинами навчання та іспитовим балом.

```
    import matplotlib.pyplot as plt
    import pandas as pd  # Бібліотека Pandas ідеально підходить для роботи з табличними даними

    # Читання даних з файлу CSV
    try:
        # Припустимо, файл має такі стовпці: hours_studied, exam_score
        data = pd.read_csv('student_data.csv')
    except FileNotFoundError:
        print("Помилка. Файл 'student_data.csv' не знайдено.")
        # Створення прикладних даних для демонстрації
        import numpy as np
        np.random.seed(42)
        hours = np.random.normal(10, 3, 50)
        scores = 2.5 * hours + 50 + np.random.normal(0, 5, 50)
        data = pd.DataFrame({'hours_studied': hours, 'exam_score': scores})
        print("Створено демонстраційний набір даних.")

    # Витягуємо необхідні стовпці у змінні
    x = data['hours_studied']
    y = data['exam_score']

    # Побудова діаграми розсіювання
    plt.figure(figsize=(8, 6))
```

```
    plt.scatter(x, y, alpha=0.6, edgecolors='w', s=80, c='green')
# s - розмір маркера

    # Додавання лінії тренду (полінома 1-го ступеня - пряма)
    import numpy as np
    z = np.polyfit(x, y, 1)  # Обчислення коефіцієнтів прямої
    p = np.poly1d(z)         # Створення функції прямої
    plt.plot(x, p(x), "r--", linewidth=1.5, label=f'Лінія тренду:
y={z[0]:.2f}x+{z[1]:.2f}')

    # Форматування графіка
    plt.title('Залежність балу іспиту від часу навчання',
fontsize=14)
    plt.xlabel('Години навчання', fontsize=12)
    plt.ylabel('Бал на іспиті', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Приклад вхідних та вихідних даних:

Вхід: Файл student_data.csv з двома стовпчиками числових даних. → Вихід: Діаграма розсіювання, де кожна точка – студент, а червона пунктирна лінія показує загальну тенденцію.

Якщо файл відсутній: Програма створить демонстраційні дані та побудує графік на їх основі.

Коментарі:

1. Бібліотека Pandas (pd.read_csv()) надає найпростіший та найпотужніший спосіб читання структурованих даних.

2. Обробка винятків (try-except) робить програму стійкою до відсутності файлу, що дуже важливо в реальних умовах.

3. Параметр alpha (прозорість) у scatter() допомагає оцінити щільність розподілу точок.

4. Функції np.polyfit() та np.poly1d() дозволяють легко додати лінію тренду для якісного аналізу кореляції.

Задача 7. Створення фігури з чотирма підграфіками різних типів

Створити фігуру (figure) розміром 10x8 дюймів, яка містить чотири підграфіки (axes), розташовані у сітці 2x2. На кожному побудувати різний тип графіку з одних і тих же або різних даних.

```
    import matplotlib.pyplot as plt
    import numpy as np

    # Генерація спільних даних
    x = np.linspace(0, 10, 30)
    categories = ['A', 'B', 'C', 'D', 'E']
    values_bar = np.random.randint(1, 20, len(categories))
    values_pie = np.abs(np.random.randn(5)) * 100
    values_pie = values_pie / values_pie.sum() * 100

    # Створення фігури та сітки підграфіків 2x2
    fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))
    fig.suptitle('Композитна візуалізація даних', fontsize=16,
y=1.02)

    # 1. Лінійний графік з маркерами (верхній лівий)
    axs[0, 0].plot(x, np.sin(x), 'o-', label='sin(x)')
    axs[0, 0].plot(x, np.cos(x), 's--', label='cos(x)')
    axs[0, 0].set_title('Лінійні графіки')
    axs[0, 0].set_xlabel('x')
    axs[0, 0].set_ylabel('y')
    axs[0, 0].legend()
    axs[0, 0].grid(True)

    # 2. Стовпчаста діаграма (верхній правий)
    bars = axs[0, 1].bar(categories, values_bar, color='teal')
    axs[0, 1].set_title('Стовпчаста діаграма')
    axs[0, 1].set_xlabel('Категорія')
    axs[0, 1].set_ylabel('Значення')
    # Додаємо значення на стовпці
    for bar in bars:
        height = bar.get_height()
        axs[0, 1].text(bar.get_x() + bar.get_width()/2., height,
                       f'{int(height)}', ha='center',
va='bottom')

    # 3. Кругова діаграма (нижній лівий)
    wedges, texts, autotexts = axs[1, 0].pie(values_pie,
labels=categories, autopct='%1.1f%%',
```

```
                                                    startangle=90,
explode=(0, 0.1, 0, 0, 0))
    axs[1, 0].set_title('Кругова діаграма')
    # Змінюємо колір тексту на секторах для контрасту
    for autotext in autotexts:
        autotext.set_color('white')
        autotext.set_fontweight('bold')

    # 4. Діаграма розсіювання з кольоровою шкалою (нижній правий)
    x_scat = np.random.rand(50) * 10
    y_scat = np.random.rand(50) * 100
    colors_scat = np.random.rand(50)
    size_scat = np.random.rand(50) * 200
    scatter = axs[1, 1].scatter(x_scat, y_scat, c=colors_scat,
s=size_scat, alpha=0.6, cmap='viridis')
    axs[1, 1].set_title('Діаграма розсіювання')
    axs[1, 1].set_xlabel('Ознака X')
    axs[1, 1].set_ylabel('Ознака Y')
    fig.colorbar(scatter, ax=axs[1, 1], label='Інтенсивність')  #
Додаємо колірну шкалу

    # Автоматичне регулювання відстаней між графіками
    plt.tight_layout()
    plt.show()
```

Приклад вхідних та вихідних даних:

Вхід: Скрипт генерує дані випадково, але детерміновано (завдяки seed). → Вихід: З'являється одне велике вікно, розділене на 4 частини, у кожній з яких представлений різний тип графіку з повним форматуванням.

Коментарі:

1. plt.subplots(nrows, ncols) – найважливіша функція для створення сітки підграфіків. Вона повертає кортеж (figure, axes_array).
2. Доступ до кожного окремого підграфіка здійснюється через індексацію масиву axs, напр. axs[0, 0].
3. Загальний заголовок для всієї фігури задається через fig.suptitle().
4. plt.tight_layout() є критично важливим при роботі з декількома підграфіками, щоб уникнути накладання текстів та міток.
5. Додавання колірної шкали (colorbar) до діаграми розсіювання значно покращує інформативність графіка.

Типові помилки і шляхи їх усунення

1. **ModuleNotFoundError: No module named ‘matplotlib’ / ‘turtle’**

Причина. Бібліотека не встановлена у середовищі Python.

Рішення. Встановіть її за допомогою менеджера пакетів pip у терміналі/командному рядку: pip install matplotlib. Turtle зазвичай входить у стандартну бібліотеку Python.

2. **Графік Turtle відкривається і миттєво закривається**

Причина. Скрипт завершує виконання, і вікно автоматично закривається.

Рішення. Додайте в кінець програми turtle.done() або screen.mainloop() (для об’єктно-орієнтованого підходу). У Matplotlib аналог – plt.show().

3. **Всі графіки Matplotlib малюються на одній осі або в одному вікні**

Причина. Виклики plt.plot(), plt.bar() тощо додаються до поточної активної осі.

Рішення. Для нового вікна використовуйте plt.figure() перед побудовою. Для підграфіків явно створюйте осі за допомогою fig, ax = plt.subplots() і потім працюйте з ax.plot().

4. **Неправильна побудова фігур у Turtle (не та форма)**

Причина. Неправильно розрахований кут повороту для багатокутника або зірки.

Рішення. Згадайте геометрію. Сума зовнішніх кутів будь-якого опуклого багатокутника = 360°. Для зірки з непарною кількістю променів n використовуйте кут = 180 - 180/n.

5. **Зайві лінії при переміщенні черепашки**

Причина. Забули підняти перо командою penup() перед викликом goto() або іншим переміщенням.

Рішення. Чітко контролюйте стани пера: penup() – "переїзд", pendown() – "малювання".

6. **Налаштування Matplotlib (легенда, заголовок) не відображаються**

Причина. Команди форматування (як plt.title(), plt.legend()) викликані **після** plt.show().

Рішення. Усі команди для побудови та форматування графіка повинні йти **до** команди plt.show().

Корисні поради

1. **Іменуйте змінні зрозуміло.** Замість x1, y1 використовуйте hours, scores або angle, side_length. Це полегшує читання та налагодження коду.

2. **Експериментуйте з аргументами.** Змінюйте color, linestyle, marker, alpha у Matplotlib та speed(), width() у Turtle, щоб краще зрозуміти їх вплив і знайти оптимальний вигляд.

3. **Розбивайте складні візерунки на прості частини.** Складний патерн часто є комбінацією простих фігур, намальованих у циклі зі зміщенням або поворотом. Спробуйте спочатку намалювати один елемент.

4. **Використовуйте функції для повторюваного коду.** Якщо малюєте одну й ту ж фігуру багато разів (наприклад, сніжинку або дерево), оберніть її у функцію draw_snowflake(angle, size). Це спростить код і зробить його модульним.

5. **Для Matplotlib: почніть з об'єктно-орієнтованого інтерфейсу (ООП).** Навіть для простих графіків використовуйте fig, ax = plt.subplots(), а потім ax.plot(...). Цей підхід є більш гнучким і передбачуваним, особливо при роботі з декількома графіками.

6. **Зберігайте ваші графіки.** Використовуйте plt.savefig('my_plot.png', dpi=300, bbox_inches='tight') **перед** plt.show(), щоб експортувати результат у файл з високою якістю для звіту або презентації.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Кольоровий квадрат із променем

Намалюйте квадрат зі стороною 120 пікселів. Кожна сторона квадрата повинна бути іншого кольору (наприклад, червона, зелена, синя, чорна). Від центру квадрата до кожного з його кутів має йти лінія (промінь) того ж кольору, що й відповідна сторона.

Вхід: Програма не має вхідних даних.

Вихід: Графічне вікно із зображенням: квадрат з чотирьох різнокольорових сторін та чотири лінії від центру до кутів.

Завдання 1.2. Шахівниця Turtle

За допомогою черепашки намалюйте шахівницю розміром 4x4 клітинки. Розмір однієї клітинки – 40 пікселів. Клітинки повинні чергуватися двома кольорами (наприклад, сірий та білий). Фон вікна встановіть у світлий колір.

Вхід: Програма не має вхідних даних.

Вихід: Графічне вікно із зображенням шахівниці 4x4.

Завдання 1.3. Сходинки до неба

Намалюйте сходи, що ведуть зліва направо вгору. Перша сходинка має висоту 10 пікселів і ширину 20 пікселів. Кожна наступна сходинка має висоту на 5 пікселів більшу за попередню, а ширина залишається 20 пікселів. Всього намалюйте 8 сходинок. Кожна сходинка – іншого відтінку сірого (від темного до світлого).

Вхід: Програма не має вхідних даних.

Вихід: Графічне вікно із зображенням сходів із 8 сходинок, що зростають у висоті та змінюють колір.

Завдання 1.4. Палетка кольорів

Намалюйте три однакових квадрати (сторона 60 пікселів), розташованих горизонтально. Заповніть перший квадрат градієнтом від червоного до жовтого (по горизонталі), другий – від зеленого до блакитного, третій – від фіолетового до рожевого. *Підказка: використовуйте дуже багато тонких прямокутників.*

Вхід: Програма не має вхідних даних.

Вихід: Графічне вікно із трьома квадратами, кожен із яких має плавний горизонтальний градієнт заливки.

Завдання 1.5. Спираль зі змінним кроком

Намалюйте спіраль Архімеда. Початкова довжина кроку (відстань між витками) – 2 пікселі. Після кожного повного кола (360 градусів) крок збільшується на 1 піксель. Намалюйте 8 повних кіл. Колір лінії повинен плавно змінюватись від синього на початку до червоного в кінці.

Вхід: Програма не має вхідних даних.

Вихід: Графічне вікно із кольоровою спіраллю, що розширюється.

Завдання 1.6. Графік квадратичної функції

Побудуйте графік функції $y = x^2 - 4$ на інтервалі x від -5 до 5. Вісь X позначте "Аргумент", вісь Y – "Значення функції". Додайте заголовок "Графік квадратичної функції". Обов'язково нанесіть сітку.

Вхід: Програма не має вхідних даних.

Вихід: Вікно з лінійним графіком параболи.

Завдання 1.7. Діаграма успішності групи

Створіть стовпчасту діаграму, яка відображає кількість студентів, які отримали певну оцінку за іспит. Дані: 'Незадовільно' (2): 1 студент, 'Задовільно' (3): 4 студенти, 'Добре' (4): 8 студентів, 'Відмінно' (5): 5 студентів. Кожен стовпець має бути іншого кольору. Додайте числові мітки поверх стовпців.

Вхід: Програма не має вхідних даних.

Вихід: Вікно з кольоровою стовпчастою діаграмою з 4 стовпцями та числами на них.

Завдання 1.8. Кругова діаграма витрат

Побудуйте кругову діаграму з відділенням одного сектора для відображення бюджету студента. Дані: 'Їжа': 2500 грн, 'Навчання': 5000 грн, 'Розваги': 1500 грн, 'Транспорт': 800 грн, 'Інше': 1200 грн. Сектор з найбільшими витратами ('Навчання') має бути трохи винесеним (explode). На діаграмі обов'язково мають бути відсотки.

Вхід: Програма не має вхідних даних.

Вихід: Вікно з круговою діаграмою, де один сектор винесено.

Завдання 1.9. Діаграма розсіювання випадкових точок

Згенеруйте два масиви (x та y) по 30 випадкових цілих чисел у діапазоні від 1 до 100. Побудуйте діаграму розсіювання. Розмір маркера (s) має бути пропорційний значенню x (помножте x на 5), а колір маркера (c) – пропорційний значенню y. Додайте підписи осей ("X випадкова величина", "Y випадкова величина").

Вхід: Програма не має вхідних даних (дані генеруються випадково).

Вихід: Вікно з діаграмою розсіювання кольорових точок різного розміру.

Завдання 1.10. Подвійна вісь Y

На одному графіку зі спільною віссю X побудуйте два графіки з різними осями Y. Для x від 0 до 10 з кроком 0.1: y1 = sin(x) (ліва вісь, синій колір), y2 = 100 * cos(x) (права вісь, червоний колір). Додайте легенду, сітку та заголовок "Синус і косинус (різні масштаби)".

Вхід: Програма не має вхідних даних.

Вихід: Вікно з двома графіками на одних осях X, але з двома осями Y (ліворуч і праворуч).

Частина 2. Середні завдання

Завдання 2.1. Квітка з n пелюстками

Напишіть програму, яка запитує у користувача ціле число n (кількість пелюсток, n >= 3). Програма має намалювати квітку, де кожна пелюстка є колом. Всі кола мають однаковий радіус (50 пікселів) і повинні торкатися один одного, утворюючи квітку навколо центру. Колір пелюсток має змінюватися плавно по колу (використовуйте кольорову модель HSV через colorsys або циклічну зміну RGB).

Вхід 1: n = 6 → Вихід: Квітка з 6 пелюстками-колами.

Вхід 2: n = 4 → Вихід: Квітка з 4 пелюстками-колами.

Вхід 3: n = 8 → Вихід: Квітка з 8 пелюстками-колами.

Завдання 2.2. Фрактал «Крива Коха» (сніжинка)

Реалізуйте малювання фрактала «Сніжинка Коха» рекурсивно. Глибину рекурсії level задає користувач (1 <= level <= 5). Намалюйте сніжинку на основі рівностороннього трикутника (довжина сторони початкового трикутника – 250 пікселів). Колір ліній на кожному рівні рекурсії має ставати світлішим (від темно-синього до світло-блакитного).

Вхід 1: level = 1 → Вихід: Рівносторонній трикутник.

Вхід 2: level = 2 → Вихід: Трикутник із заміненими середніми третинами сторін на «зірочки».

Вхід 3: level = 4 → Вихід: Деталізована сніжинка Коха.

Завдання 2.3. Анімований годинник Turtle

Намалюйте циферблат аналогового годинника (коло з розміткою 12 годин). Додайте стрілки: годинну, хвилинну та секундну. Запрограмуйте їх рух, використовуючи поточний системний час (модуль datetime).

Годинник має оновлюватися (рухатися) у реальному часі. Використовуйте turtle.ontimer() для анімації.

Вхід: Програма не має вхідних даних.

Вихід: Графічне вікно із циферблатом та трьома рухомими стрілками, які показують поточний час.

Завдання 2.4. Підграфік: порівняння функцій та їх похідних

Створіть фігуру з чотирма підграфіками (2x2). У верхньому рядку побудуйте графіки функцій sin(x) та cos(x) на інтервалі [0, 4π] у різних підграфіках. У нижньому рядку побудуйте графіки їхніх похідних (cos(x) та -sin(x) відповідно). Кожен підграфік має мати свій заголовок ("sin(x)", "cos(x)", "d(sin)/dx", "d(cos)/dx"), сітку та легенду. Використовуйте спільну вісь X для графіків у стовпчику.

Вхід: Програма не має вхідних даних.

Вихід: Вікно з чотирма підграфіками, що демонструють функції та їх похідні.

Завдання 2.5. Комбінована діаграма: стовпці та лінія

Побудуйте комбіновану діаграму. На основі даних: місяці = [‘Вер’, ‘Жов’, ‘Лис’, ‘Гру’, ‘Січ’, ‘Лют’], доходи = [120, 135, 118, 165, 145, 160] (стовпчаста діаграма, сині стовпці), витрати = [90, 110, 95, 140, 115, 125] (лінійний графік з червоними маркерами). Додайте дві легенди (для стовпців та лінії), числові мітки на стовпцях та заголовок "Бюджет студента (доходи та витрати)".

Вхід: Програма не має вхідних даних.

Вихід: Вікно з комбінованою діаграмою, де доходи показані стовпцями, а витрати – лінією.

Частина 3. Складні завдання

Завдання 3.1. Інтерактивний малювальник Turtle

Створіть інтерактивну програму-малювальник. Користувач повинен керувати черепашкою за допомогою клавіш:

- W, A, S, D – рух вперед, вліво, назад, вправо.
- Q, E – поворот вліво/вправо на 15 градусів.
- Пробіл – підняти/опустити перо (перемикач).
- C – очистити екран.

- 1, 2, 3 – змінити колір пера на червоний, зелений, синій.

- R – змінити колір заповнення на випадковий (при заповненні фігури).

- F – увімкнути/вимкнути режим автоматичного заповнення замкнутої фігури.

Програма має виводити коротку інструкцію на самому графічному вікні.

Вхід: Взаємодія користувача з програмою через клавіатуру.

Вихід: Інтерактивне графічне вікно, в якому користувач може малювати та керувати процесом.

Завдання 3.2. Фрактальне дерево Піфагора з параметрами

Напишіть програму для малювання фрактала «Дерево Піфагора». Користувач задає параметри через діалогові вікна: level (глибина рекурсії, 1-8), initial_length (довжина початкового стовбура), angle (кут відхилення гілок, 10-60 градусів), scale_factor (коефіцієнт скорочення довжини у дочірніх гілок, 0.5-0.9). Гілки різних рівнів мають різну товщину та колір (від темно-коричневого для стовбура до світло-зеленого для найменших гілок).

Вхід 1: level=4, length=100, angle=30, scale=0.7

Вхід 2: level=6, length=80, angle=45, scale=0.6

Вхід 3: level=3, length=120, angle=20, scale=0.8

Вихід: Графічне вікно із деревом Піфагора, збудованим за заданими параметрами.

Завдання 3.3. Аналіз та візуалізація даних з CSV (лог веб-сервера)

Завантажте дані з файлу server_log.csv (створити демо-файл, якщо немає). Файл містить колонки: timestamp, page, response_time_ms, user_id. Ваша програма повинна:

1. Прочитати файл за допомогою pandas.

2. Перетворити timestamp у тип datetime.

3. Побудувати фігуру з трьома підграфіками (3x1):

- **Верхній:** Лінійний графік середнього часу відповіді сервера (response_time_ms) по годинах доби (0-23). Позначте пікові години.

- **Середній:** Стовпчаста діаграма топ-10 найпопулярніших сторінок (page) за кількістю запитів.

- **Нижній:** Діаграма розсіювання залежності часу відповіді від порядкового номера запиту (індексу) з кольоровою шкалою за user_id, щоб побачити активність різних користувачів.

4. Зберегти отриману фігуру у файл server_analysis.png.

Вхід: Файл server_log.csv (або його відсутність з наступною генерацією демо-даних).

Вихід: Вікно з трьома графіками-підграфіками та файл server_analysis.png.

Питання для самоперевірки

1. Які дві основі функції керування станом пера в Turtle і для чого вони використовуються?

2. Як розрахувати кут повороту черепашки для малювання правильного n-кутника? Наведіть формулу.

3. Що відбудеться, якщо команди t.begin_fill() та t.end_fill() у Turtle розмістити в різних частинах програми або забути одну з них?

4. Яка різниця між об'єктно-орієнтованим (fig, ax = plt.subplots()) та процедурним (plt.plot()) підходами до побудови графіків у Matplotlib? Який підхід є кращим для складних візуалізацій і чому?

5. Як додати легенду до графіка в Matplotlib? Які дві обов'язкові умови для її появи?

6. Для чого використовується функція plt.tight_layout()? У яких випадках її виклик є критично важливим?

7. Як побудувати декілька незалежних графіків у одному вікні (підграфіки) за допомогою plt.subplots()? Поясніть на прикладі створення сітки 2x3.

8. Яким чином можна згенерувати масив рівномірно розподілених значень для побудови графіка функції? Наведіть приклад коду для інтервалу від -2 до 2 з кроком у 0.01.

9. Що відображає діаграма розсіювання (scatter plot) і чим вона відрізняється від лінійного графіка (line plot)? Наведіть приклади даних, для яких кожен тип графіка є оптимальним.

10. Які основні етапи роботи з даними для побудови графіка з файлу? Назвіть ключові бібліотеки та функції.

11. У чому полягає принцип рекурсії при малюванні фракталів (наприклад, сніжинки Коха)? Що є базовим випадком (умовою виходу) у такій рекурсивній функції?

12. Який принцип роботи анімації в Turtle з використанням turtle.ontimer()? Чому не можна використовувати для цього нескінченний цикл while True у поєднанні з turtle.update() без таймера?

13. Як змінити колір лінії в Turtle, використовуючи модель RGB з випадково згенерованими значеннями?

14. Які параметри графіка Matplotlib обов'язково потрібно налаштувати, щоб він був інформативним та читабельним (назвіть щонайменше 5)?

15. Як можна зберегти побудований графік Matplotlib у файл з високою роздільною здатністю? В який момент програми це потрібно робити – до чи після виклику plt.show()?

16. Що таке "лінія тренду" на діаграмі розсіювання та як її додати за допомогою NumPy?

17. Поясніть, як працює групування стовпців на стовпчастій діаграмі. Як розрахувати позиції для кожного стовпця групи за допомогою np.arange()?

18. Які є способи обробки ситуації, коли файл із даними для Matplotlib не знайдено? Наведіть приклад застосування блоку try-except для цієї мети.

19. Чому при створенні складних візерунків у Turtle код часто краще оформлювати у вигляді окремих функцій? Наведіть приклад.

20. Як можна зробити візуалізацію в Matplotlib більш доступною для людей із порушенням кольоросприйняття (колірна сліпота)? Назвіть практичні поради щодо вибору кольорових палітр.

ЛАБОРАТОРНА РОБОТА №18
Створення GUI з Tkinter та CustomTkinter

1. Мета

Опанувати базові навички створення графічного інтерфейсу користувача (GUI) у мові програмування Python за допомогою бібліотеки Tkinter та її сучасного розширення CustomTkinter. Студенти навчаться розміщати віджети (елементи управління) у вікні, обробляти події та створювати інтуїтивно зрозумілі програми з візуальним інтерфейсом.

2. Завдання

1. Вивчити базові принципи роботи з бібліотеками Tkinter та CustomTkinter.
2. Навчитися створювати головне вікно програми та налаштовувати його властивості.
3. Опанувати роботу з основними віджетами (кнопки, мітки, поля введення тощо).
4. Навчитися розміщувати віджети у вікні за допомогою геометр-менеджерів Pack, Grid та Place.
5. Реалізувати обробку подій (натискання кнопок, введення тексту).
6. Створити прості програми з GUI, які виконують обчислення або обробляють дані, введені користувачем.

3. Короткі теоретичні відомості

Графічний інтерфейс користувача (GUI – Graphical User Interface) дозволяє взаємодіяти з програмою через візуальні елементи (вікна, кнопки, списки тощо), що робить її зручною та доступною. У Python стандартною бібліотекою для створення простих GUI є **Tkinter**.

Tkinter – це інтерфейс до кросплатформної бібліотеки Tk. Він постачається разом з Python, не вимагає додаткового встановлення та є простим для вивчення.

CustomTkinter – це сучасне розширення Tkinter, яке пропонує покращені, стильніші та налаштовувані віджети з сучасним дизайном (подібним до темних/світлих режимів Windows 11, macOS). Воно значно покращує зовнішній вигляд програм при мінімальних змінах коду.

3.1. Основи Tkinter

Будь-який GUI на Tkinter починається з створення головного вікна – кореневого об'єкта (Tk).

```
import tkinter as tk

# Створення головного вікна
root = tk.Tk()
root.title("Моя перша програма")  # Заголовок вікна
root.geometry("400x300")          # Розмір вікна (ширина x висота)

# Запуск головного циклу обробки подій
root.mainloop()
```

Метод mainloop() запускає нескінченний цикл, який очікує дій користувача (подій) та обробляє їх.

3.2. Основні віджети (елементи управління):

- **Label** – мітка для відображення тексту або зображення.
- **Button** – кнопка, яка виконує дію при натисканні.
- **Entry** – однострокове поле для введення тексту.
- **Text** – багаторядкове поле для введення та відображення тексту.
- **Frame** – контейнер для групування інших віджетів.

Приклад створення та розміщення віджетів:

```
import tkinter as tk

def on_button_click():
    # Отримуємо текст з поля введення і встановлюємо його у мітку
    greeting = "Привіт, " + entry_name.get() + "!"
    label_greeting.config(text=greeting)

root = tk.Tk()
root.title("Привітання")

# Створення віджетів
label_name = tk.Label(root, text="Введіть ваше ім'я:")
entry_name = tk.Entry(root)
button_submit = tk.Button(root, text="Привітатися", command=on_button_click)
label_greeting = tk.Label(root, text="")
```

```
    # Розміщення віджетів за допомогою менеджера Grid (сітка)
    label_name.grid(row=0, column=0, padx=10, pady=10)
    entry_name.grid(row=0, column=1, padx=10, pady=10)
    button_submit.grid(row=1, column=0, columnspan=2, pady=10)
    label_greeting.grid(row=2, column=0, columnspan=2, pady=10)

    root.mainloop()
```

3.3. Геометр-менеджери (розміщувачі)

Віджети можна розташовувати трьома основними способами:

1. pack() – упаковує віджети один за одним (вертикально або горизонтально). Просто, але мало гнучкості.

```
    label.pack(side="top", padx=5, pady=5)
    button.pack(side="bottom")
```

2. grid() – розміщує віджети в таблиці (сітці) з рядків і стовпців. Найбільш поширений та гнучкий спосіб.

```
    label.grid(row=0, column=0, sticky="w")      # sticky - вирівнювання (w - west/ліворуч)
    entry.grid(row=0, column=1)
```

3. place() – дозволяє точно задати позицію та розмір у пікселях або відносно батьківського контейнера. Використовується рідко для специфічних потреб.

3.4. Обробка подій

Більшість дій у GUI є подіями (event): натискання мишею, натискання клавіш, закриття вікна. Для обробки подій використовуються функції зворотного виклику – **callback-функції**.

У попередньому прикладі command=on_button_click прив’язує функцію on_button_click до події «натискання кнопки».

3.5. Основи CustomTkinter

CustomTkinter використовує схожий принцип, але віджети виглядають сучасніше та мають додаткові параметри (наприклад, appearance_mode – тема).

```
    import customtkinter as ctk

    # Налаштування теми
    ctk.set_appearance_mode("dark")  # "dark", "light", "system"
    ctk.set_default_color_theme("blue")  # "blue", "green", "dark-blue"
```

```
app = ctk.CTk()
app.title("Сучасний інтерфейс")

label = ctk.CTkLabel(app, text="Це CustomTkinter!")
label.pack(padx=20, pady=20)

entry = ctk.CTkEntry(app, placeholder_text="Введіть текст...")
entry.pack(padx=20, pady=10)

button = ctk.CTkButton(app, text="Кнопка", corner_radius=10)
button.pack(padx=20, pady=10)

app.mainloop()
```

Порівняння основних віджетів Tkinter та CustomTkinter:

Призначення	Tkinter	CustomTkinter
Головне вікно	tk.Tk()	ctk.CTk()
Мітка	tk.Label()	ctk.CTkLabel()
Кнопка	tk.Button()	ctk.CTkButton()
Поле введення	tk.Entry()	ctk.CTkEntry()
Перемикач	tk.Checkbutton()	ctk.CTkCheckBox()
Радіокнопка	tk.Radiobutton()	ctk.CTkRadioButton()

CustomTkinter також має додаткові корисні віджети, такі як CTkSlider (слайдер), CTkProgressBar (індикатор прогресу) та CTkOptionMenu (список вибору).

Основна відмінність при роботі – це необхідність встановлення бібліотеки CustomTkinter за допомогою pip install customtkinter, тоді як Tkinter вже є в стандартній бібліотеці Python. Для створення простого та функціонального інтерфейсу з мінімальними зусиллями CustomTkinter є відмінним вибором.

4. Методичні рекомендації

Задача 1. Створення вікна з базовими віджетами (Tkinter)

Створіть головне вікно розміром 500x400 пікселів з заголовком "Калькулятор ваги на Місяці". Додайте:

- Мітку "Введіть вагу на Землі (кг):"
- Поле для введення числа
- Кнопку "Розрахувати"
- Мітку для виведення результату

При натисканні на кнопку програма повинна взяти значення з поля введення, помножити його на 0.165 (прискорення вільного падіння на Місяці) та вивести результат у форматі "Ваша вага на Місяці: X кг" у мітці результату.

```
    import tkinter as tk
    from tkinter import messagebox

    def calculate_weight():
        """Функція для розрахунку ваги на Місяці"""
        try:
            # Отримуємо значення з поля введення та конвертуємо в
число
            earth_weight = float(entry_weight.get())

            # Розраховуємо вагу на Місяці
            moon_weight = earth_weight * 0.165

            # Форматуємо результат та відображаємо його
            result_text = f"Ваша вага на Місяці: {moon_weight:.2f}
кг"
            label_result.config(text=result_text)

        except ValueError:
            # Обробка помилки, якщо введено не число
            messagebox.showerror("Помилка", "Будь ласка, введіть
коректне число!")
    # Створення головного вікна
    root = tk.Tk()
    root.title("Калькулятор ваги на Місяці")
    root.geometry("500x400")
    root.resizable(False, False)  # Заборона зміни розміру вікна

    # Створення та розміщення віджетів
    label_title = tk.Label(root, text="Калькулятор ваги на
Місяці",
```

```
                          font=("Arial", 16, "bold"))
    label_title.pack(pady=20)

    label_instruction = tk.Label(root, text="Введіть вашу вагу на
Землі (кг):",
                                 font=("Arial", 12))
    label_instruction.pack(pady=10)

    # Поле для введення
    entry_weight = tk.Entry(root, font=("Arial", 12), width=20)
    entry_weight.pack(pady=10)

    # Кнопка розрахунку
    button_calculate = tk.Button(root, text="Розрахувати",
                                 font=("Arial", 12, "bold"),
                                 command=calculate_weight,
                                 bg="#4CAF50",    fg="white",       #
Зелений колір
                                 padx=20, pady=10)
    button_calculate.pack(pady=20)

    # Мітка для результату
    label_result   =   tk.Label(root,   text="",   font=("Arial",   14,
"bold"),
                          fg="#2196F3")  # Синій колір
    label_result.pack(pady=20)

    # Запуск головного циклу програми
    root.mainloop()
```

Приклад вхідних та вихідних даних:

```
  Вхід: 70 → Вихід: "Ваша вага на Місяці: 11.55 кг"
  Вхід: 0 → Вихід: "Ваша вага на Місяці: 0.00 кг"
  Вхід:  abc  →  Вихід:  Вікно  помилки  "Будь  ласка,  введіть
коректне число!"
```

Коментарі:

- try-except блок використовується для обробки помилок введення
- messagebox - стандартний модуль Tkinter для показу діалогових вікон
- resizable(False, False) забороняє зміну розміру вікна
- Використання параметрів padx, pady для створення відступів між віджетами

Задача 2. Використання менеджера розміщення Grid (Tkinter)

Створіть форму реєстрації користувача з такими полями:

- Ім’я користувача
- Електронна пошта
- Пароль (з приховуванням символів)
- Підтвердження пароля
- Кнопка "Зареєструватися"

Розташуйте всі елементи за допомогою менеджера grid(). При натисканні на кнопку перевіряйте, чи збігаються паролі.

```
import tkinter as tk
from tkinter import messagebox

def register_user():
    """Функція реєстрації користувача"""
    username = entry_username.get()
    email = entry_email.get()
    password = entry_password.get()
    confirm_password = entry_confirm.get()

    # Перевірка заповнення всіх полів
    if not all([username, email, password, confirm_password]):
        messagebox.showwarning("Увага", "Будь ласка, заповніть всі поля!")
        return

    # Перевірка збігу паролів
    if password != confirm_password:
        messagebox.showerror("Помилка", "Паролі не збігаються!")
        return

    # Успішна реєстрація
    messagebox.showinfo("Успіх",
                        f"Користувач {username} успішно зареєстрований!\nEmail: {email}")

    # Очищення полів
    entry_username.delete(0, tk.END)
    entry_email.delete(0, tk.END)
    entry_password.delete(0, tk.END)
    entry_confirm.delete(0, tk.END)
```

```
    # Створення головного вікна
    root = tk.Tk()
    root.title("Форма реєстрації")
    root.geometry("450x350")
    root.configure(padx=20, pady=20)  # Відступи від країв вікна

    # Створення та розміщення віджетів через grid()
    tk.Label(root, text="Форма реєстрації", font=("Arial", 16,
"bold")).grid(
        row=0, column=0, columnspan=2, pady=(0, 20), sticky="w")

    # Рядок 1: Ім’я користувача
    tk.Label(root, text="Ім’я користувача:").grid(row=1, column=0,
sticky="w", pady=5)
    entry_username = tk.Entry(root, width=30)
    entry_username.grid(row=1, column=1, pady=5, padx=(10, 0))

    # Рядок 2: Email
    tk.Label(root, text="Електронна пошта:").grid(row=2, column=0,
sticky="w", pady=5)
    entry_email = tk.Entry(root, width=30)
    entry_email.grid(row=2, column=1, pady=5, padx=(10, 0))
    # Рядок 3: Пароль
    tk.Label(root,       text="Пароль:").grid(row=3,       column=0,
sticky="w", pady=5)
    entry_password = tk.Entry(root, width=30, show="*")
    entry_password.grid(row=3, column=1, pady=5, padx=(10, 0))

    # Рядок 4: Підтвердження пароля
    tk.Label(root,      text="Підтвердіть      пароль:").grid(row=4,
column=0, sticky="w", pady=5)
    entry_confirm = tk.Entry(root, width=30, show="*")
    entry_confirm.grid(row=4, column=1, pady=5, padx=(10, 0))

    # Рядок 5: Кнопка реєстрації
    btn_register = tk.Button(root, text="Зареєструватися",
                             command=register_user,
                             bg="#2196F3", fg="white",
                             padx=20, pady=5)
    btn_register.grid(row=5, column=0, columnspan=2, pady=20)

    # Налаштування розтягування колонок
    root.columnconfigure(1, weight=1)
```

```
# Запуск програми
root.mainloop()
```

Приклади роботи програми:

```
Всі поля заповнені, паролі збігаються → Повідомлення про успіх
Паролі не збігаються → Повідомлення про помилку
Не всі поля заповнені → Попередження
```

Коментарі:

- show="*" приховує символи пароля
- sticky="w" вирівнює текст по лівому краю
- columnspan=2 об’єднує дві колонки для одного віджета
- root.configure(padx=20, pady=20) додає відступи від країв вікна

Задача 3. Створення тематичного перемикача (CustomTkinter)

Створіть програму з CustomTkinter, яка дозволяє перемикатися між темною та світлою темами. Додайте:

- кнопку для перемикання теми
- декілька віджетів для демонстрації теми
- відображення поточної теми

```
import customtkinter as ctk

def switch_theme():
    """Функція для перемикання теми"""
    current_mode = ctk.get_appearance_mode()

    if current_mode == "Light":
        new_mode = "Dark"
    else:
        new_mode = "Light"

    # Зміна теми
    ctk.set_appearance_mode(new_mode)

    # Оновлення тексту кнопки
    theme_button.configure(text=f"Поточна тема: {new_mode}")

    # Оновлення інформаційної мітки
```

```
        info_label.configure(text=f"Активна тема: {new_mode}")

    # Налаштування за замовчуванням
    ctk.set_appearance_mode("Light")
    ctk.set_default_color_theme("blue")  # "blue", "green", "dark-blue"

    # Створення головного вікна
    app = ctk.CTk()
    app.title("Перемикач тем")
    app.geometry("400x500")

    # Заголовок
    title_label = ctk.CTkLabel(app, text="Демонстрація
CustomTkinter",
                               font=("Arial", 24, "bold"))
    title_label.pack(pady=30)

    # Інформаційна мітка
    info_label = ctk.CTkLabel(app, text="Активна тема: Light",
                              font=("Arial", 14))
    info_label.pack(pady=10)

    # Кнопка перемикання теми
    theme_button = ctk.CTkButton(app, text="Поточна тема: Light",
                                 command=switch_theme,
                                 font=("Arial", 14, "bold"),
                                 height=40)
    theme_button.pack(pady=20)

    # Фрейм для демонстраційних віджетів
    demo_frame = ctk.CTkFrame(app)
    demo_frame.pack(pady=30, padx=40, fill="both", expand=True)

    # Різні віджети для демонстрації
    ctk.CTkLabel(demo_frame, text="Текстова мітка").pack(pady=10)
    ctk.CTkEntry(demo_frame, placeholder_text="Поле
введення").pack(pady=10)

    # Checkbox
    checkbox_var = ctk.BooleanVar(value=True)
    checkbox = ctk.CTkCheckBox(demo_frame, text="Включити опцію",
                               variable=checkbox_var)
    checkbox.pack(pady=10)

    # Радіокнопки
```

```
    radio_var = ctk.StringVar(value="Option 1")
    ctk.CTkRadioButton(demo_frame, text="Опція 1",
                       variable=radio_var,            value="Option
1").pack(pady=5)
    ctk.CTkRadioButton(demo_frame, text="Опція 2",
                      variable=radio_var, value="Option 2").pack(pady=5)

    # Слайдер
    slider = ctk.CTkSlider(demo_frame, from_=0, to=100)
    slider.pack(pady=20)
    slider.set(50)

    # Запуск програми
    app.mainloop()
```

Очікувана поведінка:

```
  При запуску активна світла тема
  При натисканні кнопки тема змінюється на протилежну
  Всі віджети автоматично оновлюють свій вигляд відповідно до
теми
```

Коментарі:

- ctk.set_appearance_mode() встановлює глобальну тему
- ctk.get_appearance_mode() повертає поточну тему
- CustomTkinter автоматично оновлює всі віджети при зміні теми
- Використання BooleanVar та StringVar для зв'язування значень віджетів

Задача 4. Простий калькулятор з обробкою подій клавіатури

Створіть простий калькулятор (Tkinter), який підтримує введення з клавіатури. Реалізуйте операції: +, -, *, /.

```
import tkinter as tk
from tkinter import messagebox

def button_click(value):
    """Обробка натискання кнопок калькулятора"""
    current = entry_display.get()
    entry_display.delete(0, tk.END)
    entry_display.insert(0, current + str(value))

def clear_display():
```

```
        """Очищення дисплея"""
        entry_display.delete(0, tk.END)

    def calculate():
        """Виконання обчислення"""
        try:
            expression = entry_display.get()
            # Безпечне обчислення виразу
            result = eval(expression)
            entry_display.delete(0, tk.END)
            entry_display.insert(0, str(result))
        except ZeroDivisionError:
            messagebox.showerror("Помилка", "Ділення на нуль!")
            clear_display()
        except Exception as e:
            messagebox.showerror("Помилка",          f"Неправильний
вираз!\n{str(e)}")
            clear_display()

    def key_press(event):
        """Обробка натискання клавіш"""
        key = event.char

        # Цифри та оператори
        if key in '0123456789+-*/.()':
            button_click(key)
        # Enter для обчислення
        elif key == '\r':
            calculate()
        # Escape для очищення
        elif key == '\x1b':
            clear_display()
        # Backspace для видалення останнього символу
        elif key == '\x08':
            current = entry_display.get()
            entry_display.delete(0, tk.END)
            entry_display.insert(0, current[:-1])

    # Створення головного вікна
    root = tk.Tk()
    root.title("Калькулятор")
    root.geometry("300x400")
    # Дисплей калькулятора
```

```
    entry_display = tk.Entry(root, font=("Arial", 20),
justify="right", bd=10)
    entry_display.grid(row=0, column=0, columnspan=4,
sticky="nsew", padx=5, pady=5)

    # Прив'язка обробки клавіш
    root.bind('<Key>', key_press)

    # Кнопки калькулятора
    buttons = [
        ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
        ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
        ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
        ('0', 4, 0), ('.', 4, 1), ('=', 4, 2), ('+', 4, 3)
    ]
    # Створення кнопок
    button_objects = []
    for (text, row, col) in buttons:
        if text == '=':
            btn = tk.Button(root, text=text, font=("Arial", 16),
                            command=calculate, bg="#4CAF50", fg="white")
        elif text in 'C':
            btn = tk.Button(root, text=text, font=("Arial", 16),
                            command=clear_display, bg="#f44336",
fg="white")
        else:
            btn = tk.Button(root, text=text, font=("Arial", 16),
                            command=lambda t=text: button_click(t))

        btn.grid(row=row, column=col, sticky="nsew", padx=2, pady=2)
        button_objects.append(btn)

    # Кнопка очищення
    btn_clear = tk.Button(root, text="C", font=("Arial", 16),
                          command=clear_display, bg="#f44336",
fg="white")
    btn_clear.grid(row=5, column=0, columnspan=2, sticky="nsew",
padx=2, pady=2)
    # Кнопка виходу
    btn_exit = tk.Button(root, text="Вихід", font=("Arial", 16),
                         command=root.quit, bg="#607D8B", fg="white")
    btn_exit.grid(row=5, column=2, columnspan=2, sticky="nsew",
padx=2, pady=2)

    # Налаштування розтягування
```

```
for i in range(5):
    root.rowconfigure(i, weight=1)
for i in range(4):
    root.columnconfigure(i, weight=1)

# Фокус на полі введення
entry_display.focus_set()

# Запуск програми
root.mainloop()
```

Приклади обчислень:

Вхід: 12+34= → Вихід: 46
Вхід: 10/2= → Вихід: 5
Вхід: 10/0= → Вихід: Повідомлення про помилку "Ділення на нуль!"

Коментарі:

- eval() використовується для обчислення виразів (у реальних проектах вимагає обережності)
- bind('<Key>', key_press) прив'язує обробку натискань клавіш
- focus_set() встановлює фокус на поле введення
- Лямбда-функції використовуються для передачі параметрів у функції обробки

Задача 5. Прогрес-бар та багатопотоковість (Tkinter)

Створіть програму з прогрес-баром, який імітує довгу операцію. Використайте threading для уникнення "зависання" інтерфейсу.

```
import tkinter as tk
from tkinter import ttk
import threading
import time
import random

class ProgressApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Прогрес-бар з потоками")
        self.root.geometry("400x250")

        self.setup_ui()
        self.is_running = False
```

```
        def setup_ui(self):
            """Налаштування інтерфейсу"""
            # Заголовок
            self.label_title = tk.Label(self.root, text="Імітація
довгої операції",
                                        font=("Arial", 16, "bold"))
            self.label_title.pack(pady=20)

            # Прогрес-бар
            self.progress = ttk.Progressbar(self.root, length=300,
                                            mode='determinate')
            self.progress.pack(pady=20)

            # Мітка статусу
            self.label_status = tk.Label(self.root, text="Готово
до роботи",
                                         font=("Arial", 10))
            self.label_status.pack(pady=10)

            # Фрейм для кнопок
            button_frame = tk.Frame(self.root)
            button_frame.pack(pady=20)

            # Кнопка запуску
            self.btn_start = tk.Button(button_frame, text="Старт",
                                       command=self.start_progress,
                                       width=10, bg="#4CAF50",
fg="white")
            self.btn_start.pack(side="left", padx=10)

            # Кнопка зупинки
            self.btn_stop = tk.Button(button_frame, text="Стоп",
                                      command=self.stop_progress,
                                      width=10, bg="#f44336",
fg="white",
                                      state="disabled")
            self.btn_stop.pack(side="left", padx=10)

        def start_progress(self):
            """Запуск імітації довгої операції"""
            if not self.is_running:
                self.is_running = True
                self.btn_start.config(state="disabled")
```

```
                self.btn_stop.config(state="normal")
                self.progress['value'] = 0

                # Запуск операції в окремому потоці
                self.thread =
threading.Thread(target=self.long_operation)
                self.thread.daemon = True    # Поток завершиться
разом з програмою
                self.thread.start()

        def stop_progress(self):
            """Зупинка операції"""
            self.is_running = False
            self.label_status.config(text="Операція зупинена")
            self.btn_start.config(state="normal")
            self.btn_stop.config(state="disabled")

        def long_operation(self):
            """Імітація довгої операції"""
            total_steps = 100
            for i in range(total_steps + 1):
                if not self.is_running:
                    break

                # Оновлення прогрес-бару з головного потоку
                self.root.after(0, self.update_progress, i)

                # Імітація затримки
                time.sleep(0.05 + random.random() * 0.1)   # 50-150
мс

            if self.is_running:
                self.root.after(0, self.operation_complete)

        def update_progress(self, value):
            """Оновлення інтерфейсу (викликається з головного
потоку)"""
            self.progress['value'] = value
            percent = (value / 100) * 100
            self.label_status.config(
                text=f"Виконано: {value}% ({value}/100 кроків)")

            # Зміна кольору при досягненні 100%
            if value == 100:
```

```
                self.label_status.config(fg="green",
font=("Arial", 10, "bold"))

        def operation_complete(self):
            """Завершення операції"""
            self.is_running = False
            self.label_status.config(text="Операцію                завершено
успішно!",
                                     fg="green")
            self.btn_start.config(state="normal")
            self.btn_stop.config(state="disabled")

    # Запуск програми
    if __name__ == "__main__":
        root = tk.Tk()
        app = ProgressApp(root)
        root.mainloop()
```

Очікувана поведінка:

```
При натисканні "Старт" прогрес-бар починає заповнюватися
Інтерфейс залишається відповідальним під час виконання
Можливість зупинити операцію кнопкою "Стоп"
```

Коментарі:

- threading.Thread використовується для запуску довгої операції в окремому потоці
- self.root.after(0, ...) гарантує, що оновлення GUI відбувається в головному потоці
- daemon=True робить потік фоновим (завершується з програмою)
- ttk.Progressbar - стилізований прогрес-бар з Tkinter

Типові помилки і шляхи їх усунення

1. **Помилка. "TclError: couldn’t connect to display"**
   Причина. Проблеми з графічним середовищем (часто на серверах без GUI)
   Рішення. Використовувати віртуальний дисплей (xvfb) або переконатися, що запускаєте код на системі з графічним інтерфейсом.
2. **Помилка. "ImportError: No module named ‘customtkinter’"**
   Причина. Бібліотека не встановлена
   Рішення. Виконати pip install customtkinter у терміналі
3. **Проблема. Інтерфейс "зависає" під час довгих обчислень**

Причина. Довга операція виконується в головному потоці GUI

Рішення. Використовувати модуль threading для запуску довгих операцій у фоновому потоці

4. **Помилка. Віджети не відображаються або з'являються у неправильних місцях**

Причина. Неправильне використання геометр-менеджерів

Рішення. Використовувати лише один тип менеджера (pack, grid або place) для кожного контейнера

5. **Проблема. Програма закривається відразу після запуску**

Причина. Відсутність виклику mainloop() або використання root.mainloop() в неправильному місці

Рішення. Переконатися, що mainloop() викликається після налаштування всіх віджетів

6. **Помилка з eval() в калькуляторі**

Причина. Небезпечний ввід користувача

Рішення. Обмежити допустимі символи або використовувати безпечні методи обчислення

Корисні поради

1. **Почніть з Tkinter, потім переходьте до CustomTkinter.** Tkinter має простішу документацію та більше прикладів. CustomTkinter вимагає додаткового встановлення, але дає кращий вигляд.

2. **Використовуйте об'єктно-орієнтований підхід для складних програм.** Створюйте класи для ваших вікон. Це полегшує управління станом програми та повторне використання коду.

3. **Розділяйте логіку та інтерфейс.** Не змішувайте обчислювальну логіку з кодом GUI. Створюйте окремі функції для обробки даних.

4. **Тестуйте на різних розмірах екрану.** Використовуйте minsize() та maxsize() для обмеження розміру вікна. Перевіряйте, як ваш інтерфейс виглядає на екранах різної роздільної здатності.

5. **Документуйте свої callback-функції.** Завжди пишіть docstring для функцій, які обробляють події. Це допоможе зрозуміти код через деякий час.

6. **Використовуйте менеджер grid() для складних макетів.** grid() надає більше контролю над розміщенням. Використовуйте columnspan та rowspan для об'єднання клітинок.

7. **Не забувайте про обробку помилок.** Завжди перевіряйте введення користувача. Використовуйте try-except блоки для критичних операцій.

8. **Експериментуйте з різними темами та кольорами.** CustomTkinter пропонує кілька вбудованих тем. Використовуйте колірні схеми, що відповідають призначенню програми.

5. Завдання для виконання

Частина 1. Легкі завдання

Завдання 1.1. Вітальна картка

Створіть програму, яка відображає вітальну картку. Вікно розміром 400x300 пікселів має заголовок "Моя вітальна картка". У вікні розмістіть:

- Заголовок "Вітаю!" шрифтом Arial 18 пунктів
- Ваше ім’я під заголовком
- Назву вашої групи
- Кнопку "Закрити" яка завершує роботу програми

Приклад виходу: Вікно з вказаними елементами. При натисканні кнопки "Закрити" програма завершується.

Завдання 1.2. Конвертер температури

Створіть програму для конвертації градусів Цельсія в Фаренгейти. Вікно має містити:

- Поле для введення температури в Цельсіях
- Кнопку "Конвертувати"
- Мітку для відображення результату в Фаренгейтах

Формула конвертації: $F = C \times 9/5 + 32$

```
Вхід: 0 → Вихід: 32.0
Вхід: 100 → Вихід: 212.0
Вхід: -40 → Вихід: -40.0
```

Завдання 1.3. Простий лічильник

Створіть програму з лічильником, який має:

- Мітку з поточним значенням лічильника (починається з 0)
- Кнопку "+" для збільшення значення на 1
- Кнопку "-" для зменшення значення на 1
- Кнопку "Скинути" для повернення до 0

Приклад роботи: Початкове значення: 0. При 3 натисканнях "+": 3. Потім 1 натискання "-": 2. Натискання "Скинути": 0.

Завдання 1.4. Перемикач кольорів

Створіть програму з трьома кнопками, кожна з яких змінює колір фону головного вікна на певний колір:

- "Червоний" - змінює фон на червоний

- "Зелений" - змінює фон на зелений
- "Синій" - змінює фон на синій

Додайте четверту кнопку "Початковий", яка повертає білий фон.

Приклад роботи: При натисканні "Зелений" фон стає зеленим, потім "Синій" - синім, "Початковий" - білим.

Завдання 1.5. Обчислення площі прямокутника

Створіть програму для обчислення площі прямокутника. Вікно має містити:

- Два поля для введення: довжина та ширина
- Кнопку "Обчислити"
- Мітку для відображення результату

```
Вхід: 5, 4 → Вихід: "Площа: 20"
Вхід: 10, 3.5 → Вихід: "Площа: 35.0"
Вхід: 0, 10 → Вихід: "Площа: 0"
```

Завдання 1.6. Перевірка пароля

Створіть форму для перевірки пароля. Програма має:

- Поле для введення пароля (з приховуванням символів)
- Кнопку "Перевірити"
- Мітку, яка відображає результат перевірки

Пароль вважається правильним, якщо:

- Має щонайменше 8 символів
- Містить хоча б одну цифру
- Містить хоча б одну велику літеру

```
Вхід: "Password123" → Вихід: "Пароль надійний"
Вхід: "qwerty" → Вихід: "Занадто короткий (мінімум 8)"
Вхід: "пароль" → Вихід: "Потрібна цифра та велика літера"
```

Завдання 1.7. Генератор випадкових чисел

Створіть генератор випадкових чисел з діапазоном. Вікно має:

- Два поля для введення мінімального та максимального значення
- Кнопку "Згенерувати"
- Мітку для відображення згенерованого числа
- Кнопку "Скопіювати", яка копіює число в буфер обміну

```
Діапазон: 1-10 → Вихід: випадкове число між 1 та 10
Діапазон: -5-5 → Вихід: випадкове число між -5 та 5
Діапазон: 100-100 → Вихід: 100
```

Завдання 1.8. Таймер зворотного відліку

Створіть програму для встановлення таймера. Вікно має:

- Поле для введення кількості секунд
- Кнопку "Старт"
- Мітку, яка відображає залишковий час
- Мітку "Час вийшов!", яка з'являється після завершення

При натисканні "Старт" мітка починає відлік у секундах.

```
Приклад роботи: Вхід: 5 → Вихід: відлік 5, 4, 3, 2, 1, 0 → "Час вийшов!"
```

Завдання 1.9. Форма зворотного зв'язку

Створіть просту форму зворотного зв'язку з такими полями:

- Ім'я (текстове поле)
- Email (текстове поле)
- Повідомлення (багаторядкове текстове поле)
- Кнопка "Надіслати"

При натисканні кнопки програма повинна перевіряти, чи заповнені всі поля, та виводити повідомлення про успішне відправлення.

```
Приклад роботи: Всі поля заповнені → "Дякуємо за відгук!"; Не всі поля заповнені → "Заповніть усі поля!"
```

Завдання 1.10. Калькулятор чайових

Створіть калькулятор для розрахунку чайових. Вікно має:

- Поле для введення суми рахунку
- Радіокнопки для вибору відсотка (10%, 15%, 20%)
- Кнопку "Розрахувати"
- Мітку для відображення суми чайових та загальної суми

```
Рахунок: 1000, 15% → Вихід: "Чайові: 150, Всього: 1150"
Рахунок: 500, 20% → Вихід: "Чайові: 100, Всього: 600"
Рахунок: 200, 10% → Вихід: "Чайові: 20, Всього: 220"
```

Частина 2. Середні завдання

Завдання 2.1. Менеджер завдань (To-Do List)

Створіть простий менеджер завдань. Вікно має:

- Поле для введення нового завдання
- Кнопку "Додати" для додавання завдання до списку
- Список (Listbox) з усіма завданнями
- Кнопку "Видалити" для видалення вибраного завдання

- Кнопку "Очистити всі" для видалення всіх завдань
- Лічильник кількості завдань

Приклад роботи: Додати "Купити молоко" → список: "Купити молоко", лічильник: 1. Вибрати та натиснути "Видалити" → список порожній.

Завдання 2.2. Редактор тексту з форматуванням

Створіть простий текстовий редактор з можливістю форматування:

- Багаторядкове текстове поле (Text)
- Кнопки для форматування: "Жирний", "Курсив", "Підкреслений"
- Вибір розміру шрифту (Combobox з варіантами: 10, 12, 14, 16, 18)
- Вибір кольору тексту (3 кнопки: чорний, синій, червоний)
- Кнопки "Копіювати", "Вставити", "Очистити"

Приклад роботи: Введення тексту, вибір "Жирний" та "16" → текст стає жирним зі шрифтом 16 пунктів.

Завдання 2.3. Конвертер валют з API (симуляція)

Створіть конвертер валют з фіксованими курсами. Вікно має:

- Поле для введення суми
- Два випадаючих списки (Combobox) для вибору валюти "з" та "в"
- Кнопку "Конвертувати"
- Мітку для результату
- Таблицю (Treeview) з історією конвертацій

Фіксовані курси (до UAH): USD = 38.5, EUR = 41.2, PLN = 9.1

```
100 USD → UAH: 3850
500 EUR → USD: 500 × (41.2/38.5) ≈ 535.06
1000 UAH → PLN: 1000/9.1 ≈ 109.89
```

Завдання 2.4. Калькулятор з пам’яттю та історією

Розширте простий калькулятор (+, -, *, /) додавши:

- Кнопки M+, M-, MR, MC для роботи з пам’яттю
- Вікно історії обчислень (Listbox або Text)
- Кнопку "Зберегти історію" для запису в файл
- Кнопку "Завантажити історію" для читання з файлу

Приклад роботи: 5 + 3 = 8 (додається в історію). M+ → збереження 8 в пам’ять. MR → вставка 8 в поле введення.

Завдання 2.5. Гра "Вгадай число" з рейтингом

Створіть гру, де комп’ютер загадує число від 1 до 100, а гравець намагається вгадати. Вікно має:

- Поле для введення числа
- Кнопку "Спробувати"
- Мітку з підказками ("більше", "менше", "вірно!")
- Лічильник спроб
- Таблицю рейтингу гравців (ім’я, кількість спроб)
- Кнопку "Нова гра"

Приклад роботи: Загадано 42. Гравець: 50 → "Менше!", 25 → "Більше!", 42 → "Вірно! Спроб: 3". Збереження в рейтинг.

Частина 3. Складні завдання

Завдання 3.1. Текстовий редактор з меню та панеллю інструментів

Створіть повноцінний текстовий редактор з такими елементами:

1. **Головне меню:**
   - Файл (Новий, Відкрити, Зберегти, Зберегти як, Вийти)
   - Редагування (Вирізати, Копіювати, Вставити, Видалити, Вибрати все)
   - Формат (Шрифт, Розмір, Колір, Вирівнювання)
   - Довідка (Про програму)
2. **Панель інструментів** з іконками/кнопками для основних дій
3. **Статусний рядок** з інформацією:
   - кількість символів
   - кількість слів
   - поточна позиція курсора
4. **Основна область** - багаторядкове текстове поле з прокруткою
5. **Функціонал:**
   - робота з файлами (відкриття, збереження)
   - пошук та заміна тексту
   - друк документу (симуляція)
   - автозбереження кожні 5 хвилин

Приклад роботи: Створення документа, форматування, збереження в .txt файл, статистика в статусному рядку.

Завдання 3.2. Менеджер персональних фінансів

Створіть програму для обліку особистих фінансів з такими функціями:

1. **Головне вікно з вкладками (Notebook):**

- "Операції" - додавання нових транзакцій
- "Категорії" - управління категоріями витрат/доходів
- "Звіти" - графіки та статистика
- "Бюджет" - планування та контроль бюджету

2. **Форма додавання операції:**
   - тип (витрата/дохід)
   - категорія (випадаючий список)
   - сума
   - дата (календар)
   - опис
   - теги
3. **Аналітика та звіти:**
   - кругова діаграма витрат по категоріям
   - графік доходів/витрат по місяцях
   - таблиця з найбільшими витратами
   - фільтрація за періодом та категоріями
4. **Робота з даними:**
   - збереження в JSON/CSV файл
   - експорт звітів у PDF (симуляція)
   - резервне копіювання даних
5. **Додаткові функції:**
   - нагадування про регулярні платежі
   - встановлення бюджету на категорії
   - аналіз перевищення бюджету

Приклад роботи: Додавання операції "Продукти: 500 грн", перегляд статистики за місяць, аналіз витрат по категоріям.

Завдання 3.3. Додаток для навчання з використанням CustomTkinter

Створіть сучасний додаток для вивчення іноземних слів з такими елементами:

1. **Сучасний інтерфейс на CustomTkinter:**
   - темна/світла тема з автоматичним перемиканням
   - анімовані переходи між екранами
   - сучасні кнопки та віджети з закругленнями
2. **Модулі додатку:**
   - Словник - додавання/редагування/видалення слів

- Тренування - різні режими (вибір перекладу, написання, аудіювання)
- Прогрес - графіки успішності, статистика
- Налаштування - тема, мова, параметри тренувань

3. **Картки слів:**
   - відображення слова та перекладу
   - кнопки "знаю"/"потрібно повторити"
   - прогрес-бар для кожного слова
   - прикріплення зображень до слів
4. **Система повторень за алгоритмом Leitner:**
   - розподіл слів по рівнях знань
   - автоматичне планування повторень
   - нагадування про тренування
5. **Мультимедійні функції:**
   - текст-в-мова (симуляція) для озвучування слів
   - генератор вправ
   - імпорт/експорт словників
6. **Гейміфікація:**
   - система балів та досягнень
   - щоденні цілі
   - рейтингова таблиця (якщо багато користувачів)

Приклад роботи: Додавання слова "apple - яблуко", тренування з вибором перекладу, перегляд статистики успішності, зміна теми на темну.

Питання для самоперевірки

1. Яка основна функція методу mainloop() в Tkinter і чому вона критично важлива?
2. Назвіть три основні геометр-менеджери Tkinter та в чому їхні ключові відмінності?
3. Що таке callback-функція (функція зворотного виклику) і як вона пов’язана з обробкою подій у GUI?
4. Як правильно обробляти помилки введення даних користувачем (наприклад, введення тексту замість числа)? Наведіть приклад коду.
5. Поясніть різницю між tk.Entry та tk.Text. У яких випадках краще використовувати кожен з них?
6. Як зробити, щоб символи пароля не відображались відкрито у полі введення?

7. Що таке StringVar, IntVar, BooleanVar і навіщо вони використовуються?

8. Чому важливо уникати виконання довготривалих операцій у головному потоці GUI та як це можна зробити?

9. Яку проблему може викликати одночасне використання pack() та grid() для віджетів у одному контейнері (наприклад, у головному вікні)?

10. Назвіть щонайменше три переваги використання CustomTkinter перед стандартним Tkinter.

11. Як створити випадаючий список (combobox) у стандартному Tkinter?

12. Яким чином можна додати зображення до програми на Tkinter?

13. Що робить метод after() і в яких ситуаціях він корисний?

14. Як створити розділювальні меню (як "Файл", "Редагувати") у головному вікні програми?

15. Яка роль параметрів padx, pady, sticky у менеджері grid()?

16. Що таке модальне вікно і як його створити?

17. Як можна змінити стандартну тему (зовнішній вигляд) віджетів у Tkinter?

18. Чому важлива правильна структура коду (наприклад, використання класів) при створенні складних GUI-додатків?

19. Як організувати оновлення елементів GUI (наприклад, прогрес-бару) з фонового потоку?

20. Назвіть основні етапи створення будь-якої GUI-програми на Tkinter/CustomTkinter.

ЛАБОРАТОРНА РОБОТА №19
Фінальний проект: GUI-додаток з використанням Tkinter/CustomTkinter

1. Мета

Систематизувати та застосувати знання, отримані протягом курсу, для розробки повноцінного, функціонального графічного додатку з інтуїтивно зрозумілим інтерфейсом. Проект має продемонструвати вміння студента інтегрувати основні концепції програмування (змінні, структури даних, функції, ООП, робота з файлами, обробка винятків) у практичному продукті.

Тип роботи

Індивідуальний або командний (2 особи) підсумковий проект. Проект складається з послідовних етапів, кожен з яких має чіткі критерії виконання та оцінювання.

Прикладні теми проектів (на вибір)

Студент може обрати одну з наведених тем або запропонувати свою (з обов'язковим затвердженням викладачем). Приклад власної теми має включати схожий рівень складності.

1. **Персональний менеджер завдань (To-Do List):** додавання, видалення, редагування завдань, відмітка про виконання, сортування за пріоритетом/датою, збереження списку у файл.
2. **Простий текстовий редактор:** відкриття, збереження, створення нових .txt файлів. Опціонально: пошук по тексту, зміна шрифту.
3. **Калькулятор розрахунку витрат (Budget Tracker):** введення статей доходів/витрат, категорії, суми, дати. Візуалізація за допомогою простих кругових або стовпчастих діаграм (бібліотека matplotlib або прості засоби Tkinter Canvas). Збереження даних.
4. **Додаток для квізу (вікторини):** завантаження питань та варіантів відповідей з файлу (JSON/CSV), таймер, підрахунок балів, відображення результату.
5. **Гра "Вгадай число" або "Хрестики-нулики" з історією:** гра проти комп'ютера або другого гравця, ведення таблиці рекордів, яка зберігається між сесіями.

2. Етапи виконання та критерії оцінювання

Загальна максимальна оцінка: 100 балів.

Етап 1: Планування та проектування (10 балів)

Завдання: створити технічне завдання та макет вашого додатку.

1.1. (3 бали) Вибір та обґрунтування теми проекту. Короткий опис (5-7 речень): що робить додаток, яку проблему вирішує?

1.2. (4 бали) Створення ескізу інтерфейсу (wireframe). Можна намалювати від руки, в Paint, або використати спеціальні інструменти (diagrams.net). Ескіз має показати розташування **всіх** основних віджетів (кнопок, полів вводу, міток, списків тощо) на головному та додаткових вікнах.

1.3. (3 бали) Опис функціональних вимог у вигляді нумерованого списку. Наприклад: "1. Програма повинна дозволяти користувачу ввести нову задачу у текстове поле. 2. При натисканні кнопки ‘Додати’ задача має з’явитися у списку. 3. Користувач повинен мати можливість видалити обрану задачу."

Етап 2: Розробка базового інтерфейсу (GUI) (15 балів)

Завдання: реалізувати "макет" програми без активної логіки, згідно з ескізом.

2.1. (5 балів) Створення головного вікна програми з заголовком та базовими налаштуваннями (розмір, можливість зміни розміру).

2.2. (5 балів) Коректне розміщення всіх запланованих віджетів за допомогою менеджерів компонування (pack, grid або place). Обов’язково використати мінімум **5 різних типів** віджетів (наприклад: Label, Entry, Button, Listbox/Treeview, Combobox, Checkbutton, Text, Canvas).

2.3. (5 балів) Базова стилізація: вибір кольорової схеми (фон, кнопки), налаштування шрифтів для заголовків та основного тексту. Інтерфейс має бути акуратним та зручним для сприйняття.

Етап 3: Реалізація бізнес-логіки та функціоналу (40 балів)

Завдання: "оживити" інтерфейс, додавши обробку дій користувача та основний функціонал програми.

3.1. (10 балів) Написання функцій-обробників подій (event handlers) для всіх інтерактивних елементів (кнопки, поля вибору тощо).

Код має реагувати на дії користувача (наприклад, виводити повідомлення при натисканні).

3.2. (10 балів) Реалізація валідації введених користувачем даних (перевірка на порожність, правильний тип даних, діапазон значень). Виведення зрозумілих повідомлень про помилку (наприклад, у Label або messagebox).

3.3. (10 балів) Реалізація роботи з файлами для збереження стану програми (наприклад, списку завдань, історії операцій). Дані мають зберігатися при закритті та завантажуватися при запуску. Формат: JSON, CSV або текстовий файл.

3.4. (10 балів) Інтеграція всієї логіки програми. Усі модулі (інтерфейс, обробка даних, робота з файлами) мають працювати як єдине ціле. Додаток виконує свою основну функцію (керує завданнями, веде бюджет і т.д.).

Етап 4: Обробка помилок та тестування (10 балів)

Завдання: забезпечити стабільну роботу програми в нестандартних ситуаціях.

4.1. (5 балів) Використання конструкцій try-except-finally для всіх критичних операцій: робота з файлами (відсутній файл, відмова в доступі), потенційні помилки перетворення типів (ValueError), ділення на нуль тощо.

4.2. (5 балів) Проведення ручного тестування: перевірити всі сценарії використання (коректні та некоректні). Створити короткий звіт або коментар у коді/README про те, що було протестовано.

Етап 5: Документація та якість коду (20 балів)

5.1. Документація (10 балів):

- README.md у корені проекту з назвою, описом, **скріншотами інтерфейсу**, інструкцією зі встановлення (потрібні бібліотеки) та запуску.
- Чіткі інлайнові коментарі для неочевидних ділянок коду.
- Docstrings для всіх функцій та класів (короткий опис, аргументи, що повертає).

5.2. Якість коду (10 балів):

- Логічне розділення коду на функції/класи/модулі (наприклад, окремий файл для логіки та окремий для GUI).
- Відсутність дублювання коду (DRY principle).

- Читабельність: зрозумілі назви змінних і функцій, дотримання стилю PEP 8 (відступи, довжина рядків).
- Використання ООП підходу (наприклад, створення класу для головного додатку або для об'єктів даних) – **заохочується, але не є обов'язковим для всіх тем**.

Етап 6: Презентація та здача (5 балів)

6.1. (5 балів) Підготовка до захисту проекту. Студент має бути готовий:

- продемонструвати основний функціонал програми;
- пояснити ключові архітектурні рішення;
- розповісти про виниклі складнощі та способи їх подолання;
- відповісти на запитання щодо коду.

Обов'язкові технічні вимоги до проекту

- використання стандартної бібліотеки tkinter або сучасної customtkinter;
- наявність мінімум **5 різних типів** віджетів;
- обробка подій користувача (кліки, введення тексту);
- збереження та завантаження даних у файл (стан програми не має втрачатися після перезапуску);
- наявність обробки помилок (try-except);
- код повинен бути чистим, добре структурованим і закоментованим;
- проект має бути розміщений на **GitHub** (або аналогічному сервісі);
- репозиторій **обов'язково** має містити файл README.md.

3. Форма здачі проекту

1. **Посилання на публічний репозиторій GitHub.** Назва репозиторію: Final-Project-GUI-[Прізвище] (наприклад, Final-Project-GUI-Petrenko).
2. **Репозиторій має містити:**

- повний вихідний код проекту;
- файл README.md (за описом вище);
- файли даних (якщо потрібно) або папку з ресурсами;
- файл requirements.txt зі списком залежностей (наприклад, customtkinter), якщо вони використовуються.

3. **Усна презентація та захист проекту** на останньому занятті або за розкладом.

ПІСЛЯМОВА

Завершення виконання 19 лабораторних робіт цього курсу – це значне досягнення, яке знаменує перехід від початкового знайомства з програмуванням до здатності створювати завершені програмні продукти. Пройдений шлях охопив усі ключові аспекти сучасної розробки програмного забезпечення на мові Python, заклавши міцний фундамент для вашої майбутньої кар'єри в IT.

Підсумки опанованих знань та навичок

Розпочавши з найпростіших інструкцій виведення тексту та роботи з базовими типами даних (int, float, str, bool), ви навчилися будувати фундамент будь-якої програми – взаємодію з користувачем через функції input() та print(). Розуміння пріоритетів арифметичних та логічних операторів дозволило вам створювати перші математичні моделі.

Етап вивчення керуючих конструкцій (if-elif-else) та циклів (for, while) став переломним у формуванні вашого алгоритмічного мислення. Ви навчилися не просто писати лінійний код, а керувати логікою програми, реалізовувати ітераційні процеси та працювати зі складними вкладеними структурами.

Особливу увагу було приділено роботі з колекціями. Ви опанували незмінність рядків та потужність зрізів, гнучкість списків та надійність кортежів. Вивчення словників та множин відкрило шлях до ефективного збереження унікальних даних та побудови асоціативних зв'язків. Застосування лаконічних comprehensions (генераторів) дозволило вам писати «пітонічний», читабельний та продуктивний код.

Перехід до функціонального програмування та декомпозиції навчив вас розбивати складні проблеми на ізольовані модулі, що підлягають повторному використанню. Ви занурилися у розширені механізми: анонімні lambda-функції, функції вищого порядку та рекурсію. Робота з модулями та файловою системою навчила вас створювати стійкі програми, що зберігають свій стан між запусками.

Вивчення структур даних та алгоритмів (стеки, черги, методи сортування та пошуку) разом із аналізом їхньої складності (Big O notation) дало вам інструменти для оцінки ефективності ваших рішень. Кульмінацією теоретичної підготовки стало опанування об'єктно-орієнтованого програмування. Концепції класів, наслідування, поліморфізму та інкапсуляції дозволили вам моделювати реальні сутності та будувати масштабовані системи.

Нарешті, ви навчилися візуалізувати результати своєї роботи за допомогою Matplotlib та створювати сучасні графічні інтерфейси (GUI) за допомогою Tkinter та CustomTkinter. Фінальний проект продемонстрував вашу здатність інтегрувати всі ці знання в єдиний функціональний продукт.

Шлях у світ IT тільки починається

Програмування – це не лише набір технічних навичок, це мистецтво розв'язання проблем. Ви навчилися використовувати інструменти штучного інтелекту для аналізу та оптимізації коду, що є критично важливою компетенцією в сучасних реаліях.

Світ інформаційних технологій безмежний. Отримані знання з Python відкривають перед вами двері в такі галузі:

- Data Science та Machine Learning: аналіз великих даних та створення моделей ШІ;
- Web Development: розробка серверної частини складних веб-додатків;
- Automation & DevOps: автоматизація системних процесів та розгортання інфраструктури;
- Cybersecurity: тестування систем на проникнення та розробка захисного ПЗ.

Поради на майбутнє

1. **Ніколи не припиняйте вчитися.** Технології змінюються швидко, і здатність до самоосвіти – ваш головний капітал.
2. **Практикуйтеся щодня.** Навіть невеликі завдання підтримують ваш "код-тонус".
3. **Читайте чужий код.** Досліджуйте репозиторії на **GitHub**, вивчайте відкриті проекти – це найкращий спосіб побачити різні підходи до розв'язання задач.
4. **Створюйте власні проекти.** Не бійтеся помилятися. Кожна помилка в консолі – це крок до розуміння того, як система працює насправді.

Ці методичні рекомендації були вашим путівником, але тепер ви готові до вільного плавання в океані IT. Будьте сміливими в своїх ідеях, наполегливими в їх реалізації та завжди прагніть до досконалості у своєму коді.

Успіхів вам у підкоренні нових вершин програмування!

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритмізація і програмування : метод. реком. до виконання практичних робіт здобувачами вищої освіти ступеня «молодший бакалавр» факультету менеджменту спеціальності 122 «Комп'ютерні науки» денної форми навчання / уклад. Ю. В. Волосюк. Миколаїв : МНАУ, 2021. 52 с. URL: https://dspace.mnau.edu.ua/jspui/handle/123456789/10866
2. Алгоритмізація та програмування. Частина 1. Базові концепції програмування. Лабораторний практикум : навчальний посібник / уклад. В. В. Романов, Т. І. Просянкіна-Жарова, О. Ю. Безносик. Київ : КПІ ім. Ігоря Сікорського, 2022. 150 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/596d62b6-8251-4013-9c48-960e10b6dfb2/content.
3. Бандоріна Л. М., Климкович Т. О., Удачина К. О. Основи алгоритмізації та програмування : навч. посібник. Дніпро : УДУНТ, 2022. 158 с. URL: https://crust.ust.edu.ua/server/api/core/bitstreams/bfb9f823-172b-49bf-92a2-48bbbef5f113/content.
4. Беррі П. Head First Python. Легкий для сприйняття довідник. Харків : Фабула, 2023. 624 с.
5. Булгакова О. С., Зосімов В. В., Ходякова Г. В. Алгоритмізація і програмування: теорія та практика : навчальний посібник. Миколаїв : СПД Румянцева, 2021. 138 с. URL: http://dspace.mdu.edu.ua/jspui/handle/123456789/931.
6. Васильєв О. М. Програмування мовою Java. Тернопіль : Богдан НК, 2022. 696 с.
7. Висоцька В. А., Оборська О. В. Python: алгоритмізація та програмування : навчальний посібник. Львів: Новий світ-2000, 2021. 514 с.
8. Григорович В. Г. Алгоритмізація та програмування. Частина 1 : навчальний посібник. Львів : Магнолія 2006, 2023. 357 с.
9. Злобін Г. Г. Основи алгоритмізації та програмування мовою Сі : підручник. Київ : Каравела, 2022. 168 с.
10. Кублій Л. І. Алгоритмізація та програмування. Практикум : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2019. 209 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/a2c179fd-fb1f-4536-aae7-08a8928f8569/content
11. Лосєв М. Ю., Федорченко В. М. Програмування мовою Python : навчальний посібник. Харків; Львів : Новий Світ – 2000, 2024. 178 с.

12. Мартін Р. Чистий кодер. Кодекс поведінки для професійних розробників. Харків : Фабула, 2023. 256 с.
13. Пастернак І. І., Костик А. Т. Інструментальні засоби веб-технологій : навчальний посібник. Львів : Магнолія, 2024. 197 с.
14. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 2 : навчальний посібник. Львів : Новий Світ – 2000, 2020. 320 с.
15. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування. Частина 2. Модульне програмування : навчальний посібник. Львів : Львівський державний університет внутрішніх справ, 2024. 176 с. URL: https://dspace.lvduvs.edu.ua/handle/1234567890/6994
16. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування. Частина 1. Структурне програмування : навчальний посібник. Львів : Львівський державний університет внутрішніх справ, 2023. 240 с. URL: https://dspace.lvduvs.edu.ua/handle/1234567890/5515
17. Селіверстов Р., Мельничин А. Основи програмування мовою Python : навчальний посібник. Львів : ЛНУ, 2020. 190 с.
18. Спірінцева О. В., Литвинов О. А., Герасимов В. В. Java-технології та мобільні пристрої. Алгоритми і структури даних : навч. посіб. Дніпро : ДНУ, 2016. 140 с.
19. Трофименко О. Г., Манаков С. Ю., Ларін Д. Г. Основи програмної інженерії : навч.-метод. посібник. Одеса : Фенікс, 2022. 197 с. URL: https://dspace.onua.edu.ua/items/25698e60-3d50-42c0-a87a-27bd8b22a54d
20. Трофименко О. Г., Прокоп Ю. В., Логінова Н. І., Задерейко О. В. C++. Алгоритмізація та програмування : підручник. Одеса : Фенікс, 2019. 477 с. URL: https://dspace.onua.edu.ua/items/6c40c92b-c7d4-43ae-93da-e195f3daf3d1
21. Щербаков О. В., Парфьонов Ю. Е., Федорченко В. М. Основи об`єктно-орієнтованого програмування : навчальний посібник. Харків : ХНЕУ, 2019. 237с. URL: http://surl.li/oecpq.

Навчальне видання

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Методичні рекомендації

Укладачі: **Пархоменко** Олександр Юрійович

Тищенко Світлана Іванівна

Ємельянов Святослав Ігорович

Жебко Олександр Олегович

Богатєнкова Олександра Євгенівна

Формат 60х84 1/16. Ум. друк. арк. 4,0.
Тираж 20 прим. Зам. № ____

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54008, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013 р.