МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ МЕНЕДЖМЕНТУ

Кафедра економічної кібернетики,
комп'ютерних наук та інформаційних технологій

ВСТУП ДО ФАХУ

Методичні рекомендації до практичних занять
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
В84

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., доцент, доцент кафедри електроенергетики, електротехніки та електромеханіки Миколаївського національного аграрного університету.

Вступ до фаху : методичні рекомендації до практичних занять для В84 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. 169 с.

УДК 004

ЗМІСТ

Передмова 4

Практичне заняття 1. Вступ до комп'ютерних наук: визначення, основні поняття та історія розвитку IT 6

Практичне заняття 2. Особливості професійної підготовки фахівців у галузі комп'ютерних наук 14

Практичне заняття 3. Стандарти та системи підготовки фахівців з комп'ютерних наук 23

Практичне заняття 4. Етика та професійна відповідальність в IT-сфері 32

Практичне заняття 5. Огляд сучасних мов програмування та основних парадигм програмування 42

Практичне заняття 6. Методології розробки програмного забезпечення (agile, scrum, kanban тощо) 53

Практичне заняття 7. Основи проєктування та архітектури програмного забезпечення 64

Практичне заняття 8. Вступ до тестування та забезпечення якості програмного забезпечення 75

Практичне заняття 9. Структура та принципи роботи сучасних IT-компаній. 87

Практичне заняття 10. Різноманіття професій в IT-галузі 99

Практичне заняття 11. М'які навички (soft skills) для IT-фахівців 110

Практичне заняття 12. Сучасні тренди в IT-галузі 121

Практичне заняття 13. Поєднання навчання в університеті та практичної роботи в IT-компанії 132

Практичне заняття 14. Стажування, інтернатура та створення портфоліо для IT-фахівця 143

Практичне заняття 15. Самоосвіта, кар'єрне планування та розвиток у сфері комп'ютерних наук 154

Післямова 165

Рекомендовані джерела 167

ПЕРЕДМОВА

Сучасний світ неможливо уявити без інформаційних технологій, які трансформують медицину, освіту, економіку та повсякденне спілкування. Для студента-першокурсника спеціальності «Комп'ютерні науки» шлях у професію починається не лише з написання перших рядків коду, а й з формування цілісного розуміння екосистеми IT-індустрії. Ці методичні рекомендації, що складаються з 15 практичних занять, розроблені як «дорожня карта» для успішного старту у світі технологій.

Мета методичних рекомендацій – закласти фундаментальні знання про комп'ютерні науки як галузь, ознайомити з міжнародними стандартами підготовки фахівців та надати практичні інструменти для планування майбутньої кар'єри. Структура охоплює чотири ключові блоки професійного становлення.

Перший блок (заняття 1–4) присвячений концептуальним основам. Студенти дізнаються, що комп'ютерні науки – це не лише програмування, а вивчення теоретичних основ інформації та обчислень. Особлива увага приділяється історії розвитку IT та міжнародним стандартам (ACM, IEEE, ABET), які визначають компетенції фахівця світового рівня. Важливим аспектом є розгляд професійної етики: від конфіденційності даних до відповідальності за алгоритмічну упередженість ШІ.

Другий блок (заняття 5–8) занурює у технічну складову розробки. Методичні рекомендації знайомлять із різноманіттям мов програмування (Python, JavaScript, Java тощо) та ключовими парадигмами – від процедурної до об'єктно-орієнтованої. Студенти опановують принципи архітектури програмного забезпечення, зокрема модульність та слабку зв'язаність компонентів. Окремий розділ присвячено методологіям розробки (Agile, Scrum, Kanban) та забезпеченню якості (QA), де студенти вчаться не лише писати код, а й тестувати його за допомогою бібліотеки unittest.

Третій блок (заняття 9–11) фокусується на «м'яких навичках» (soft skills) та роботі в команді. Сучасна IT-компанія – це складна структура, де взаємодіють розробники, тестувальники, дизайнери та менеджери. Методичні рекомендації допомагають студентам розвинути критично важливі для галузі навички: активне слухання, вирішення конфліктів, тайм-менеджмент (методи Pomodoro та матриця Ейзенхауера) та ефективну комунікацію.

Четвертий блок (заняття 12–15) орієнтований на майбутнє та працевлаштування. Студенти досліджують сучасні тренди: штучний інтелект,

хмарні обчислення та блокчейн. Практичні завдання вчать створювати професійне резюме (CV), наповнювати портфоліо на GitHub та використовувати LinkedIn для нетворкінгу. Завершується курс концепцією Lifelong Learning – навчання протягом усього життя, що є необхідною умовою виживання у динамічному IT-середовищі.

Кожне практичне заняття побудоване за принципом поєднання теорії та практики. Студенти виконують інтерактивні вправи (наприклад, рольові ігри «Інтерв'ю з фахівцем» або симуляції Scrum-спринтів), працюють із реальними інструментами (Trello, Replit, Miro) та виконують аналітичні самостійні роботи.

Методичні поради студентам:

1. Сприймайте університет як систему, де кожна дисципліна – це частина вашої цілісної картини світу.

2. Не бійтеся помилятися. Перші алгоритми можуть бути неточними, але саме аналіз помилок формує досвід.

3. Розвивайте soft skills паралельно з технічними знаннями. В IT вміння домовитися часто цінується вище, ніж знання ще однієї мови програмування.

4. Будуйте свій профіль уже зараз. Ваші лабораторні та курсові роботи – це фундамент вашого майбутнього портфоліо.

Ці методичні рекомендації стануть вашим першим кроком до того, щоб не просто бути користувачем технологій, а стати їх свідомим творцем. Бажаємо успіхів у навчанні та професійному зростанні!

ПРАКТИЧНЕ ЗАНЯТТЯ 1
ВСТУП ДО КОМП'ЮТЕРНИХ НАУК: ВИЗНАЧЕННЯ, ОСНОВНІ ПОНЯТТЯ ТА ІСТОРІЯ РОЗВИТКУ ІТ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників цілісного уявлення про сферу комп'ютерних наук, її структуру, ключові поняття та історичний розвиток.

Основні завдання заняття:

1. Ознайомити з визначенням комп'ютерних наук як галузі знань та її місцем у сучасному світі.
2. Прослідкувати еволюцію обчислювальної техніки та інформаційних технологій від простих механізмів до сучасних систем.
3. Дослідити багатогранний вплив ІТ на різні сфери суспільства: освіту, медицину, економіку, культуру та комунікації.
4. Розвинути ключові навички («soft skills» та «hard skills»):

– *аналітичне мислення*: вміння декомпозувати складні завдання на прості кроки;

– *комунікація та колаборація*: робота в команді, представлення результатів;

– *інформаційна грамотність*: пошук, аналіз та верифікація даних з різних джерел;

– *самоорганізація*: планування часу та відповідальність за виконання самостійної роботи.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

Перед виконанням практичних завдань студенту рекомендується ознайомитися з наступними ключовими поняттями.

2.1. Комп'ютерні науки (Computer Science) – це наука про теоретичні основи інформації та обчислень, а також про практичні методи їх застосування в комп'ютерних системах. Вона охоплює все: від абстрактних алгоритмів і теорії інформації до конкретних питань програмування, розробки апаратного забезпечення та штучного інтелекту.

2.2. Поняття алгоритму

Алгоритм – це основа програмування. Це точний, скінченний набір інструкцій, які описують послідовність дій для досягнення певної мети.

Властивості алгоритмів:

- *дискретність* – розбиття процесу на окремі кроки.
- *детермінованість* – однозначність виконання кожної інструкції.
- *скінченність* – обов'язкове завершення після скінченної кількості кроків.
- *результативність* – отримання очікуваного результату.
- *масовість* – можливість застосування для класу однотипних задач.

2.3. Інформаційні технології (ІТ)

Це ширше поняття, яке включає не лише наукові основи, але й усі аспекти створення, зберігання, обміну та використання інформації за допомогою комп'ютерної техніки. Якщо Computer Science – це «наука про те, як це працює», то ІТ – це «практика застосування цих знань».

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп'ютер з доступом до мережі Інтернет;
- для аудиторної роботи: аркуші паперу А4, ручки;
- програмне забезпечення (на вибір):
  - текстовий процесор (Google Docs, Microsoft Word Online, LibreOffice).
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - інструменти для створення ментальних карт (MindMeister, Canva, XMind).

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Інтерактивна гра «Алгоритм повсякденного життя»

Мета завдання: навчитися формулювати алгоритми, розуміти важливість точності та однозначності інструкцій, що є критичним у програмуванні.

Теоретичне підґрунтя

У програмуванні комп'ютер «розуміє» лише чітко прописані команди. Будь-яка двозначність призводить до помилок (багів). Сьогодні ви спробуєте себе в ролі «програміста» та «комп'ютера», щоб на власному досвіді відчути цю потребу в точності.

Детальна інструкція виконання

Етап 1. Вибір задачі (5 хвилин).

Кожен студент індивідуально обирає простий, але не тривіальний процес. Уникайте дій, які виконуються «на автоматі», і оберіть ті, що мають чітку послідовність. *Приклади:* «Як відкрити електронний лист з новим паролем», «Як завантажити фотографію в Instagram з телефону», «Як приготувати бутерброд з маслом та сиром», «Як знайти визначення слова в онлайн-словнику».

Етап 2. Написання алгоритму (10 хвилин).

Запишіть алгоритм у вигляді нумерованого списку. Кожен крок має бути однією простою дією.

Порада: уявіть, що ви пояснюєте це роботу, який не має жодного «життєвого досвіду». Наприклад, для алгоритму «Налити води в чайник» поганим кроком буде «Наповнити чайник водою». Хорошим кроком буде: «1. Візьми чайник за ручку. 2. Підійди до крана з водою. 3. Відкрий кран, повернувши його проти годинникової стрілки. 4. Підстав чайник під струмінь води...» і т.д.

Етап 3. Виконання алгоритму (10 хвилин).

Об’єднайтеся в пари. Обміняйтеся алгоритмами.

Один студент стає «виконавцем». Він повинен виконувати інструкції буквально, не додаючи власного здорового глузду. Якщо в інструкції написано «Візьми чашку», але не сказано, де вона лежить, виконавець має запитати: «Я не можу продовжити, об’єкт "чашка" не знайдено».

Спостерігайте, на яких кроках виникають труднощі.

Етап 4. Рефлексія та обговорення (5 хвилин).

Дайте відповідь на питання:

- Чи були кроки, які ваш партнер виконав не так, як ви очікували? Чому?
- Де в алгоритмі виникла неоднозначність?
- Як це пов’язано з програмуванням? Чому комп’ютери не можуть «домислити» за програміста?

Критерії оцінювання (Завдання 1):

Вчасне виконання та участь у всіх етапах.

Логічність та несуперечливість алгоритму (він має бути здійсненним).

Чіткість формулювань (відсутність фраз на кшталт «потім зроби як треба»).

Активність під час обговорення помилок.

Завдання 2. Групова презентація «Віха в історії ІТ»

Мета завдання: розвинути навички командної роботи, публічних виступів та критичного осмислення історичних подій, які сформували сучасний світ технологій.

Детальна інструкція виконання

Етап 1. Формування груп та вибір теми (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть одну ключову подію. *Розширений список для вибору:*

- *Епоха механіки:* аналітична машина Чарльза Беббіджа (1837), програмований верстат Жаккарда.
- *Епоха електроніки:* ENIAC (1946), винахід транзистора (1947), створення мови програмування Fortran (1957).
- *Епоха персональних комп’ютерів:* Altair 8800 (1975), створення Apple I (1976), випуск IBM PC (1981).
- *Епоха інтернету:* ARPANET (1969), винайдення World Wide Web (1989), поява першого веб-браузера Mosaic (1993), запуск Google (1998).
- *Епоха мобільності та ШІ:* запуск iPhone (2007), вихід Android (2008), створення хмарних сервісів (AWS, 2006), перемога AlphaGo (2016), поява ChatGPT (2022).

Етап 2. Пошук інформації та створення контенту (15 хвилин).

Використовуйте надійні джерела: computerhistory.org, сайти музеїв, науково-популярні статті, Вікіпедію (звертайте увагу на виноски).

Розподіліть ролі в групі: хто шукає інформацію, хто оформлює слайди, хто готує усну доповідь.

Структура презентації (3 слайди):

Слайд 1 (Титульний). Назва події, дата (рік), фото/ілюстрація, прізвища учасників групи.

Слайд 2 (Суть події). Що це було? Хто винахідники/творці? Де це сталося? Які технічні характеристики мав винахід (якщо доречно)?

Слайд 3 (Вплив та значення). Як ця подія вплинула на розвиток технологій? Як вона змінила життя звичайних людей? Чи використовуємо ми результати цього винаходу сьогодні?

Порада. Не перевантажуйте слайди текстом. Використовуйте ключові слова, зображення, короткі схеми.

Етап 3. Презентація та обговорення (20 хвилин).

Кожна група має 2-3 хвилини на виступ. Говорити повинен **кожен** учасник групи (можна поділити слайди).

Уважно слухайте інші групи, фіксуючи ключові ідеї.

Етап 4. Загальна дискусія (5 хвилин).

Після всіх виступів модератор (викладач) ставить питання для обговорення:

- Який винахід ви вважаєте найбільш революційним і чому?
- Чи простежується зв'язок між різними подіями (наприклад, транзистор -> мікропроцесор -> персональний комп'ютер)?
- Якби не було інтернету, як би це змінило ваше сьогоднішнє навчання?

Критерії оцінювання (Завдання 2):

Змістовність: глибина розкриття теми, достовірність фактів.

Структурованість: чітка відповідність структурі, логічність викладу.

Якість презентації: дизайн, читабельність, доречність ілюстрацій.

Командна робота: залученість всіх учасників до виступу.

Ораторська майстерність: вільне володіння матеріалом, контакт з аудиторією.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Аналітичний огляд «Як ІТ змінюють наше життя»

Мета завдання: навчитися самостійно досліджувати вплив технологій на конкретну сферу, працювати з інформаційними джерелами, структурувати думки та презентувати результати у зручному форматі.

Вступ. Інформаційні технології сьогодні – це не лише програмування. Це рушійна сила змін в усіх галузях. Ваше завдання – обрати одну сферу та стати експертом з питання «Як ІТ її трансформують?».

Детальна інструкція виконання

Крок 1. Вибір сфери дослідження.

Оберіть сферу, яка вам справді цікава. Це може бути:

– Медицина: телемедицина, 3D-друк органів, AI для діагностики (наприклад, аналіз знімків МРТ), електронні карти пацієнтів.

– Освіта: онлайн-курси (Coursera, Prometheus), віртуальні класи (Zoom, Google Meet), гейміфікація навчання, інтерактивні підручники.

– Транспорт та логістика: безпілотні автомобілі (Tesla, Waymo), навігаційні системи (Google Maps, Waze), дрони-доставники, оптимізація маршрутів за допомогою ШІ (Uber, Glovo).

– Розваги та медіа: стримінгові сервіси (Netflix, Spotify), технології віртуальної (VR) та доповненої (AR) реальності, комп'ютерна графіка в кіно, індустрія відеоігор.

– Бізнес та фінанси: FinTech (Apple Pay, Приват24), інтернет-банкінг, криптовалюти та блокчейн, автоматизація бізнес-процесів (CRM, ERP-системи), електронна комерція (Amazon, Rozetka).

– Сільське господарство: точне землеробство (використання дронів та датчиків), автоматизація теплиць, прогнозування врожайності за допомогою AI.

Крок 2. Пошук та аналіз інформації.

Знайдіть 2-3 надійні джерела:

Науково-популярні статті (New Scientist, Wired, MIT Technology Review).

Аналітичні звіти (від компаній PwC, Deloitte, Gartner).

Відеолекції (TED Talks, YouTube-канали технологічних компаній).

Офіційні сайти розробників технологій.

Крок 3. Створення огляду (формат на вибір).

Варіант А: Текстовий огляд (200-300 слів).

Структура тексту:

1. Вступ (1 абзац): Яку сферу ви обрали та чому? Яке загальне твердження про вплив ІТ на неї?

2. Основна частина (2-3 абзаци): Опишіть 2-3 конкретні технології або приклади. Для кожної вкажіть, як саме вона змінює процеси, створює нові можливості або вирішує старі проблеми.

3. Висновок (1 абзац): Підсумуйте, чи є ці зміни позитивними? Чи є ризики або негативні наслідки? Яке майбутнє чекає на цю сферу?

Варіант Б: Ментальна карта (Mind Map).

У центрі – назва вашої сфери.

Перший рівень гілок – ключові технології (наприклад, для сфери "Освіта": "Відеоконференції", "Адаптивне навчання", "Відкриті ресурси").

Другий рівень гілок – конкретні приклади та їхній вплив (наприклад, для гілки "Відеоконференції": "Zoom" -> "Можливість навчатися з будь-якої точки світу").

Ментальна карта має бути візуально структурованою, з кольорами та іконками.

Крок 4. Оформлення та здача.

Вкажіть наприкінці роботи список використаних джерел (мінімум 1, бажано з гіперпосиланнями).

Завантажте файл (PDF, DOCX, JPG/PNG) або надайте публічне посилання на онлайн-дошку (MindMeister, Canva) у відповідний розділ курсу на платформі Moodle.

Критерії оцінювання самостійної роботи:

Відповідність темі: робота чітко стосується обраної сфери та впливу ІТ.

Змістовність: глибина аналізу, наявність конкретних прикладів, а не загальних фраз.

Структурованість: логічність викладу (для тексту) або ієрархічність (для карти).

Самостійність та доказова база: використання джерел, коректність інформації.

Охайність оформлення: відсутність граматичних помилок, читабельність.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Дайте означення комп’ютерних наук. Чим вони відрізняються від інформаційних технологій?
2. Назвіть п’ять основних властивостей алгоритму. Поясніть кожну на прикладі приготування кави.
3. Чому історія розвитку комп’ютерів є важливою для сучасного ІТ-фахівця?
4. Назвіть три ключові винаходи XX століття, які, на вашу думку, найбільше вплинули на появу сучасних смартфонів.
5. Оберіть одну сферу (наприклад, медицину) та опишіть один позитивний і один негативний (або ризикований) наслідок впровадження в ній ІТ.

6. Як навички, отримані під час створення «Алгоритму повсякденного життя», допоможуть вам у вивченні мов програмування?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

1. Computer History Museum (www.computerhistory.org) – Віртуальні тури, статті та фотоархіви з історії техніки.

2. TED Talks: Technology (www.ted.com/topics/technology) – Надихаючі виступи про майбутнє технологій.

3. Як це зроблено? (YouTube-канали): Існують десятки науково-популярних каналів (як українською, так і англійською), які пояснюють принципи роботи технологій (наприклад, Alpha Centauri, Kurzgesagt тощо).

4. Енциклопедія «Історія комп'ютерів» у Вікіпедії – Відправна точка для пошуку інформації за конкретними темами.

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Не бійтеся помилятися.** Перший алгоритм, швидше за все, буде неточним. Головне – зрозуміти *чому*.

2. **Ставте питання.** Якщо в завданні щось незрозуміло – питайте одногрупників або викладача. В IT вміння правильно поставити питання – половина успіху.

3. **Дивіться в майбутнє.** Виконуючи огляд, подумайте, як саме *ви* хотіли б вплинути на свою обрану сферу через 10-15 років.

ПРАКТИЧНЕ ЗАНЯТТЯ 2
ОСОБЛИВОСТІ ПРОФЕСІЙНОЇ ПІДГОТОВКИ ФАХІВЦІВ У ГАЛУЗІ КОМП'ЮТЕРНИХ НАУК

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників цілісного уявлення про структуру фахової підготовки, розуміння взаємозв'язку між теоретичними знаннями та практичними навичками, а також усвідомленого підходу до планування власної освітньої траєкторії.

Основні завдання заняття:

1. Ознайомити зі структурою освітньо-професійної програми «Комп'ютерні науки», її компонентами (обов'язкові та вибіркові дисципліни).
2. Проаналізувати роль фундаментальних (теоретичних) і прикладних (практичних) дисциплін у формуванні ІТ-фахівця.
3. Дослідити вимоги ринку праці до технічних навичок («hard skills») та універсальних компетентностей («soft skills») для різних ІТ-професій.
4. Розвинути ключові навички:

– *аналітичне мислення*: вміння структурувати навчальний матеріал, виділяти головне;

– *планування та самоорганізація*: розробка індивідуальної траєкторії навчання;

– *комунікація*: робота в команді, проведення інтерв'ю, представлення результатів;

– *дослідницькі навички*: пошук та аналіз інформації про професії та освітні програми.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

Перед виконанням практичних завдань студенту рекомендується ознайомитися з наступними ключовими поняттями.

2.1. Освітньо-професійна програма (ОПП)

ОПП «Комп'ютерні науки» – це системний документ, який визначає зміст навчання, перелік дисциплін, послідовність їх вивчення та очікувані результати. Вона складається з:

Обов’язкових дисциплін: формують фундаментальні знання та загальні компетентності (наприклад, «Вища математика», «Алгоритмізація та програмування», «Архітектура комп’ютера»).

Вибіркових дисциплін: дозволяють студенту формувати індивідуальну траєкторію навчання відповідно до власних інтересів та кар’єрних планів (наприклад, «Веб-дизайн», «Основи кібербезпеки», «Машинне навчання»).

2.2. Теоретичні та практичні дисципліни: баланс знань

Успішний ІТ-фахівець поєднує глибоке розуміння фундаментальних принципів із здатністю застосовувати їх на практиці.

Теоретичні дисципліни (алгоритми, дискретна математика, теорія ймовірностей) формують науковий світогляд, розвивають абстрактне мислення та вчать знаходити ефективні рішення складних задач. Вони змінюються повільніше, ніж технології.

Практичні дисципліни (конкретні мови програмування, фреймворки, інструменти розробки) дають навички, необхідні для виконання реальних завдань. Вони швидко оновлюються, тому фахівець має вчитися постійно.

2.3. Hard Skills та Soft Skills

Hard Skills (технічні навички): конкретні, вимірювані знання та вміння, необхідні для виконання професійних обов’язків. *Приклади:* знання мови Python, вміння писати SQL-запити, налаштовувати комп’ютерні мережі.

Soft Skills (універсальні/м’які навички): особистісні якості та комунікативні здібності, які допомагають ефективно взаємодіяти з людьми, вирішувати проблеми та керувати власним часом. *Приклади:* комунікабельність, командна робота, критичне мислення, адаптивність, тайм-менеджмент.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет;
- для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постера);
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online);
  - текстовий процесор (Google Docs, Microsoft Word Online);

- електронні таблиці (Google Sheets, Microsoft Excel Online) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Аналіз структури ІТ-освіти

Мета завдання: навчитися аналізувати освітню програму, розуміти призначення різних дисциплін та їхню роль у підготовці сучасного ІТ-фахівця.

Теоретичне підґрунтя

Уявіть, що освітня програма – це «карта знань». Одні предмети (теоретичні) закладають фундамент, навчають мислити. Інші (практичні) – дають конкретні інструменти для роботи. Сьогодні ви навчитеся читати цю карту.

Детальна інструкція виконання

Етап 1. Формування груп та вибір об'єкта аналізу (5 хвилин).

Об'єднайтесь у групи по 3-4 особи.

Оберіть освітню програму для аналізу:

Варіант А: Офіційний сайт вашого університету (розділ «Освітні програми», «Навчальні плани»).

Варіант Б: Програма відомого закордонного університету, наприклад, MIT OpenCourseWare (курси з напряму Electrical Engineering and Computer Science).

Варіант В: Освітня програма будь-якого іншого українського університету, що готує комп'ютерних науковців (КПІ ім. Ігоря Сікорського, Львівська політехніка, ХНУРЕ тощо).

Етап 2. Аналіз та класифікація дисциплін (10 хвилин).

Перегляньте перелік дисциплін 1-2 курсів.

Оберіть **5-7 ключових дисциплін** (наприклад, Алгоритми та структури даних, Об'єктно-орієнтоване програмування, Вища математика, Бази даних, Комп'ютерна графіка, Людино-машинний інтерфейс).

Для кожної дисципліни проведіть аналіз:

1. Класифікація: Це теоретична чи практична дисципліна? Чому ви так вважаєте?

2. Призначення: Які знання/навички вона формує?

3. Застосування: Наведіть 1-2 приклади, як знання з цієї дисципліни використовуються в реальній ІТ-розробці. (Наприклад, «Дискретна математика потрібна для розуміння логіки роботи баз даних та оптимізації запитів»).

Етап 3. Створення презентації/постеру (10 хвилин).

Оформіть результати у вигляді 2-3 слайдів або постера на аркуші А4.

Структура презентації/постеру:

Заголовок: назва університету та освітньої програми.

Таблиця або схема: перелік дисциплін з їхнім розподілом на теоретичні та практичні. Можна використати два стовпчики або два кольори.

Приклади: для 2-3 найцікавіших дисциплін коротко опишіть їхнє практичне застосування.

Етап 4. Презентація та обговорення (15 хвилин).

Кожна група має 2-3 хвилини на виступ.

Після всіх виступів проведіть обговорення:

– Чи відрізняються підходи до навчання в різних університетах? Як саме?

– Чи можна стати хорошим програмістом, вивчаючи лише практичні дисципліни?

– Чому в програмі так багато математичних дисциплін?

Критерії оцінювання (Завдання 1):

Глибина аналізу: чи було просто переписано назви, чи зроблено спробу класифікації та пояснення?

Логічність: чи обґрунтовано віднесення дисциплін до теоретичних/практичних?

Практичні приклади: наявність конкретних прикладів застосування знань.

Якість презентації: чіткість, структурованість, дизайн.

Участь у дискусії: активність під час обговорення.

Завдання 2. Рольова гра «Інтерв'ю з ІТ-фахівцем»

Мета завдання: розвинути навички комунікації, вміння ставити питання та аналізувати отриману інформацію; дослідити вимоги до різних ІТ-професій зсередини.

Теоретичне підґрунтя

У реальному житті ІТ-фахівці постійно проходять співбесіди. Але ще частіше їм доводиться спілкуватися з колегами, менеджерами, замовниками, пояснюючи складні технічні речі простою мовою. Це завдання – тренування цієї навички, а також навички активного слухання.

Детальна інструкція виконання

Етап 1. Підготовка до інтерв'ю (5 хвилин).

Об'єднайтесь у пари. Розподіліть ролі: «Інтерв'юер» та «ІТ-фахівець».

«Інтерв'юер» обирає професію, про яку питатиме (наприклад: Backend-розробник, Frontend-розробник, DevOps-інженер, Data Scientist, Тестувальник (QA), Системний адміністратор, Project Manager).

«ІТ-фахівець» готується відповідати від імені людини, яка працює в цій професії 2-3 роки. Він може скористатися підказками (див. Додаткові ресурси в кінці документа або власні знання).

Етап 2. Проведення інтерв'ю (10 хвилин).

«Інтерв'юер» ставить запитання. Мета – з'ясувати якомога більше деталей про повсякденну роботу.

Приклади запитань:

Опишіть свій типовий робочий день. З чого він починається?

Які інструменти та технології ви використовуєте найчастіше?

З якими людьми (посадами) ви найбільше взаємодієте?

Яка найскладніша проблема, яку вам довелося вирішувати на роботі?

Які 3 головні «hard skills» потрібні початківцю, щоб влаштуватися на вашу посаду?

Які «soft skills» є критично важливими у вашій роботі?

Чого вам не вистачало після університету, і де ви це наздоганяли?

«Інтерв'юер» робить короткі нотатки.

Етап 3. Фіксація підсумків (5 хвилин).

Разом обговоріть почуте.

Запишіть короткий підсумок (5-7 речень) на окремому аркуші.

Структура підсумку:

– Назва професії.
– Ключові технічні навички (hard skills).
– Ключові універсальні навички (soft skills).
– Як, за словами «фахівця», краще розвивати ці навички.

Етап 4. Презентація підсумків (10 хвилин).

Викладач запрошує 2-3 пари (за бажанням) поділитися найцікавішими висновками з групою.

Проведіть міні-обговорення:

Що вас здивувало в отриманих відповідях?

Чи збігаються вимоги до різних професій? Що спільного в усіх?

Критерії оцінювання (Завдання 2):

Активність: участь обох партнерів у грі.

Якість підготовки: змістовність запитань (для інтерв'юера) та відповідей (для фахівця).

Аналіз: чіткість та інформативність письмового підсумку.

Участь в обговоренні: готовність ділитися висновками.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

План розвитку навичок IT-фахівця

Мета завдання: сформувати навички цілепокладання, самостійного дослідження ринку праці та планування власного професійного розвитку.

Вступ. Кар'єра в IT – це не випадковість, а результат усвідомлених кроків. Найуспішніші фахівці постійно вчаться й розвиваються. Цей план стане вашою першою дорожньою картою у світ професії.

Детальна інструкція виконання

Крок 1. Вибір професії та дослідження.

Оберіть одну IT-професію, яка вас найбільше цікавить. Проведіть міні-дослідження, використовуючи:

Сайти з вакансіями: djinni.co, work.ua, robota.ua (проаналізуйте 3-5 вакансій, випишіть вимоги, які повторюються).

Професійні блоги та статті: Medium, DOU.ua, Habr.

Описи курсів: на Prometheus, Coursera, Udemy (подивіться, чому навчають на курсах для початківців).

Приклад: Якщо оберете «Frontend-розробника», ви побачите, що майже скрізь вимагають HTML, CSS, JavaScript та знання будь-якого фреймворку (React, Vue, Angular).

Крок 2. Створення плану розвитку (формат на вибір).

Варіант А: Таблиця (рекомендовано).

Створіть таблицю в Google Sheets або Excel з такими колонками:

Категорія	Навичка / Знання	Пріоритет (1-3)	Ресурси для вивчення (назва + посилання)	Приблизний термін	Статус (план/в процесі)
Hard Skills	Мова Python	1	[Prometheus: "Python для початківців"]	2 місяці	План
	Основи SQL	2	[Книга "SQL для простих смертних"]	1 місяць	План
	Git	2	[YouTube-канал "Фрілансер по життю"]	2 тижні	План
Soft Skills	Англійська мова (B1)	1	[Додаток Duolingo]	Постійно	В процесі
	Командна робота	3	Курс на Coursera: "Teamwork Skills"	-	План
	Тайм-менеджмент	2	[Стаття: "Метод Pomodoro"]	-	План

Варіант Б: Текстовий документ.

Оформіть план у вигляді структурованого тексту:

1. Моя цільова професія: [назва професії]. Чому я її обрав/обрала? (Коротке пояснення на 50-100 слів).

2. Ключові технічні навички:

- *Навичка 1:* [назва]. План вивчення: [Ресурс 1], [Ресурс 2].
- *Навичка 2:* [назва]. План вивчення: [Ресурс 1], [Ресурс 2].

3. Ключові універсальні навички:

- *Навичка 1:* [назва]. Як планую розвивати: [Участь в дискусіях, вивчення англійської тощо].

4. Висновок: Як цей план допоможе мені досягти мети.

Крок 3. Оформлення та здача.

Обов’язково додайте мінімум 2 активні посилання на ресурси (онлайн-курси, статті, відео, книги), які ви плануєте використовувати.

Додайте короткий опис (50-100 слів) про мотивацію вибору професії.

Завантажте готовий файл (PDF, DOCX, XLSX) або надайте посилання на Google Doc/Sheet з відкритим доступом у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Обґрунтованість вибору: чітке пояснення, чому обрано саме цю професію.

Повнота дослідження: навички відповідають реальним вимогам ринку (вивченим вакансіям).

Структурованість: зручний формат (таблиця або чіткий текст), легкість сприйняття.

Реалістичність: чи є план досяжним? Чи вказані конкретні ресурси?

Самостійність: використання джерел, унікальність підходу.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Поясніть різницю між обов'язковими та вибірковими дисциплінами. Чому вибіркові дисципліни важливі для студента?
2. Чому в освітній програмі з комп'ютерних наук так багато математичних дисциплін? Наведіть приклад застосування вищої математики в програмуванні.
3. Як співвідносяться теоретичні та практичні дисципліни у вашому навчанні? Чи можна замінити одне одним?
4. Назвіть три «hard skills» та три «soft skills», які є критичними для обраної вами IT-професії. Чому?
5. Які ресурси для самоосвіти ви плануєте використовувати під час навчання в університеті?
6. Чи змінилося ваше уявлення про IT-професії після рольової гри «Інтерв'ю»? Як саме?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

1. DOU.ua – найбільша спільнота програмістів України. Розділи «Ринок праці», «Блоги» – статті про те, як влаштована робота в різних компаніях.
2. djinni.co – сайт пошуку роботи. Корисний для аналізу вимог до вакансій (розділ «Секретні вакансії» показує реальний зріз ринку).
3. freeCodeCamp.org – безкоштовний ресурс для вивчення програмування з великою кількістю статей про кар'єру в IT.
4. Coursera / Prometheus – платформи з курсами, де можна подивитися програми навчання для різних спеціалізацій.

Ресурси для розвитку Soft Skills:

1. TED Talks за темами «Communication», «Leadership», «Time management».

2. Project Management Institute (PMI) – матеріали про командну роботу та управління (англ.).

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Сприймайте університет як систему, а не як набір предметів.** Кожна дисципліна – це пазл, який формує вашу цілісну картину світу. Навіть якщо зараз якийсь предмет здається вам «зайвим», запитайте викладача, як він пов’язаний з іншими.

2. **Почніть будувати свій профіль вже зараз.** GitHub, LinkedIn, участь у студентських проектах – це ваш майбутній портфоліо. План розвитку, який ви склали, – перший крок до його наповнення.

3. **Не ігноруйте Soft Skills.** В IT ви працюєте в команді. Вміння домовитися, пояснити і почути іншого часто цінується вище, ніж знання ще однієї мови програмування.

ПРАКТИЧНЕ ЗАНЯТТЯ 3
СТАНДАРТИ ТА СИСТЕМИ ПІДГОТОВКИ ФАХІВЦІВ З КОМП'ЮТЕРНИХ НАУК

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів системного розуміння стандартів вищої освіти в галузі комп'ютерних наук, механізмів забезпечення якості освіти (акредитації) та професійних компетенцій, визнаних на міжнародному рівні.

Основні завдання заняття:

1. Ознайомити з міжнародними (ACM, IEEE) та національними стандартами підготовки фахівців з комп'ютерних наук.
2. Дослідити структуру та вимоги освітньо-професійних програм провідних українських та закордонних університетів.
3. Проаналізувати ключові професійні компетенції (knowledge areas), визначені міжнародними спільнотами.
4. Розвинути ключові навички:

– *порівняльний аналіз*: вміння знаходити спільне та відмінне в освітніх системах;

– *критичне мислення*: оцінка відповідності освітніх програм вимогам ринку праці;

– *інформаційна грамотність*: робота з першоджерелами (сайти університетів, професійних асоціацій);

– *комунікація*: участь у дискусії, аргументація власної позиції.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Поняття стандарту вищої освіти

Стандарт вищої освіти – це сукупність вимог до освітньої програми, які визначають її зміст, обсяг, рівень підготовки випускників та форми атестації. В Україні діють Державні стандарти вищої освіти, які встановлюють перелік компетентностей та нормативний зміст підготовки для кожної спеціальності (для нас – спеціальність 122 «Комп'ютерні науки»).

2.2. Міжнародні професійні асоціації та їхні рекомендації

Найавторитетніші організації, що формують світові стандарти IT-освіти:

• ACM (Association for Computing Machinery) – найбільше міжнародне товариство в галузі комп'ютерних наук. Випускає рекомендації «Computer Science Curricula» (спільно з IEEE), які визначають, які знання має містити сучасна програма з CS.

• IEEE Computer Society – підрозділ Інституту інженерів електротехніки та електроніки, який фокусується на комп'ютерних науках та інженерії. Також бере участь у розробці освітніх стандартів.

• ABET (Accreditation Board for Engineering and Technology) – американська рада з акредитації в галузі інженерії та технологій. ABET акредитує освітні програми з комп'ютерних наук по всьому світу, підтверджуючи, що вони відповідають високим міжнародним вимогам.

2.3. Компетентнісний підхід

Сучасна освіта базується на компетентнісному підході. Це означає, що результатом навчання є не просто «знання», а здатність випускника застосовувати ці знання для вирішення реальних задач.

Інтегральна компетентність – здатність розв'язувати складні спеціалізовані задачі та практичні проблеми у сфері комп'ютерних наук.

Загальні компетентності – універсальні навички (soft skills): здатність до абстрактного мислення, аналізу та синтезу, знання англійської мови, навички міжособистісної взаємодії.

Фахові компетентності – спеціальні знання та вміння (hard skills): здатність проєктувати архітектуру програмного забезпечення, розробляти алгоритми, забезпечувати кібербезпеку тощо.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

• персональний комп'ютер з доступом до мережі Інтернет;

• для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постерів);

• програмне забезпечення (на вибір):

– сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).

– текстовий процесор або електронні таблиці (Google Docs/Sheets, Microsoft Word/Excel) – для самостійної роботи.

– браузер для доступу до сайтів університетів та професійних асоціацій.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Порівняльний аналіз освітніх програм

Мета завдання: навчитися проводити порівняльний аналіз освітніх систем, виявляти сильні сторони власної програми та можливі точки для зростання, орієнтуючись на міжнародний досвід.

Теоретичне підґрунтя

Порівняння з кращими зразками (бенчмаркінг) – це стандартний інструмент удосконалення в будь-якій сфері. Аналізуючи програми топових університетів світу, ми можемо краще зрозуміти глобальні тренди в освіті та вимоги до фахівців.

Детальна інструкція виконання

Етап 1. Формування груп та вибір об’єкта для порівняння (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть 2 програми для порівняння:

Базова програма: Освітньо-професійна програма (ОПП) «Комп’ютерні науки» вашого університету. Знайдіть її текст на сайті університету (зазвичай у розділі «Вступнику» або «Освіта»).

Закордонна програма: Оберіть один із провідних університетів світу. *Рекомендації:*

- MIT (США): Electrical Engineering and Computer Science. Шукайте навчальний план (Course 6-3).
- Stanford (США): Computer Science. Розділ Undergraduate Programs.
- University of Helsinki (Фінляндія): Computer Science. Відома своїм практичним підходом.
- ETH Zurich (Швейцарія): Computer Science. Один з найкращих технічних університетів Європи.

Етап 2. Аналіз за критеріями (10 хвилин).

Уважно перегляньте обидві програми. Заповніть таблицю порівняння (можна на аркуші або в спільному документі):

Критерій для порівняння	Наш	Закордонний

	університет (назва)	університет (назва)
Структура програми (скільки років, поділ на курси)		
Ключові обов'язкові дисципліни (перелічіть 5-7 основних)		
Співвідношення теорії та практики (є багато лабораторних? курсових проєктів?)		
Наявність вибіркових дисциплін (скільки відсотків на вибір студента?)		
Практична підготовка (чи є обов'язкова виробнича практика? стажування?)		
Фокус програми (науково-дослідницький, інженерно-прикладний, бізнес-орієнтований?)		

Етап 3. Створення презентації/постеру (10 хвилин).

Оформіть результати порівняння у вигляді 2-3 слайдів або постера.

Структура:

Слайд 1. Назви університетів, що порівнюються.

Слайд 2. Таблиця або схема з основними спільними та відмінними рисами. Виділіть кольорами (наприклад, зеленим – спільне, синім – відмінності).

Слайд 3. Висновки. Що спільного? Що відмінного? Що можна було б запозичити з досвіду закордонного університету для покращення вашої програми? Як ці програми готують до вимог ІТ-ринку?

Етап 4. Презентація та обговорення (15 хвилин).

Кожна група презентує свої висновки (2-3 хвилини).

Загальна дискусія:

Які відмінності виявилися найбільш несподіваними?

Чи вважаєте ви, що наша програма потребує змін? Яких саме?

Чому в закордонних програмах часто більше уваги приділяється вибірковим дисциплінам?

Критерії оцінювання (Завдання 1)

Глибина аналізу: чи було порівняння поверхневим, чи зроблено спробу проаналізувати сутнісні характеристики програм?

Структурованість: наявність чітких критеріїв порівняння, логічність викладу.

Наочність: якість оформлення таблиці/схеми, зрозумілість матеріалу.

Обґрунтованість висновків: чи підкріплені висновки фактами з аналізу?

Завдання 2. Дискусія «Компетенції ІТ-фахівця»

Мета завдання: розвинути навички критичного мислення, аргументації та командної роботи; ознайомитися з професійними стандартами, визнаними міжнародною спільнотою.

Теоретичне підґрунтя

ACM та IEEE регулярно публікують документ «Computer Science Curricula» (рекомендації щодо змісту освіти). Він містить перелік так званих «Knowledge Areas» (галузей знань) – фундаментальних тем, які повинен освоїти випускник. Сьогодні ми спробуємо визначити, що з цього переліку є найважливішим.

Детальна інструкція виконання

Етап 1. Ознайомлення з рекомендаціями (5 хвилин).

Об’єднайтесь у пари або міні-групи по 3 особи.

Швидко перегляньте інформацію на сайтах:

- ACM Education: https://www.acm.org/education
- IEEE Computer Society: https://www.computer.org/education

Зверніть увагу на ключові напрямки, які вони виділяють (наприклад: Algorithms, Programming Languages, Software Engineering, Systems, Human-Computer Interaction, Information Management, Social and Professional Issues).

Етап 2. Визначення топ-5 компетенцій (10 хвилин).

Уявіть, що ви – робоча група при Міністерстві освіти, яка має визначити 5 найважливіших компетенцій для ІТ-фахівця 2025-2030 років.

Ваше завдання – скласти такий список. Він може включати як технічні знання (hard skills), так і універсальні навички (soft skills).

Для кожної з 5 компетенцій запишіть обґрунтування (2-3 речення), чому вона потрапила до списку. Наприклад:

Компетенція: здатність до безперервного навчання (Continuous Learning).

Обґрунтування: технології змінюються щороку. Фахівець, який не вчиться самостійно, втрачає конкурентоспроможність через 3-5 років. Ця компетенція є запорукою довготривалої кар’єри.

Компетенція: розуміння алгоритмів та структур даних.

Обґрунтування: це фундамент, який не залежить від мови програмування. Він дозволяє писати ефективний та оптимальний код, а не просто "щось, що працює".

Етап 3. Презентація та загальна дискусія (15 хвилин).

Представники 2-3 груп виходять із своїми списками (по 1-2 хвилини на групу).

Після виступів проведіть загальну дискусію:

- Які компетенції згадувалися найчастіше? Чи можна скласти єдиний «топ-5» для всієї групи?
- Які компетенції, на вашу думку, найскладніше розвинути під час навчання в університеті? Чому?
- Чи згодні ви з твердженням, що знання англійської мови – це така ж професійна компетенція, як і знання мови програмування?

Критерії оцінювання (Завдання 2)

Логічність списку: чи є компетенції різноплановими, чи охоплюють різні аспекти діяльності?

Якість обґрунтування: переконливість аргументів, наявність прикладів.

Активність: участь в обговоренні, вміння слухати інших та реагувати на їхні аргументи.

Командна робота: чи було чутно думку кожного учасника групи.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Порівняльна таблиця стандартів підготовки ІТ-фахівців

Мета завдання: поглибити навички самостійного дослідження, систематизації інформації та аналізу відповідності освітніх програм міжнародним стандартам.

Вступ. Коли український диплом визнають за кордоном? Це залежить від того, наскільки наша освітня програма відповідає міжнародним стандартам. Ваше завдання – провести власне невелике дослідження та зрозуміти, де ми знаходимось на цій "мапі стандартів".

Детальна інструкція виконання

Крок 1. Вибір об’єктів для порівняння.

Об’єкт А (Україна). Освітньо-професійна програма «Комп’ютерні науки» вашого університету. Завантажте її з сайту. Зверніть увагу на розділи «Компетентності випускника» та «Програмні результати навчання».

Об’єкт Б (Закордонна програма). Оберіть програму бакалаврату з комп’ютерних наук в одному з університетів, який має акредитацію ABET (це

важливий маркер якості). *Приклади:* Purdue University, University of Illinois at Urbana-Champaign, University of Toronto. Знайдіть на їхньому сайті розділ "Program Educational Objectives" або "Student Outcomes".

Крок 2. Аналіз стандартів

Дослідіть, на які стандарти спираються ці програми:

- Для української програми – це Стандарт вищої освіти України за спеціальністю 122 (знайдіть його текст на сайті МОН або свого університету).
- Для закордонної програми – це, найімовірніше, критерії ABET або рекомендації ACM/IEEE.

Крок 3. Створення порівняльної таблиці.

Створіть таблицю (в Google Sheets, Excel або Word) за зразком:

Критерій порівняння	**Український університет (назва)**	**Закордонний університет (назва)**	**Коментар / Висновок**
Назва стандарту, на який орієнтується програма	[наприклад: Стандарт ВО України, 122]	[наприклад: ABET Computing Accreditation Commission]	
Орган, що затверджує/акредитує	МОН України	ABET / національні агентства	
Перелік ключових компетенцій (3-5 прикладів)	(виписати з програми)	(виписати з програми)	Чим схожі? Чим відрізняються?
Вимоги до практичної підготовки	(курсові, практика, диплом)	(capstone project, internship)	
Можливість отримання професійних сертифікацій	(чи є в програмі?)	(часто інтегровані, напр. Cisco, Microsoft)	

Крок 4. Написання аналітичного коментаря

Додайте до таблиці короткий текст (100-150 слів), у якому дайте відповідь на питання:

- Наскільки українська програма відповідає міжнародним стандартам за змістом?
- Що, на вашу думку, є сильними сторонами нашої програми?
- Що можна було б покращити, щоб випускники були більш конкурентоспроможними на глобальному ринку праці?

Крок 5. Оформлення та здача

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2: посилання на програми університетів та/або сайти ACM/IEEE/ABET).

Завантажте готовий файл (PDF, DOCX, XLSX) або надайте посилання на онлайн-документ з відкритим доступом у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Повнота дослідження: чи обидві програми проаналізовано глибоко, чи взято лише поверхневі дані?

Структурованість: таблиця є зручною для читання, критерії порівняння логічні.

Аналітичність коментаря: наявність власних висновків, а не просто переказ фактів.

Доказовість: використання першоджерел (сайтів, стандартів), коректність посилань.

Оформлення: чіткість, відсутність граматичних помилок.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Що таке стандарт вищої освіти? Яку роль він відіграє в підготовці фахівців?
2. Які міжнародні організації визначають стандарти IT-освіти? Чим вони відомі?
3. Поясніть, що таке компетентнісний підхід. Чим «компетентність» відрізняється від просто «знання»?
4. Назвіть 3 загальні (soft skills) та 3 фахові (hard skills) компетентності, які є обов’язковими для випускника вашої спеціальності згідно зі стандартом.
5. Що таке акредитація ABET? Чому наявність такої акредитації є престижною для університету?
6. За результатами вашого самостійного дослідження, який головний висновок щодо порівняння українських та закордонних стандартів ви можете зробити?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Офіційні документи та стандарти

1. Стандарт вищої освіти України: перший (бакалаврський) рівень, галузь знань 12 «Інформаційні технології», спеціальність 122 «Комп'ютерні науки». (Знайдіть актуальну версію на сайті МОН України).

2. ACM/IEEE Computer Science Curricula 2023 (або остання версія). Доступно на сайті ACM.

Ресурси для порівняльного аналізу

1. MIT OpenCourseWare (OCW): https://ocw.mit.edu – Вільний доступ до матеріалів курсів та навчальних планів.

2. Stanford Engineering Everywhere (SEE): https://see.stanford.edu – Аналогічний ресурс Стенфорда.

3. ABET Accredited Programs Search: http://main.abet.org/aps/Accreditedprogramsearch.aspx – Пошук акредитованих програм по всьому світу.

Ресурси про компетенції та кар'єру

1. DOU.ua "Як стати …" (Серія статей): Корисні матеріали про те, які знання потрібні для різних IT-спеціалізацій в Україні.

2. Coursera / Prometheus: Перегляньте програми професійних сертифікацій (наприклад, Google IT Support, IBM Data Science), щоб побачити, які компетенції вимагають роботодавці.

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Диплом – це не лише "корочка", а й відповідність стандарту.** Освітній стандарт гарантує, що де б ви не навчались, ви отримаєте певний набір знань. Це своєрідний "знак якості".

2. **Вивчайте міжнародні стандарти самостійно.** Рекомендації ACM/IEEE – це дороговказ для саморозвитку. Якщо якась тема з їхнього переліку вам незнайома, це привід вивчити її додатково.

3. **Компетенції – це ваш актив.** Сприймайте навчання не як процес "здачі іспитів", а як процес накопичення компетенцій. Кожен курс дає вам кілька "цеглинок" у вашу професійну стіну. Будьте свідомими, які саме цеглинки ви отримуєте.

ПРАКТИЧНЕ ЗАНЯТТЯ 4
ЕТИКА ТА ПРОФЕСІЙНА ВІДПОВІДАЛЬНІСТЬ В ІТ-СФЕРІ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників усвідомлення морально-етичних аспектів професійної діяльності в галузі інформаційних технологій, розуміння відповідальності перед суспільством за створювані продукти та рішення.

Основні завдання заняття:

1. Ознайомити з основними етичними принципами та кодексами професійної поведінки в ІТ (ACM, IEEE).
2. Дослідити ключові етичні проблеми сучасної ІТ-індустрії: конфіденційність даних, алгоритмічна упередженість, інтелектуальна власність, соціальна відповідальність.
3. Розвинути навички аналізу етичних дилем та прийняття зважених рішень у професійному контексті.
4. Розвинути ключові навички:

– *критичне мислення*: вміння розпізнавати етичні проблеми там, де вони не лежать на поверхні.

– *аргументація та дебати*: формулювання та захист власної позиції з повагою до опонента.

– *комунікація*: робота в команді, представлення результатів аналізу.

– *дослідницькі навички*: пошук інформації про реальні етичні скандали та прецеденти.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Професійна етика в ІТ

Етика – це система моральних принципів, які регулюють поведінку людини або групи людей. Професійна етика ІТ-фахівця – це застосування загальних етичних норм до специфічних ситуацій, що виникають при розробці, впровадженні та використанні інформаційних технологій. Чому це важливо? Тому що програмне забезпечення та алгоритми дедалі більше впливають на життя людей: від визначення кредитного рейтингу до діагностики захворювань і навіть винесення судових вироків.

2.2. Ключові етичні поняття

Конфіденційність (Privacy): право особи на захист своїх персональних даних від несанкціонованого збору, використання та поширення. Для IT-фахівця це означає відповідальність за проєктування систем, які захищають дані користувачів.

Інтелектуальна власність (Intellectual Property): результати інтелектуальної діяльності, які охороняються законом (авторське право, патенти, ліцензії). Використання чужого коду без дозволу або порушення ліцензійних угод є неетичним і часто незаконним.

Прозорість та підзвітність (Transparency & Accountability): алгоритми, особливо ті, що приймають важливі рішення (наприклад, у сфері фінансів, охорони здоров'я), мають бути зрозумілими, а їхні творці – нести відповідальність за наслідки їхньої роботи.

Алгоритмічна упередженість (Algorithmic Bias): ситуація, коли алгоритм систематично дискримінує певні групи людей через неякісні дані, на яких він навчався, або через помилки в його проєктуванні.

2.3. Професійні кодекси етики

Провідні міжнародні організації розробили кодекси, які визначають стандарти поведінки для IT-фахівців:

ACM Code of Ethics and Professional Conduct: містить 24 принципи, згруповані у чотири розділи: загальні моральні принципи, професійні обов'язки, обов'язки перед лідерами та організаціями, відповідність кодексу.

IEEE Code of Ethics: фокусується на зобов'язаннях інженерів перед суспільством, клієнтами, роботодавцями та колегами, наголошуючи на безпеці, чесності та уникненні конфліктів інтересів.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп'ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки.
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - текстовий процесор (Google Docs, Microsoft Word Online).
  - інструменти для створення інфографіки (Canva, Piktochart) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Кейс-аналіз етичної дилеми в ІТ

Мета завдання: навчитися виявляти етичні проблеми в реальних професійних ситуаціях, аналізувати їх з різних точок зору та пропонувати обґрунтовані рішення.

Теоретичне підґрунтя

У реальному житті рідко буває однозначно "правильне" або "неправильне" рішення. Найчастіше ми стикаємося з дилемами, де кожен вибір має свої позитивні та негативні наслідки. Вміння аналізувати такі ситуації є ключовою компетенцією відповідального фахівця.

Детальна інструкція виконання

Етап 1. Формування груп та вибір кейсу (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть один із запропонованих кейсів або скористайтеся ресурсами (наприклад, Центр етики Університету Санта-Клари).

Кейс 1: Дискримінаційний алгоритм підбору персоналу.

Ви – команда розробників у великій технологічній компанії. Ви створили AI-систему для автоматичного відбору резюме. Після запуску ви помічаєте, що система систематично відхиляє резюме жінок та кандидатів певних національностей. Аналіз показує, що алгоритм навчався на історичних даних компанії, де більшість успішних наймань були чоловіками певної етнічної групи. Керівництво не хоче зупиняти використання системи, адже вона економить багато часу.

Кейс 2: Приховування вразливості.

Ви – тестувальник (QA) програмного забезпечення. Під час тестування нового мобільного банкінгу ви знаходите критичну вразливість, яка потенційно може дозволити зловмисникам отримати доступ до рахунків користувачів. Ви доповідаєте про це керівнику проєкту. Керівник визнає проблему, але просить нікому не повідомляти про неї (ні клієнту-банку, ні тим паче публічно), поки вони "тихо" не виправлять її в наступному оновленні, яке вийде за 3 місяці. Він пояснює це тим, що публічне визнання проблеми зараз призведе до паніки клієнтів, репутаційних втрат і зірваних контрактів.

Кейс 3: Дані користувачів як товар.

Ви – аналітик даних у стартапі, який розробив безкоштовний мобільний застосунок для відстеження фізичної активності. Застосунок збирає величезну кількість даних: геолокацію, маршрути пробіжок, пульс, якість сну. Компанія пропонує продавати ці знеособлені (як вони вважають) дані страховим компаніям. Страхові компанії зможуть використовувати їх для коригування вартості полісів для клієнтів. В угоді користувача (яку ніхто не читає) є пункт про передачу даних третім особам у маркетингових цілях.

Етап 2. Аналіз кейсу (10 хвилин).

Використовуйте наступну схему для обговорення в групі:

1. Ідентифікація проблеми: сформулюйте етичну дилему одним-двома реченнями.

2. Зацікавлені сторони: хто постраждає або виграє від різних рішень? (Користувачі, компанія, суспільство, ви самі).

3. Порушені принципи: які етичні принципи (конфіденційність, чесність, підзвітність, справедливість) порушуються або ставляться під сумнів? Зверніться до кодексів ACM/IEEE.

4. Можливі дії: запропонуйте 2-3 варіанти дій. Які наслідки (короткострокові та довгострокові) матиме кожен з них?

5. Ваше рішення: який варіант ви обираєте як група? Чому він є найбільш етичним? Як ви його обґрунтуєте керівництву?

Етап 3. Створення презентації (5 хвилин).

Підготуйте 2-3 слайди, які відображають результати вашого аналізу.

Структура презентації:

- *Слайд 1.* назва кейсу та короткий опис ситуації.
- *Слайд 2.* аналіз (зацікавлені сторони, порушені принципи).
- *Слайд 3.* запропоноване рішення та його обґрунтування.

Етап 4. Презентація та загальне обговорення (20 хвилин).

Кожна група презентує своє рішення (2-3 хвилини).

Після презентацій проведіть загальну дискусію:

- Чиї рішення були схожими? Чиї кардинально відрізнялися?
- Який із кейсів виявився найскладнішим для аналізу? Чому?
- Чи може технічне рішення (наприклад, краще шифрування) вирішити етичну проблему? Чи потрібні інші заходи?

Критерії оцінювання (Завдання 1):

Глибина аналізу: чи було розглянуто проблему з різних сторін, чи враховано інтереси різних груп?

Обґрунтованість: чи базується рішення на етичних принципах, а не лише на "так простіше"?

Командна робота: залученість всіх учасників групи до обговорення та презентації.

Якість презентації: чіткість, логічність, дотримання регламенту.

Завдання 2. Дебати «Етика vs. Інновації»

Мета завдання: розвинути навички публічних виступів, аргументації та критичного мислення, а також продемонструвати складність пошуку балансу між технологічним прогресом і моральними цінностями.

Теоретичне підґрунтя

Дебати – це структурований спосіб обговорення спірних питань. Вони вчать не лише висловлювати власну думку, але й уважно слухати опонента, знаходити слабкі місця в його аргументах та будувати контраргументи.

Детальна інструкція виконання

Етап 1. Розподіл ролей та підготовка (10 хвилин).

Об’єднайтесь у пари або невеликі групи по 2-3 особи.

Розподіліть позиції:

Команда А (За інновації): доводить, що технологічний прогрес (наприклад, розвиток штучного інтелекту, збір даних для персоналізації) є настільки важливим для суспільства, що деякі етичні ризики можна виправдати. Головне – це швидкість розвитку та економічна вигода.

Команда Б (За етику): доводить, що етичні принципи (приватність, безпека, справедливість) є абсолютним пріоритетом. Жодна інновація не може бути виправдана, якщо вона завдає шкоди людям або порушує їхні права.

Протягом 10 хвилин кожна команда готує 3-4 сильні аргументи на підтримку своєї позиції, бажано з конкретними прикладами.

Етап 2. Проведення дебатів (10 хвилин).

Встановіть регламент:

- *Виступ команди А:* 2 хвилини.

- *Виступ команди Б:* 2 хвилини.
- *Раунд запитань та відповідей (крос-допит):* 2 хвилини (по 1 хвилині на команду для запитань до опонента).
- *Заключні слова (кожна команда):* по 1 хвилині.

Завдання – не просто виголосити свої аргументи, а й спробувати спростувати аргументи опонентів.

Етап 3. Фіксація підсумків (5 хвилин).

Після завершення дебатів кожна пара/група записує короткий підсумок (5-7 речень), де фіксує:

- Найсильніший аргумент своєї команди.
- Найсильніший аргумент команди-опонента.
- Чи вдалося дійти консенсусу або знайти "золоту середину" під час обговорення?

Етап 4. Загальне обговорення (5 хвилин).

Викладач ініціює дискусію з групою:

- Чи можна знайти баланс між цими двома полюсами? Як він може виглядати на практиці?
- Чи змінив хтось свою думку після дебатів?
- Як впровадження етичних стандартів (наприклад, державне регулювання) може вплинути на темпи інновацій?

Критерії оцінювання (Завдання 2):

Якість аргументації: логічність, переконливість, підкріплення прикладами.

Активність: участь усіх членів команди в підготовці та виступі.

Навички комунікації: чіткість мовлення, вміння слухати опонента, коректність у суперечці.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Дослідження етичних принципів в ІТ

Мета завдання: поглибити знання про конкретний аспект ІТ-етики, розвинути навички роботи з інформаційними джерелами, навчитися доносити складну інформацію у структурованому та візуально привабливому вигляді.

Вступ. Етичні проблеми в ІТ – це не абстрактні філософські питання. Це конкретні виклики, з якими стикаються компанії та фахівці щодня. Ваше завдання – стати експертом з одного з таких викликів.

Детальна інструкція виконання

Крок 1. Вибір теми дослідження.

Оберіть один аспект для глибокого аналізу. Ось розширений список:

Конфіденційність даних (Data Privacy). Як компанії збирають, зберігають та використовують наші дані? Що таке GDPR? Приклади витоків даних (Facebook/Cambridge Analytica, Yahoo!).

Інтелектуальна власність та відкритий код (IP & Open Source). Коли ви можете використовувати чужий код? Що таке ліцензії GPL, MIT, Apache? Чому порушення ліцензій – це неетично?

Прозорість алгоритмів (Algorithmic Transparency). Чому ми маємо право знати, як працює алгоритм, який вирішує, дати нам кредит чи ні? Що таке "чорний ящик" в ШІ?

ШІ та автоматизація робочих місць. Чи етично замінювати людей роботами? Хто несе відповідальність, якщо безпілотний автомобіль потрапляє в аварію?

Етичний хакінг та кібербезпека. Де межа між тестуванням на проникнення та несанкціонованим втручанням? Чи може бути виправданим злам заради "добра"?

Технології та довкілля (Green IT). Етичний аспект споживання енергії дата-центрами, утилізація електронних відходів.

Крок 2. Пошук та аналіз інформації.

Знайдіть мінімум 3 надійні джерела:

- наукові статті (Google Scholar).
- аналітичні звіти (Pew Research Center, Electronic Frontier Foundation).
- кейси відомих компаній (статті у Wired, The Verge, Forbes).
- відеолекції експертів (TED Talks).

Крок 3. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (250-350 слів).

Структура:

1. Вступ. визначте тему та поясніть її актуальність (чому це важливо зараз?).

2. Основна частина: дайте визначення ключовим поняттям. Опишіть 1-2 реальні приклади порушення або етичної проблеми в цій сфері. Проаналізуйте, чому це сталося.

3. Рекомендації: запропонуйте 2-3 конкретні кроки, які можуть допомогти ІТ-фахівцям діяти більш етично в цьому контексті (наприклад,

"використовувати методи анонімізації даних", "проводити аудит алгоритмів на упередженість").

4. Висновок: підсумуйте ключову думку.

Варіант Б: Інфографіка.

Використовуйте Canva, Piktochart або будь-який інший зручний інструмент.

Елементи інфографіки:

- Яскравий заголовок.
- Визначення ключового поняття.
- Статистика або ключові цифри (наприклад, "80% користувачів не читають угоди...").
- Хронологія або схема: "Як відбувається витік даних?".
- Приклад відомого скандалу (логотип компанії + короткий опис).
- "Що робити?" – блок з рекомендаціями у вигляді простих іконок або коротких тез.
- Список джерел дрібним шрифтом.

Крок 4. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2, для гарної роботи – 3-4). Оформлюйте посилання коректно (як гіперпосилання).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте публічне посилання на онлайн-інфографіку/документ у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Актуальність та глибина: тема розкрита не поверхово, є розуміння суті проблеми.

Доказовість: використання конкретних прикладів та посилань на джерела.

Практична цінність: рекомендації є конкретними та реалістичними.

Структурованість та дизайн: чіткість викладу (для тексту) або візуальна привабливість та зрозумілість (для інфографіки).

Самостійність: робота виконана власноруч, а не скопійована з Інтернету.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Чому професійна етика важлива для ІТ-фахівця? Наведіть приклад, коли неетичне рішення програміста може мати серйозні наслідки.

2. Назвіть основні етичні принципи, зафіксовані в кодексах ACM або IEEE.

3. Поясніть різницю між конфіденційністю (privacy) та безпекою (security) даних.

4. Що таке алгоритмічна упередженість? Чому вона виникає і чим небезпечна?

5. Чи може інженер нести моральну відповідальність за те, як використовують створений ним продукт? Аргументуйте.

6. Як ви розумієте термін "прозорість алгоритму"? Чому це важливо для суспільства?

7. Наведіть приклад дилеми "інновації проти етики" з реального життя (окрім розібраних на уроці).

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Офіційні кодекси етики:

1. ACM Code of Ethics and Professional Conduct: https://www.acm.org/code-of-ethics

2. IEEE Code of Ethics: https://www.ieee.org/about/corporate/governance/p7-8.html

Ресурси з етики в технологіях:

1. Markkula Center for Applied Ethics (Santa Clara University): https://www.scu.edu/ethics/focus-areas/technology-ethics/ – Величезна база кейсів, статей та навчальних матеріалів.

2. Electronic Frontier Foundation (EFF): https://www.eff.org/ – Організація, що захищає цифрові права громадян. Багато матеріалів про конфіденційність, свободу слова в інтернеті.

3. AIAAIC (AI, Algorithmic, and Automation Incidents and Controversies): https://www.aiaaic.org/ – Відкрита база даних інцидентів та суперечок, пов’язаних з етикою ШІ.

Корисні статті та Відео

1. TED Playlist: "The ethics of AI": https://www.ted.com/playlists/778/the_ethics_of_ai – Добірка виступів про етичні виклики штучного інтелекту.

2. Wired: "Ethics" section: https://www.wired.com/tag/ethics/ – Свіжі статті про етичні проблеми в технологіях.

3. DOU.ua: статті за тегом "Етика": Аналітика українською мовою про локальні та глобальні етичні дилеми в IT.

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Етика – це не список заборон, а інструмент мислення.** Кодекси етики – це не правила дорожнього руху, де все чітко прописано. Це набір принципів, які допомагають знайти правильне рішення в складній, неоднозначній ситуації.

2. **Ставте під сумнів завдання.** Якщо ви отримали завдання, яке здається вам неетичним (наприклад, збирати дані без згоди), не бійтеся ставити запитання. Ваша професійна відповідальність – не лише писати код, а й розуміти наслідки його роботи.

3. **Етичний скандал може зруйнувати кар'єру.** Пам'ятайте, що IT-спільнота тісна, а інтернет пам'ятає все. Рішення, ухвалене сьогодні заради швидкої вигоди, може наздогнати вас через багато років.

4. **Шукайте однодумців.** Якщо ви зіткнулися з етичною дилемою на роботі, обговоріть її з колегами, менторами, викладачами. Спільне обговорення часто допомагає знайти більш виважене рішення.

ПРАКТИЧНЕ ЗАНЯТТЯ 5
ОГЛЯД СУЧАСНИХ МОВ ПРОГРАМУВАННЯ ТА ОСНОВНИХ ПАРАДИГМ ПРОГРАМУВАННЯ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників цілісного уявлення про різноманітність мов програмування, їхні особливості, сфери застосування та філософські підходи до написання коду, відомі як парадигми програмування.

Основні завдання заняття:

1. Ознайомити з основними сучасними мовами програмування (Python, JavaScript, Java, C++, C#, Rust, Go, Kotlin, Swift) та їхнім позиціонуванням на ринку.

2. Дослідити ключові парадигми програмування: процедурну, об'єктно-орієнтовану (ООП), функціональну, логічну; зрозуміти їхні принципові відмінності.

3. Розвинути первинні навички написання коду через просту практичну вправу мовою Python.

4. Розвинути ключові навички:

- *аналітичне мислення*: порівняння мов за різними критеріями, вибір інструменту під задачу.
- *абстрактне мислення*: розуміння різних підходів до моделювання реальності в коді (парадигми).
- *практичне програмування*: перший досвід роботи зі змінними, введенням/виведенням даних.
- *комунікація*: робота в парі, пояснення написаного коду.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Що таке мова програмування?

Мова програмування – це формальна знакова система, призначена для запису комп'ютерних програм. Вона визначає набір правил (синтаксис) та значень (семантика), які дозволяють програмісту віддавати команди комп'ютеру. Мови програмування є "посередником" між людиною та машиною, яка розуміє лише машинні коди (0 та 1).

2.2. Парадигми програмування: способи мислення

Парадигма програмування – це сукупність ідей, понять і стилів, які визначають підхід до написання програм. Це не просто набір правил мови, а світогляд програміста. Одна й та сама задача може бути вирішена по-різному в різних парадигмах.

Процедурна (імперативна) парадигма

Суть: програма розглядається як послідовність інструкцій (команд), які змінюють стан програми. Це як покроковий рецепт. Код організовується у процедури (функції), які виконують певні дії над даними.

Ключові поняття: змінні, цикли, умовні оператори, функції.

Приклад мов: C, Pascal, BASIC, Python (підтримує).

Об'єктно-орієнтована парадигма (ООП)

Суть: програма будується як сукупність об'єктів, які взаємодіють між собою. Кожен об'єкт – це екземпляр певного класу, який об'єднує дані (властивості) та методи (дії), що можна з цими даними робити. Це моделює реальний світ.

Ключові поняття: клас, об'єкт, інкапсуляція, спадкування, поліморфізм.

Приклад мов: Java, C++, C#, Python, JavaScript, PHP.

Функціональна парадигма

Суть: програма будується як виконання (обчислення) функцій. Ключова ідея – уникати зміни стану та мутації даних. Функції є "чистими" (pure functions), тобто вони завжди повертають однаковий результат для однакових аргументів і не мають побічних ефектів (не змінюють нічого зовні).

Ключові поняття: чисті функції, незмінність даних (immutability), функції вищого порядку, рекурсія.

Приклад мов: Haskell, Clojure, Scala, F#. Багато сучасних мов (JavaScript, Python, Kotlin) також підтримують елементи функціонального програмування.

Логічна парадигма

Суть: програма задається у вигляді набору фактів і правил логіки, а потім ставиться питання (запит). Система самостійно намагається довести істинність запиту, використовуючи логічний висновок.

Ключові поняття: факти, правила, запити, уніфікація, бектрекінг.

Приклад мов: Prolog.

2.3. Класифікація мов програмування

Мови можна класифікувати за різними ознаками:

Рівень абстракції: низькорівневі (Assembler – близькі до машинного коду) та високорівневі (Python, Java – зрозумілі людині).

Спосіб компіляції: компільовані (C, C++ – перетворюються в машинний код до виконання) та інтерпретовані (Python, JavaScript – виконуються рядок за рядком спеціальною програмою-інтерпретатором).

Типізація: статична (тип змінної фіксується і не може змінитись – Java, C++) та динамічна (тип змінної визначається під час виконання і може змінюватись – Python, JavaScript).

Призначення: веб-розробка (JavaScript, PHP), системне програмування (C, Rust), аналіз даних (Python, R), мобільна розробка (Swift для iOS, Kotlin для Android), ігрова індустрія (C++ з Unreal Engine, C# з Unity).

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп'ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постерів).
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - онлайн-редактори коду:
    - Replit: https://replit.com/ – інтерактивне середовище, що підтримує десятки мов, не потребує встановлення.
    - Google Colab: https://colab.research.google.com/ – хмарний блокнот для Python, ідеальний для початківців.
    - Python.org/shell: простий онлайн-інтерпретатор Python.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Порівняння мов програмування

Мета завдання: навчитися аналізувати та порівнювати мови програмування, розуміти їхні сильні та слабкі сторони, а також визначати сфери їх найкращого застосування.

Теоретичне підґрунтя

Вибір мови програмування для проєкту – це важливе інженерне рішення. Воно залежить від багатьох факторів: типу задачі, необхідної продуктивності, наявності бібліотек, досвіду команди. Розуміння особливостей різних мов допомагає зробити правильний вибір.

Детальна інструкція виконання

Етап 1. Формування груп та вибір мов (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть **дві мови програмування** для порівняльного аналізу. Намагайтеся обирати мови з різних "таборів" або з різним призначенням. *Приклади цікавих пар для порівняння:*

- Python vs. JavaScript: (backend/data science vs. frontend/web)
- Java vs. C++: (популярні в корпоративній розробці та іграх/системному програмуванні)
- C# vs. Kotlin: (екосистема Microsoft/.NET vs. сучасна розробка під Android)
- Python vs. Rust: (простота та швидкість розробки vs. швидкість виконання та безпека пам’яті)
- JavaScript (TypeScript) vs. Go: (веб-розробка vs. високопродуктивні мережеві сервіси)

Етап 2. Збір інформації та аналіз (10 хвилин).

Використовуйте наступні ресурси для швидкого пошуку:

- W3Schools (w3schools.com) – короткі туторіали та огляди.
- Programiz (programiz.com) – прості пояснення.
- TIOBE Index (tiobe.com) – індекс популярності мов.
- Stack Overflow Developer Survey – щорічне опитування розробників про улюблені мови, зарплати тощо.

Проаналізуйте мови за наступними критеріями. Заповніть порівняльну таблицю:

Критерій	Мова 1: [назва]	Мова 2: [назва]
Основні парадигми	(наприклад, ООП, процедурна)	(наприклад, функціональна, ООП)
Типізація (статична/динамічна)		
Синтаксис (простий для початківців? зрозумілий?)		

Основна сфера застосування		
Популярність/Спільнота (багато вакансій? бібліотек?)		
Переваги (2-3)		
Недоліки (2-3)		

Етап 3. Створення презентації/постеру (5 хвилин).

Оформіть результати порівняння у вигляді 2-3 слайдів або постера.

Структура:

- *Слайд 1.* Назви мов, логотипи.
- *Слайд 2.* Порівняльна таблиця з ключовими характеристиками.
- *Слайд 3.*

Висновок. Для яких типів проєктів краще підходить Мова 1, а для яких – Мова 2? Яка з мов, на вашу думку, є перспективнішою для вивчення початківцем і чому?

Етап 4. Презентація та загальне обговорення (20 хвилин).

Кожна група презентує свої висновки (2-3 хвилини).

Після всіх презентацій проведіть загальну дискусію:

- Які мови найчастіше називали "найкращими для початківців"? Чому саме Python лідирує в цій номінації?
- Чи існує одна "ідеальна" мова програмування?
- Як змінюється популярність мов з часом? (згадайте Fortran, Cobol, Pascal).

Критерії оцінювання (Завдання 1):

Повнота аналізу: чи охоплено всі запропоновані критерії? Чи є розуміння відмінностей?

Обґрунтованість: висновки про сфери застосування базуються на фактах, а не на припущеннях.

Командна робота: залученість всіх учасників групи.

Якість презентації: чіткість, лаконічність, наочність.

Завдання 2. Вступ до програмування через практичну вправу (Python)

Мета завдання: зробити перший крок у практичному програмуванні, відчути, як працює процедурна парадигма на прикладі простої інтерактивної програми, навчитися працювати в парі (парне програмування).

Теоретичне підґрунтя

Python обрано для цього завдання через його простий та зрозумілий синтаксис, який дозволяє зосередитись на логіці програми, а не на складних правилах мови. Він чудово підходить для ілюстрації процедурної парадигми, де програма виконується крок за кроком зверху вниз.

Детальна інструкція виконання

Етап 1. Ознайомлення з інструментом (3 хвилини).

Об’єднайтесь у пари.

Відкрийте онлайн-редактор Replit (replit.com) або Google Colab (colab.research.google.com).

Створіть новий "Python" проєкт (на Replit) або новий блокнот (на Colab).

Етап 2. Написання коду (12 хвилин).

Ваше завдання – створити програму, яка:

1. Запитує ім’я користувача.
2. Запитує довжину та ширину прямокутника (як числа).
3. Обчислює площу прямокутника.
4. Виводить вітання та результат у гарно відформатованому вигляді.

Покроково:

1. Використайте функцію input() для отримання даних від користувача. input() завжди повертає текст (рядок).

2. Перетворіть отримані рядки на числа, використовуючи float() (для чисел з плаваючою комою) або int() (для цілих чисел).

3. Обчисліть площу: area = length * width.

4. Виведіть результат, використовуючи f-рядок (форматований рядок): print(f"... {name} ... {area}").

Розподіліть ролі: один студент ("водій") пише код, інший ("штурман") спостерігає, аналізує, пропонує ідеї та шукає помилки. Потім можна помінятися ролями.

Етап 3. Тестування та налагодження (5 хвилин).

Запустіть програму (кнопка "Run" на Replit або виконайте клітинку в Colab).

Протестуйте з різними вхідними даними: name = "Олена", length = 5, width = 3 (площа = 15). Що буде, якщо ввести length = 5.5? А якщо ввести текст

замість числа? (програма "впаде" з помилкою – це нормально для першого разу).

Запишіть фінальний, робочий код на аркуші папері або в спільному документі.

Етап 4. Рефлексія та пояснення (5 хвилин).

Дайте відповідь на питання:

- Чи можна сказати, що наша програма виконується "послідовно, крок за кроком"? Чому це називають процедурною парадигмою?
- Як би ви змінили програму, щоб вона обчислювала не площу, а об'єм кімнати (довжина * ширина * висота)?

За бажанням, презентуйте свій код групі, поясніть, як він працює.

Приклад коду з коментарями:

```
# Запитуємо ім'я користувача і зберігаємо його в змінній name
name = input("Будь ласка, введіть ваше ім'я: ")

# Запитуємо довжину. Отриманий текст перетворюємо на число з плаваючою комою (float)
length = float(input("Введіть довжину прямокутника (в см): "))

# Аналогічно для ширини
width = float(input("Введіть ширину прямокутника (в см): "))

# Обчислюємо площу
area = length * width

# Виводимо результат, використовуючи f-рядок для підстановки значень змінних
print(f"Вітаю, {name}! Площа прямокутника дорівнює {area} квадратних сантиметрів.")
```

Критерії оцінювання (Завдання 2):

Працездатність: код запускається та виконує поставлене завдання без синтаксичних помилок.

Коректність: програма правильно обчислює площу для різних чисел.

Розуміння: студенти можуть пояснити, що робить кожен рядок їхнього коду.

Співпраця: активна робота в парі, зміна ролей.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Дослідження парадигм програмування

Мета завдання: поглибити знання про одну з парадигм програмування, навчитися знаходити приклади її реалізації в коді та аналізувати її прикладне значення.

Вступ. За кожною мовою програмування стоїть певна філософія. Розуміння парадигм допомагає писати більш ефективний, зрозумілий і підтримуваний код. Ваше завдання – стати експертом з однієї з них.

Детальна інструкція виконання

Крок 1. Вибір парадигми.

Оберіть **одну** парадигму програмування для глибокого дослідження:

- процедурна (procedural)
- об’єктно-орієнтована (object-oriented)
- функціональна (functional)
- логічна (logic)

Крок 2. Дослідження.

Знайдіть відповіді на наступні питання, використовуючи мінімум 2-3 надійні джерела:

1. Визначення та історія. Коли і чому виникла ця парадигма? У чому її основна ідея?

2. Ключові концепції. Які головні поняття лежать в її основі? (наприклад, для ООП – класи, об’єкти, інкапсуляція, спадкування, поліморфізм).

3. Переваги. Для вирішення яких задач ця парадигма підходить найкраще? Чому?

4. Недоліки / обмеження. У яких випадках її використання може бути ускладненим або неефективним?

5. Мови програмування. Назвіть щонайменше 3 мови, які підтримують цю парадигму (або створені спеціально для неї).

Крок 3. Приклад коду.

Напишіть невеликий приклад коду (5-15 рядків), який наочно демонструє ключову ідею обраної парадигми.

Для процедурної: напишіть функцію, яка щось обчислює, і викличте її.

Для ООП: створіть простий клас (наприклад, Car або Student) з кількома властивостями та методом, а потім створіть об’єкт цього класу.

Для функціональної: напишіть "чисту функцію", яка не змінює зовнішніх змінних, і покажіть приклад її використання (можна на Python, використовуючи map або filter).

Для логічної: наведіть приклад на Prolog (факти та правило) або, якщо це складно, знайдіть готовий приклад в інтернеті та поясніть його.

Обов’язково додайте коментарі до коду, які пояснюють, як саме він ілюструє принципи парадигми.

Крок 4. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (250-350 слів).

Структура:

1. Вступ. Назва парадигми, коротке визначення.

2. Основна частина: відповіді на питання з Кроку 2 (концепції, переваги, недоліки, мови).

3. Приклад коду: вставте код з коментарями та поясніть, що він робить і як демонструє парадигму.

4. Висновок: де ця парадигма застосовується сьогодні?

Варіант Б: Ментальна карта.

У центрі – назва парадигми.

Перший рівень гілок: "Ключові концепції", "Переваги", "Недоліки", "Мови", "Приклад коду".

На гілці "Приклад коду" зробіть вставку з невеликим фрагментом коду або посиланням на нього.

Крок 5. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2). Оформлюйте посилання як гіперпосилання.

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте публічне посилання на онлайн-документ/ментальну карту у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина розуміння: робота демонструє не поверхове, а усвідомлене розуміння парадигми.

Коректність прикладу: код відповідає заявленій парадигмі та ілюструє її правильно.

Структурованість: логічність викладу, наявність всіх необхідних елементів.

Доказовість: використання авторитетних джерел, коректність посилань.

Оформлення: чистота, відсутність помилок.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Що таке парадигма програмування? Чому важливо знати різні парадигми?
2. Опишіть основну ідею процедурного програмування. Наведіть приклад.
3. Які головні принципи об’єктно-орієнтованого програмування? Наведіть приклад з реального життя, який можна описати за допомогою класу та об’єкта.
4. У чому ключова відмінність функціонального програмування від імперативного (процедурного)?
5. Чому Python часто рекомендують як першу мову для вивчення? Які парадигми він підтримує?
6. Для яких завдань найкраще підходить мова C++? А JavaScript?
7. Що таке статична та динамічна типізація? Наведіть приклади мов для кожного типу.
8. Чи можна в одній мові програмування використовувати різні парадигми? Наведіть приклад.

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Базові ресурси для вивчення мов та парадигм:

1. Programiz: Programming Paradigms: https://www.programiz.com/learn/programming-paradigms – чудовий огляд основних парадигм для початківців.
2. W3Schools: https://www.w3schools.com – прості туторіали з багатьох мов (Python, Java, C++, JavaScript).
3. freeCodeCamp: https://www.freecodecamp.org – величезна кількість безкоштовних статей та інтерактивних курсів з програмування, включаючи матеріали про парадигми.

Поглиблене вивчення:

4. Python.org – Official Tutorial: https://docs.python.org/3/tutorial/ – офіційний туторіал мовою Python.

5. Structure and Interpretation of Computer Programs (SICP): Класична книга, яка глибоко розкриває різні парадигми програмування (доступна безкоштовно онлайн).

6. Роберт Мартін (Uncle Bob). "Чиста архітектура". Книга про принципи проєктування, які базуються на розумінні парадигм.

Інструменти:

7. Replit: https://replit.com – онлайн-редактор та хостинг коду.

8. Google Colab: https://colab.research.google.com – хмарні блокноти Jupyter для Python.

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Не намагайтеся вивчити все й одразу.** Світ мов програмування величезний. Ваше завдання зараз – зрозуміти загальну карту та обрати мову для глибшого вивчення.

2. **Мова – це інструмент, а не мета.** Вивчаючи мову, ви насправді вчитеся мислити алгоритмічно. Парадигми – це різні способи такого мислення. Знання кількох парадигм робить вас гнучким фахівцем.

3. **Пишіть код якомога більше.** Теорія без практики мертва. Навіть прості вправи, як-от на уроці, допомагають "набити руку" та зрозуміти логіку роботи програм.

4. **Вивчайте код інших.** Заходьте на GitHub, дивіться, як пишуть професіонали. Аналізуйте, які парадигми та стилі вони використовують.

ПРАКТИЧНЕ ЗАНЯТТЯ 6
МЕТОДОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (AGILE, SCRUM, KANBAN ТОЩО)

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників розуміння сучасних підходів до організації роботи в ІТ-командах, принципів гнучкої розробки програмного забезпечення та практичних навичок використання інструментів управління проєктами.

Основні завдання заняття:

1. Ознайомити з основними методологіями розробки ПЗ: каскадною (Waterfall), гнучкою (Agile) та її найпопулярнішими фреймворками – Scrum і Kanban.

2. Дослідити ключові ролі, артефакти та події в Scrum, принципи візуалізації роботи в Kanban.

3. Розвинути практичні навички планування роботи в команді, використання інструменту Trello для управління завданнями.

4. Розвинути ключові навички:

– *командна робота*: розподіл ролей, координація дій, колективне прийняття рішень.

– *планування та тайм-менеджмент*: декомпозиція задач, оцінка трудомісткості, визначення пріоритетів.

– *аналітичне мислення*: порівняння різних методологій, вибір оптимальної для конкретного проєкту.

– *комунікація*: презентація результатів роботи команди, участь в обговоренні.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Що таке методологія розробки ПЗ?

Методологія розробки програмного забезпечення – це набір практик, правил, принципів і процедур, які використовує команда для планування, виконання та контролю процесу створення програмного продукту. Це своєрідний "регламент", який допомагає зробити роботу ефективною, передбачуваною та керованою.

2.2. Класичні та гнучкі методології

Каскадна модель (Waterfall)

Суть: проєкт виконується як послідовність етапів (вимоги -> дизайн -> реалізація -> тестування -> супровід). Кожен наступний етап починається тільки після повного завершення попереднього.

Коли застосовується: для невеликих проєктів з чітко визначеними та незмінними вимогами (наприклад, державні замовлення, оборонна промисловість).

Недоліки: дуже негнучка. Зміна вимог на пізньому етапі коштує дуже дорого.

Agile (Гнучка методологія)

Суть: це не одна методологія, а ціла філософія, набір цінностей та принципів, зафіксованих у Agile Manifesto (Маніфесті гнучкої розробки ПЗ). Ключові цінності:

- Люди та взаємодія важливіші за процеси та інструменти.
- Працюючий продукт важливіший за вичерпну документацію.
- Співпраця з замовником важливіша за узгодження умов контракту.
- Готовність до змін важливіша за дотримання початкового плану.

Ключовий принцип: розробка ведеться ітераційно (короткими циклами – спринтами), в кінці кожного з яких команда отримує працюючий приріст продукту.

2.3. Фреймворки Agile

- **Scrum:**

Суть: чітко структурований фреймворк для роботи команди. Робота організована у фіксовані за часом ітерації – **спринти** (зазвичай 1-4 тижні).

Ролі:

Власник продукту (Product Owner): відповідає за цінність продукту. Формує та пріоритезує завдання в беклозі продукту.

Scrum-майстер (Scrum Master): допомагає команді дотримуватися правил Scrum, усуває перешкоди, фасилітує зустрічі.

Команда розробки (Development Team): самокерована, крос-функціональна група (3-9 осіб), яка безпосередньо створює продукт.

Артефакти:

Беклог продукту (Product Backlog): упорядкований список всього, що може знадобитися в продукті.

Беклог спринту (Sprint Backlog): набір завдань, вибраних з беклогу продукту для виконання в поточному спринті.

Інкремент (Increment): працююча версія продукту, створена за спринт.

Події: планування спринту, щоденний Scrum (15-хвилинна зустріч), огляд спринту, ретроспектива спринту.

- **Kanban:**

Суть: методологія, яка фокусується на візуалізації потоку роботи та обмеженні кількості задач, що виконуються одночасно (Work In Progress, WIP). Мета – зробити процес прозорим і виявити "вузькі місця".

Інструмент: Kanban-дошка з колонками (наприклад, "To Do", "In Progress", "Testing", "Done"). Кожне завдання – це картка, яка рухається по дошці.

Ключові принципи:

- візуалізуйте роботу.
- обмежуйте кількість одночасної роботи (WIP limits).
- керуйте потоком.
- робіть правила процесу явними.
- впроваджуйте зворотній зв’язок.
- покращуйте спільно (еволюційно).

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, стікери (якщо є можливість) для симуляції фізичної Kanban-дошки.
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - онлайн-дошки для спільної роботи:
    - Miro: https://miro.com/ – ідеально для симуляції Scrum-подій та створення беклогу.
  - інструменти управління проєктами:
    - Trello: https://trello.com/ – найпростіший інструмент для створення Kanban-дошок.
    - Notion, Asana, Jira – більш складні альтернативи.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Симуляція спринту в Scrum

Мета завдання: на власному досвіді відчути, як працює Scrum-команда: розподіл ролей, планування роботи, створення беклогу та планування спринту.

Теоретичне підґрунтя

Scrum – це не просто теорія, а живий процес. Сьогодні ви станете частиною цього процесу і зрозумієте, як команда з різними ролями взаємодіє для досягнення спільної мети.

Детальна інструкція виконання

Етап 1. Формування груп та визначення проєкту (5 хвилин).

Об’єднайтесь у групи по 4-5 осіб.

Оберіть уявний проєкт. *Приклади:*

- Мобільний додаток для відстеження звичок (habit tracker).
- Веб-сайт для студентського клубу.
- Простий калькулятор калорій.
- Організація онлайн-конференції.

Визначте, хто буде виконувати ролі (можна суміщати, але бажано, щоб ролі були різні):

- Власник продукту (Product Owner): відповідає за те, ЩО ми робимо. Визначає пріоритети.
- Scrum-майстер (Scrum Master): відповідає за те, ЯК ми працюємо. Слідкує за часом, допомагає дотримуватись правил.
- Команда розробки (2-3 особи): відповідає за те, як реалізувати завдання. Оцінює трудомісткість.

Етап 2. Створення беклогу продукту (10 хвилин).

Вся команда (на чолі з Власником продукту) методом мозкового штурму створює беклог продукту – список всього, що потрібно зробити для реалізації проєкту.

Запишіть 5-7 завдань (user stories) на аркуші паперу, стікерах або в онлайн-дошці Miro. *Приклад беклогу для "Трекера звичок":*

1. Як користувач, я хочу створити обліковий запис, щоб зберігати свої дані.
2. Як користувач, я хочу додати нову звичку (наприклад, "пити воду").

3. Як користувач, я хочу відзначати виконання звички на сьогодні.

4. Як користувач, я хочу бачити статистику виконання за тиждень у вигляді графіка.

5. Як користувач, я хочу отримувати нагадування про звичку.

6. (Технічне) Налаштувати базу даних для зберігання інформації.

Етап 3. Планування спринту (5 хвилин).

Уявіть, що тривалість спринту – 1 тиждень.

Команда (разом з Власником продукту) обирає з беклогу 2-4 найважливіших завдання, які можна реально виконати за цей тиждень. Вони стають беклогом спринту.

Scrum-майстер допомагає команді оцінити, чи не забагато вони взяли.

Для кожного завдання коротко обговоріть, як його виконувати. Можна навіть розбити велике завдання на підзавдання (наприклад, "Додати нову звичку" -> "Створити форму на сайті", "Написати код для збереження в БД").

Етап 4. Створення презентації (5 хвилин).

Підготуйте 2-3 слайди, які відображають результати вашої роботи.

Структура презентації:

- *Слайд 1.* Назва проєкту, склад команди та їхні ролі.
- *Слайд 2.* Беклог продукту (повний список з 5-7 завдань).
- *Слайд 3.* Беклог спринту (завдання на 1-й спринт) та короткий план їх реалізації.

Етап 5. Презентація та обговорення (15 хвилин).

Кожна група презентує результати свого планування (2-3 хвилини).

Загальна дискусія:

- Які труднощі виникли під час визначення пріоритетів? Чи легко було обрати завдання на спринт?
- Чи були конфлікти між ролями (наприклад, Власник продукту хотів більше, ніж Команда могла зробити)? Як їх вирішували?
- Для чого потрібна роль Scrum-майстра? Що б відбувалося без нього?

Критерії оцінювання (Завдання 1):

Дотримання структури: наявність всіх ролей, беклогу продукту, беклогу спринту.

Реалістичність: завдання відповідають обраній темі проєкту, план спринту виглядає здійсненним.

Командна робота: розподіл ролей, активна участь всіх членів групи.

Якість презентації: чіткість, лаконічність, зрозумілість.

Завдання 2. Створення Kanban-дошки в Trello

Мета завдання: опанувати базовий інструмент управління проєктами (Trello) та навчитися застосовувати принципи Kanban для візуалізації робочого процесу.

Теоретичне підґрунтя

Kanban-дошка – це "серце" методу Kanban. Вона робить роботу прозорою для всієї команди. Trello – це цифрова реалізація такої дошки, яка дозволяє легко створювати картки, призначати виконавців, встановлювати дедлайни та пріоритети.

Детальна інструкція виконання

Етап 1. Створення дошки (3 хвилини).

Об’єднайтесь у пари.

Зареєструйтесь на сайті trello.com (можна використати акаунт Google).

Створіть нову дошку. Назвіть її відповідно до уявного проєкту, наприклад: "Організація студентської конференції" або "Розробка лендінгу для стартапу".

Етап 2. Налаштування колонок (3 хвилини).

За замовчуванням Trello створює три колонки: "To Do", "Doing", "Done". Це базова Kanban-дошка. Ви можете залишити їх або додати/перейменувати відповідно до вашого процесу.

Приклад розширених колонок для конференції:

- Ідеї / Беклог (все, що треба зробити)
- В роботі / In Progress (що робиться зараз)
- На перевірці / Review (завдання, які потребують підтвердження)
- Виконано / Done

Етап 3. Створення карток (10 хвилин).

Створіть щонайменше **6-8 карток** із завданнями.

Приклад карток для "Конференції":

- Знайти та забронювати приміщення.
- Розробити дизайн постера та програми.
- Створити форму реєстрації для учасників.
- Запросити спікерів (написати листи).

- Придбати каву та снеки для перерви.
- Підготувати презентацію для відкриття.

Для кожної картки виконайте наступне (клацніть на картку, щоб відкрити меню):

- Додайте опис (деталі завдання).
- Призначте виконавця (учасника пари) – функція "Members".
- Встановіть пріоритет, додавши кольорову мітку (Labels). Створіть мітки: "Високий", "Середній", "Низький".
- За бажанням, додайте термін виконання (Due date).

Етап 4. Симуляція процесу (4 хвилини).

Перемістіть 2-3 картки з колонки "To Do" (або "Ідеї") до колонки "In Progress" – це означає, що команда почала над ними працювати.

Перемістіть 1 картку до колонки "Done" – це завдання вже виконано.

Таким чином ви зробили дошку "живою".

Етап 5. Презентація та рефлексія (10 хвилин).

Зробіть **скріншот** вашої дошки.

Запишіть коротке пояснення (5-7 речень) у текстовому документі, даючи відповіді на питання:

- Який проєкт ви обрали?
- Як Kanban-дошка допомагає побачити "вузькі місця"? (Наприклад, якщо в колонці "In Progress" накопичилося 5 завдань – це означає, що команда перевантажена).
- Як призначення виконавців та пріоритетів допомагає в роботі?

За бажанням, декілька пар презентують свої дошки групі, демонструючи екран.

Критерії оцінювання (Завдання 2):

Повнота дошки: наявність мінімум 3 колонок та 6-8 карток.

Функціональність: використання можливостей Trello (опис, виконавець, мітки, терміни).

Розуміння Kanban: коректне розміщення карток за статусами, логічне пояснення принципів роботи дошки.

Співпраця: активна робота обох членів пари.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Аналіз методології розробки ПЗ

Мета завдання: поглибити знання про одну з методологій розробки, навчитися знаходити та аналізувати інформацію, а також застосувати отримані знання для створення практичного інструменту (Trello-дошки).

Вступ. Кожна методологія має свої сильні та слабкі сторони. Вибір методології залежить від типу проєкту, складу команди та вимог замовника. Ваше завдання – розібратися в одній з них настільки глибоко, щоб змогти пояснити її переваги та недоліки.

Детальна інструкція виконання

Крок 1. Вибір методології.

Оберіть **одну** методологію (або фреймворк) для дослідження:

- Waterfall (Каскадна модель)
- Agile (як філософія)
- Scrum
- Kanban
- Lean (Ощадлива розробка)
- Extreme Programming (XP)

Крок 2. Дослідження.

Знайдіть відповіді на наступні питання, використовуючи мінімум 2-3 надійні джерела (Scrum Guide, сайти Atlassian, статті на DOU, книги):

1. Історія та основна ідея: Коли і чому виникла ця методологія? Яку проблему вона вирішує?
2. Ключові принципи: Назвіть 3-5 основних принципів, на яких вона базується.
3. Ролі, артефакти, події (якщо є): Для Scrum: хто бере участь і що створюється? Для Kanban: як організований процес?
4. Переваги та недоліки: У яких проєктах цю методологію застосовувати найкраще, а в яких – варто уникати?
5. Приклад використання: Знайдіть інформацію (або придумайте), яка відома компанія використовує цю методологію. (Наприклад, Spotify відома своєю адаптацією Agile).

Крок 3. Створення практичної частини (Trello-дошки).

Створіть у Trello нову дошку для уявного проєкту, який, на вашу думку, найкраще підходить для обраної методології.

Якщо ви обрали Scrum, дошка може відображати беклог спринту (колонки: "Беклог спринту", "In Progress", "To Verify", "Done").

Якщо ви обрали Kanban, дошка повинна мати чіткі колонки, що відображають етапи потоку робіт, і бажано встановити обмеження WIP (можна просто вказати їх в описі колонки).

Додайте **5-7 карток** із завданнями, які відповідають обраній темі.

Налаштуйте мітки пріоритетів, призначте виконавців (вигаданих).

Зробіть **скріншот** дошки або налаштуйте публічний доступ (Share -> "Make board public" -> скопіюйте посилання).

Крок 4. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (300-400 слів).

Структура:

1. Вступ. Назва методології, її місце серед інших.

2. Основна частина: відповіді на питання з Кроку 2 (принципи, ролі, переваги, недоліки, приклад).

3. Практична частина: опишіть, який проєкт ви обрали для Trello-дошки і чому він підходить під обрану методологію.

4. Висновок: ваші власні враження. Чи хотіли б ви працювати за цією методологією?

Варіант Б: Ментальна карта.

У центрі – назва методології.

Перший рівень гілок: "Принципи", "Ролі", "Артефакти", "Переваги", "Недоліки", "Приклади".

Додайте гілку "Мій проєкт", де вставте скріншот Trello-дошки або посилання на неї.

Крок 5. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2). Оформлюйте посилання як гіперпосилання.

Завантажте готовий файл (PDF, DOCX, JPG, PNG) разом зі скріншотом дошки (або вставте посилання на дошку в документ) у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина аналізу: робота демонструє розуміння суті методології, а не просто переказ вікіпедії.

Практичне застосування: Trello-дошка відповідає обраній методології, завдання логічні, використано базовий функціонал.

Структурованість: логічність викладу, наявність всіх необхідних елементів.

Доказовість: використання авторитетних джерел, коректність посилань.

Оформлення: чистота, відсутність помилок.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. У чому полягає основна відмінність між каскадною (Waterfall) та гнучкою (Agile) моделями розробки?
2. Назвіть чотири ключові цінності Agile Manifesto.
3. Опишіть три основні ролі в Scrum. Хто за що відповідає?
4. Що таке "беклог продукту" (Product Backlog) та "беклог спринту" (Sprint Backlog)? У чому різниця?
5. Назвіть основні події (церемонії) в Scrum. Яка мета щоденної зустрічі (Daily Scrum)?
6. Які ключові принципи методу Kanban? Що означає обмеження WIP (Work In Progress)?
7. Для якого типу проєктів краще підійде Scrum, а для якого – Kanban? Чому?
8. Як інструменти на кшталт Trello допомагають впроваджувати Agile-методології?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Офіційні джерела та класика:

1. Agile Manifesto (Маніфест гнучкої розробки ПЗ): http://agilemanifesto.org/ (доступний багатьма мовами, включаючи українську).
2. The Scrum Guide (українською): https://scrumguides.org/ (шукайте версію Ukrainian) – обов’язкове чтиво для розуміння Scrum.

Практичні ресурси та статті:

3. Atlassian Agile Coach: https://www.atlassian.com/agile – величезна база знань про Agile, Scrum, Kanban від творців Jira. Дуже рекомендовано.

4. DOU.ua: статті за тегом "Управління проектами": Багато практичних матеріалів від українських фахівців.

5. Trello Guide: https://trello.com/guide – офіційний посібник з використання Trello.

Книги

6. Кен Швабер, Майк Бідл. "Agile-лідери: Як впровадити Scrum та змінити культуру компанії".

7. Девід Андерсон. "Kanban: Успішна еволюційна зміна для технологічних бізнесів".

8. Майк Кон. "Agile-оцінювання та планування проєктів".

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Методологія – не догма, а інструмент.** Не існує "ідеальної" методології. Досвідчені команди часто комбінують елементи Scrum, Kanban та інших підходів (це називається Scrumban). Головне – щоб правила працювали на результат, а не заважали йому.

2. **Trello – це лише початок.** В реальних IT-компаніях часто використовують більш потужні інструменти: **Jira** (для складних процесів), **Asana, Monday.com, Notion**. Але принципи, закладені в Trello (дошка, картки, колонки), є основою для всіх них.

3. **Ретроспектива – найважливіша зустріч.** В Scrum є подія "Ретроспектива спринту", де команда обговорює, що було добре, а що треба покращити в їхній роботі. Це ключ до постійного вдосконалення. Беріть цей принцип у своє студентське життя: аналізуйте, як ви вчилися, і шукайте способи робити це ефективніше.

4. **Комунікація понад усе.** Навіть найкраща методологія не врятує проєкт, якщо в команді немає довіри та відкритого спілкування. Scrum-майстер – це не "начальник", а слуга-лідер, який допомагає команді спілкуватися ефективно.

ПРАКТИЧНЕ ЗАНЯТТЯ 7
ОСНОВИ ПРОЄКТУВАННЯ ТА АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників базових уявлень про проєктування програмного забезпечення, розуміння важливості модульності, чіткого розподілу функцій між компонентами та основ створення простих програмних систем.

Основні завдання заняття:

1. Ознайомити з поняттями архітектури програмного забезпечення, модульності, зв'язності (cohesion) та взаємодії модулів на прикладі простих програм.
2. Дослідити як реальні програмні продукти (калькулятори, редактори, вебсайти) можна розділити на окремі функціональні блоки.
3. Розвинути навички створення простих діаграм модулів та написання фрагментів коду, що реалізують функціональність окремого модуля.
4. Розвинути ключові навички:
   - *аналітичне мислення*: декомпозиція складного цілого на прості частини.
   - *проєктне мислення*: планування структури програми до написання коду.
   - *командна робота*: спільне обговорення та створення діаграм.
   - *комунікація*: представлення результатів аналізу.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Що таке архітектура програмного забезпечення?

Архітектура програмного забезпечення – це загальна структура програми, набір її компонентів (модулів) та зв'язків між ними. Це як «скелет» або «план будівлі», який визначає, як різні частини програми будуть взаємодіяти. Добре спроєктована архітектура робить програму зрозумілою, гнучкою до змін і легкою в підтримці.

2.2. Модульність – ключовий принцип проєктування

Модульність – це принцип, згідно з яким програма розбивається на окремі, відносно незалежні частини – модулі. Кожен модуль відповідає за чітко визначену функцію.

Переваги модульності:

- Простота розуміння: можна вивчати програму по частинах, не намагаючись осягнути все одразу.
- Легкість розробки: різні модулі можуть створювати різні програмісти одночасно.
- Зручність тестування: кожен модуль можна перевірити окремо.
- Полегшення внесення змін: якщо потрібно змінити одну функцію, найчастіше достатньо змінити лише один модуль, не чіпаючи інші.
- Повторне використання: вдало написаний модуль можна використати в іншій програмі.

2.3. Зв'язність (Cohesion)

Зв'язність – це міра того, наскільки тісно пов'язані елементи всередині одного модуля. В ідеалі модуль повинен мати високу зв'язність, тобто всі його частини працюють над однією спільною задачею. Наприклад, модуль «Обчислення податків» не повинен займатися виведенням даних на екран – це знижує зв'язність.

2.4. Взаємодія модулів

Модулі не існують ізольовано – вони обмінюються даними. Цей обмін має бути чітко визначеним. Зазвичай один модуль викликає функції іншого або передає йому дані через вхідні параметри. Добре спроєктована система має слабку зв'язаність (low coupling) між модулями, тобто вони залежать один від одного мінімально. Це дозволяє змінювати один модуль, не ламаючи інші.

2.5. Приклад: структура простого калькулятора

Уявімо простий калькулятор. Його можна розділити на такі модулі:

1. Модуль введення: отримує числа та операцію від користувача (з клавіатури або кнопок).
2. Модуль обчислень: отримує числа та операцію, виконує додавання, віднімання тощо, повертає результат.
3. Модуль виведення: отримує результат і показує його на екрані.
4. Модуль історії: (додатково) зберігає попередні обчислення.

Ці модулі взаємодіють: модуль введення передає дані модулю обчислень, той передає результат модулю виведення, а також, можливо, модулю історії.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, олівці/фломастери (для малювання діаграм).
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - онлайн-інструменти для створення діаграм:
    - Draw.io (app.diagrams.net): безкоштовний, простий, інтегрується з Google Drive.
    - Miro: зручна онлайн-дошка для спільної роботи.
    - Lucidchart: має безкоштовний тариф з обмеженнями.
  - онлайн-редактори коду Python:
    - Replit: https://replit.com/
    - Google Colab: https://colab.research.google.com/
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Аналіз модульності в реальному проєкті

Мета завдання: навчитися розпізнавати модулі в існуючих програмних продуктах і розуміти, як поділ на частини полегшує їх створення та використання.

Теоретичне підґрунтя

Навіть дуже складні програми, як-от Google Docs або Microsoft Word, побудовані з багатьох модулів. Сьогодні ми спробуємо «розібрати» простий продукт на уявні модулі, щоб зрозуміти логіку його внутрішньої будови.

Детальна інструкція виконання

Етап 1. Формування груп та вибір продукту (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть простий програмний продукт для аналізу. *Приклади:*

- Калькулятор (на смартфоні або в Windows).
- Текстовий редактор (наприклад, Блокнот або простий редактор на кшталт Notepad++).
- Простий вебсайт (сайт-візитка, блог – уявіть його структуру).
- Мобільний додаток "Нотатки" (стандартний на смартфоні).
- Список справ (To-Do List).

Якщо є доступ до інтернету, можна швидко переглянути опис або скріншоти продукту, щоб краще зрозуміти його функції.

Етап 2. Визначення модулів (10 хвилин).

Методом мозкового штурму визначте, з яких логічних частин складається обраний продукт. Пам’ятайте: кожен модуль відповідає за одну чітку функцію.

Виділіть **3-5 модулів**. Для кожного модуля заповніть таблицю:

Назва модуля	**Основна функція (що робить?)**	**Вхідні дані (що отримує?)**	**Вихідні дані (що повертає?)**
Приклад: Калькулятор			
Введення	Отримує числа та операцію від користувача	Натискання кнопок	Числа та символ операції
Обчислення	Виконує арифметичну операцію	Числа та операція	Результат (число)
Виведення	Показує результат на екрані	Результат	Візуальне відображення

Етап 3. Опис взаємодії модулів (5 хвилин).

Продумайте та опишіть, в якому порядку модулі обмінюються даними. Намалюйте просту схему (діаграму) на папері або в документі. Використовуйте прямокутники для модулів і стрілки для передачі даних.

Приклад для калькулятора: введення (числа, операція) -> обчислення -> виведення (результат). Також можна додати модуль "Історія", який отримує результат від "Обчислення" і зберігає його.

Етап 4. Створення презентації (5 хвилин).

Оформіть результати у вигляді 2-3 слайдів або постера.

Структура:

- *Слайд 1.* Назва продукту, короткий опис його призначення.
- *Слайд 2.* Таблиця з модулями та їх функціями.

▪ *Слайд 3.* Схема взаємодії модулів (діаграма) та висновок про те, як модульність допомагає в розробці цього продукту.

Етап 5. Презентація та обговорення (15 хвилин).

Кожна група презентує свої результати (2-3 хвилини).

Загальна дискусія:

▪ Чи легко було виділити модулі? Чи виникали суперечки щодо того, що має входити в модуль, а що ні?

▪ Як зміниться структура програми, якщо додати нову функцію? (наприклад, до калькулятора – обчислення відсотків). Чи достатньо буде додати новий модуль?

▪ Чому, на вашу думку, розробники починають з проєктування модулів, а не одразу з написання коду?

Критерії оцінювання (Завдання 1):

Обґрунтованість виділення модулів: кожен модуль має чітку, непересічну функцію.

Повнота: виділено достатню кількість модулів для опису основної логіки програми.

Чіткість схеми: схема взаємодії зрозуміла, стрілки вказують напрямок передачі даних.

Командна робота: участь усіх членів групи.

Якість презентації.

Завдання 2. Практична вправа «Проєктування простої програми»

Мета завдання: спробувати себе в ролі архітектора: самостійно спроєктувати структуру невеликої програми, створити діаграму модулів і реалізувати один із модулів у коді.

Теоретичне підґрунтя

Проєктування передує написанню коду. Створюючи діаграму, ми відповідаємо на питання: «З чого складатиметься програма?» та «Як частини спілкуватимуться?». Потім ми беремо один модуль і реалізуємо його, перевіряючи, чи наша ідея працює на практиці.

Детальна інструкція виконання

Етап 1. Вибір ідеї та створення діаграми (10 хвилин).

Об’єднайтесь у пари.

Оберіть ідею для простої програми. *Приклади:*

- Список справ (To-Do List): функції – додати завдання, видалити завдання, показати всі завдання.
- Калькулятор калорій: користувач вводить назву продукту та калорії, програма додає до щоденного підсумку.
- Опитувальник: програма ставить кілька запитань і збирає відповіді.

На аркуші паперу або в онлайн-інструменті (Draw.io, Miro) створіть діаграму модулів. Вона має містити:

- 3-4 прямокутники (модулі) з назвами.
- Стрілки, що показують, які модулі передають дані іншим.

Приклад для "Списку справ":

- Модуль інтерфейсу: отримує команди від користувача (додати, видалити, показати) та дані (текст завдання).
- Модуль управління завданнями: отримує команди та дані від інтерфейсу, виконує логіку (додає в список, видаляє).
- Модуль зберігання: зберігає список завдань (у простому випадку – просто в змінній).
- Модуль виведення: отримує дані (список завдань) і показує їх користувачеві.

Схема: інтерфейс -> управління -> зберігання (туди-сюди), і зберігання -> виведення (за запитом).

Етап 2. Написання коду для одного модуля (10 хвилин).

Виберіть **один модуль** з вашої діаграми, який здається вам найпростішим для реалізації.

Відкрийте Replit або Google Colab.

Напишіть код на Python (5-10 рядків), який реалізує основну функцію цього модуля. Не обов’язково робити повноцінну програму – достатньо показати, як би цей модуль працював окремо.

Приклад коду для модуля "Управління завданнями" (без зберігання):

```
# Модуль управління завданнями (імітація)
# Просто демонструє, як функція додає завдання до списку
def add_task(tasks_list, new_task):
    """Додає нове завдання до списку."""
```

```
    tasks_list.append(new_task)
    print(f"Завдання ‘{new_task}’ додано.")
    return tasks_list

def show_tasks(tasks_list):
    """Виводить всі завдання."""
    if not tasks_list:
        print("Список справ порожній.")
    else:
        print("Ваші справи:")
        for i, task in enumerate(tasks_list, 1):
            print(f"{i}. {task}")

# Невеличкий тест
my_tasks = []
add_task(my_tasks, "Купити молоко")
add_task(my_tasks, "Зробити зарядку")
show_tasks(my_tasks)
```

Зверніть увагу: цей код не має повноцінного інтерфейсу, але демонструє логіку модуля.

Етап 3. Фіксація результатів (5 хвилин).

Запишіть діаграму (фотографуйте, якщо на папері, або збережіть файл з онлайн-інструменту) та код у текстовий документ (Google Docs).

Підготуйте коротке усне пояснення (1 хвилина), де ви розкажете:

- Яку програму ви проєктували.
- Які модулі виділили.
- Який модуль реалізували в коді та як він працює.

Етап 4. Презентація та обговорення (5 хвилин).

За бажанням, 2-3 пари презентують свої напрацювання.

Загальне обговорення:

- Чи допомогла діаграма краще зрозуміти, що саме треба програмувати?
- Якби ви писали всю програму далі, чи легко було б додавати нові модулі? Чому?

Критерії оцінювання (Завдання 2):

Наявність діаграми: чітко виділено модулі, підписано їх, показано зв’язки.

Працездатність коду: код написано без синтаксичних помилок, він виконує заявлену функцію.

Відповідність коду модулю: код реалізує саме той модуль, який було заявлено.

Розуміння: студенти можуть пояснити логіку діаграми та коду.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Аналіз архітектури простого додатку

Мета завдання: закріпити навички декомпозиції програмного забезпечення, навчитися документувати архітектуру у вигляді звіту або ментальної карти та підкріплювати теорію невеликим практичним прикладом коду.

Вступ. Кожен програмний продукт, навіть дуже простий, має свою внутрішню будову. Уміння побачити цю будову, описати її та зрозуміти, як частини взаємодіють, – важлива навичка майбутнього ІТ-фахівця.

Детальна інструкція виконання

Крок 1. Вибір додатку для аналізу.

Оберіть **простий, добре знайомий вам додаток**. Важливо, щоб ви чітко уявляли його функціонал. *Варіанти:*

- Калькулятор (звичайний або інженерний).
- Додаток "Нотатки" (як на телефоні).
- Список покупок (простий todo-менеджер).
- Будильник / Таймер.
- Гра "Вгадай число" (консольна гра).
- Простий конвертер валют.

Крок 2. Аналіз та декомпозиція.

Подумайте, з яких логічних частин (модулів) складається ваш додаток. Запишіть їх. Для кожного модуля визначте:

- Назву.
- Основне завдання (функцію).
- Які дані він отримує від інших модулів (вхід).
- Які дані він передає іншим модулям (вихід).

Крок 3. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (250-350 слів).

Структура:

1. Вступ. Назва додатку, коротко про його призначення.

2. Опис модулів: для 3-4 модулів наведіть таблицю або список з описами (назва, функція, вхід/вихід).

3. Взаємодія модулів: опишіть, в якому порядку відбувається типова робота додатку (наприклад, "Користувач вводить дані в модулі А, модуль А передає їх модулю Б, модуль Б обробляє і повертає результат у модуль В"). Додайте просту текстову схему або опишіть словами.

4. Приклад коду: наведіть код (5-10 рядків) для одного з модулів мовою Python. Код має бути працездатним і демонструвати основну логіку модуля. Додайте коментарі.

5. Висновок: чому модульність корисна для цього додатку? Як би ви змінили архітектуру, якби додаток треба було розширити?

Варіант Б: Ментальна карта.

У центрі – назва додатку.

Перший рівень гілок: назви модулів (3-4 гілки).

Для кожного модуля на другому рівні вкажіть: "Функція", "Вхідні дані", "Вихідні дані".

Окремо створіть гілку "Взаємодія", де стрілочками покажіть зв’язки між модулями.

Додайте гілку "Код модуля (приклад)", в яку вставте невеликий фрагмент коду (текстом або скріншотом).

Крок 4. Код.

Обов’язково перевірте ваш код у будь-якому онлайн-редакторі (Replit, Google Colab), щоб він точно працював.

Крок 5. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2). Це можуть бути статті на W3Schools, Programiz, відео на YouTube, книги.

Завантажте готовий файл (PDF, DOCX, JPG, PNG) у відповідний розділ на платформі Moodle. Якщо використовуєте онлайн-інструмент (наприклад, Canva для ментальної карти), надайте посилання з доступом на читання.

Критерії оцінювання самостійної роботи:

Повнота аналізу: описано не менше 3 модулів, чітко визначено їх функції.

Логічність взаємодії: опис взаємодії зрозумілий і несуперечливий.

Коректність коду: код синтаксично правильний, виконує заявлену функцію, має коментарі.

Якість оформлення: структурованість, грамотність, використання джерел.

Самостійність: робота виконана власноруч, а не скопійована.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Що таке архітектура програмного забезпечення? Чому важливо думати про архітектуру до початку написання коду?

2. Поясніть своїми словами, що означає «модульність».

3. Назвіть три переваги модульного підходу до розробки програм.

4. Що таке зв’язність (cohesion)? Чому бажано, щоб модуль мав високу зв’язність?

5. Як ви розумієте термін «взаємодія модулів»? Наведіть приклад з реального життя (не з програмування).

6. Уявіть, що ви розробляєте програму для ведення щоденника. Які модулі ви б виділили?

7. Чи може одна програма обійтися без поділу на модулі? Які проблеми виникнуть, якщо написати весь код в одному великому файлі без поділу на функції?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Базові ресурси:

1. W3Schools: Розділи, присвячені веб-розробці, часто демонструють модульний підхід (наприклад, окремо HTML, CSS, JavaScript). https://www.w3schools.com/

2. Programiz: Пошук за термінами "modular programming", "function" – пояснення на прикладах Python. https://www.programiz.com/

3. GeeksforGeeks: Статті про модульність у програмуванні (англ.) – доступною мовою. https://www.geeksforgeeks.org/

Книги (для тих, хто хоче зануритись глибше):

4. Стів Макконнелл. «Code Complete» (Розділи про проєктування та модульність). Класична книга, хоча для початківців може бути складною, але окремі розділи варто переглянути.

5. Мартін Фаулер. «Рефакторинг. Поліпшення існуючого коду». (Книга про те, як робити код модульнішим і чистішим).

Відео

6. YouTube-канали: CS50 (Гарвардський курс), freeCodeCamp, Simple Programmer – шукайте відео на тему "Introduction to Software Architecture" або "Modular Programming".

Інструменти:

7. Draw.io (diagrams.net): https://app.diagrams.net/

8. Replit: https://replit.com/

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Мисліть «модулями» в повсякденному житті.** Будь-який складний процес можна розкласти на прості кроки. Наприклад, приготування їжі: модуль «підготовка інгредієнтів», модуль «термічна обробка», модуль «сервірування». Це допомагає тренувати навичку декомпозиції.

2. **Не бійтеся змінювати архітектуру.** Перший варіант розбиття на модулі майже ніколи не буває ідеальним. У процесі написання коду ви зрозумієте, що якийсь модуль варто розділити на два, або, навпаки, об’єднати. Це нормально.

3. **Малюйте схеми.** Навіть якщо ви пишете програму самі для себе, швидка схема на папері допоможе вам не заплутатися в логіці. А якщо ви працюєте в команді – схема стане мовою спілкування.

4. **Починайте з простого.** Не намагайтеся одразу спроєктувати ідеальну систему з десятками модулів. Візьміть найпростіший додаток і спробуйте зробити його структуру максимально зрозумілою. З досвідом прийде і складність.

ПРАКТИЧНЕ ЗАНЯТТЯ 8
ВСТУП ДО ТЕСТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників базового розуміння важливості тестування в процесі розробки програмного забезпечення, знайомство з основними видами тестування та отримання первинних практичних навичок написання простих тестів.

Основні завдання заняття:

1. Ознайомити з ключовими поняттями: якість програмного забезпечення (Quality Assurance, QA), тестування, баг (дефект), тестовий сценарій (test case).

2. Дослідити різні рівні тестування: модульне (unit), інтеграційне (integration), системне (system), а також регресійне тестування (regression).

3. Розвинути практичні навички створення тестових сценаріїв на основі функціональних вимог до програми.

4. Розвинути навички написання модульних тестів з використанням бібліотеки unittest у Python.

5. Розвинути ключові навички:

- *аналітичне мислення*: вміння передбачати можливі помилки та граничні випадки в роботі програми.
- *увага до деталей*: точне документування кроків для відтворення помилки.
- *командна робота*: спільне створення тестів та аналіз результатів.
- *комунікація*: представлення знайдених дефектів (або тестових сценаріїв) у зрозумілій формі.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Чому тестування важливе?

Уявіть, що ви користуєтесь банківським додатком, і він неправильно рахує ваш баланс. Або що ваш улюблений додаток для виклику таксі раптово "падає" саме тоді, коли ви найбільше поспішаєте. Щоб уникнути таких ситуацій, існує тестування.

Мета тестування – не довести, що програма працює (це неможливо зробити на 100%), а знайти якомога більше помилок (дефектів, багів) до того, як програма потрапить до кінцевого користувача.

2.2. Основні поняття

Якість програмного забезпечення (QA): це процес, який охоплює всі етапи розробки, спрямований на забезпечення того, щоб продукт відповідав вимогам і очікуванням користувачів.

Тестування: процес виконання програми з метою виявлення помилок.

Тестовий сценарій (Test Case): документ, який описує конкретну перевірку. Він містить:

- ID: унікальний номер.
- Назва: короткий опис того, що перевіряється.
- Передумови: що має бути виконано перед тестом (наприклад, "користувач залогінений").
- Кроки: детальна послідовність дій.
- Очікуваний результат: що має статися після виконання кроків.
- Тип тестування (наприклад, модульний, функціональний).

2.3. Рівні тестування

Модульне тестування (Unit Testing): тестування окремих, ізольованих компонентів програми – функцій, методів, класів. Зазвичай виконується самими розробниками. Дозволяє швидко знайти помилки на ранніх етапах.

Інтеграційне тестування (Integration Testing): тестування взаємодії між кількома модулями. Перевіряє, чи правильно частини програми "спілкуються" одна з одною.

Системне тестування (System Testing): тестування всієї програми в цілому, як єдиної системи. Перевіряє, чи відповідає програма всім функціональним і нефункціональним вимогам.

Регресійне тестування (Regression Testing): повторне тестування вже перевірених функцій після внесення змін у код. Мета – переконатися, що виправлення однієї помилки не створило інших помилок.

2.4. Види тестування за рівнем знань про систему

Тестування "чорної скриньки" (Black Box): тестувальник не знає, як влаштована програма всередині. Він перевіряє лише її зовнішню поведінку, орієнтуючись на вимоги.

Тестування "білої скриньки" (White Box): тестувальник бачить код і будує тести, виходячи з його внутрішньої структури (наприклад, модульні тести).

2.5. Бібліотека unittest у Python

unittest – це вбудована бібліотека Python для створення та запуску модульних тестів. Вона дозволяє автоматизувати перевірку коду. Основна ідея:

1. Створити клас-нащадок від unittest.TestCase.

2. Всередині класу створити методи, назва яких починається з test_.

3. У цих методах використовувати спеціальні методи перевірки (assertions), наприклад:

- assertEqual(a, b) – перевіряє, чи дорівнює a b.
- assertTrue(x) – перевіряє, чи x є істиною.
- assertRaises(SomeException, func, arg) – перевіряє, чи викличе функція очікувану помилку.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп'ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки.
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - онлайн-редактори коду Python:
    - Replit: https://replit.com/ – найкраще підходить, оскільки підтримує виконання unittest.
    - Google Colab: https://colab.research.google.com/ – підходить для написання коду, але для unittest може знадобитися додаткове налаштування. Replit простіше.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Створення тестових сценаріїв для програми

Мета завдання: навчитися розробляти тестові сценарії на основі функціональності програми, розуміти різницю між очікуваним та фактичним результатом.

Теоретичне підґрунтя

Перед тим як автоматизувати тести, тестувальник (QA-інженер) має скласти план перевірки – тестові сценарії. Це набір інструкцій, які описують, як саме перевіряти програму. Сьогодні ми спробуємо себе в ролі QA-інженерів.

Детальна інструкція виконання

Етап 1. Формування груп та вибір продукту (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть простий програмний продукт для створення тестів. *Приклади:*

- Калькулятор (додавання, віднімання, множення, ділення на 0).
- Форма логіну на сайті (поля: email, пароль, кнопка "Увійти").
- Список справ (To-Do List) (додати завдання, видалити, позначити як виконане).
- Поле пошуку на сайті (введення тексту, натискання Enter, відображення результатів).

Етап 2. Створення тестових сценаріїв (10 хвилин).

Використовуйте наступну структуру для кожного сценарію. Створіть мінімум **3-5 сценаріїв**.

Заповніть таблицю на папері або в спільному документі:

ID	Назва тесту	Передумови	Кроки	Очікуваний результат	Тип тестування (приблизно)
Приклад: Калькулятор					
TC-01	Перевірка додавання двох позитивних чисел	Калькулятор відкрито	1. Натиснути 5 2. Натиснути + 3. Натиснути 3 4. Натиснути =	На екрані відображається 8	Функціональне (Системне)
TC-02	Перевірка ділення на нуль	Калькулятор відкрито	1. Натиснути 10 2. Натиснути ÷ 3. Натиснути 0 4. Натиснути =	На екрані відображається повідомлення "Помилка" або "Не можна ділити на 0"	Функціональне (Системне)
Приклад: Форма логіну					
TC-03	Вхід з правильними даними	Користувач зареєстрований (уявно)	1. Ввести test@mail.com в поле Email 2. Ввести password123 в поле Пароль 3. Натиснути "Увійти"	Користувач перенаправляється на особистий кабінет	Інтеграційне (перевірка взаємодії БД та інтерфейсу)
TC-04	Вхід з неправильним	Користувач зареєстрований	1. Ввести test@mail.com	З’являється повідомлення:	Функціональне

	паролем	(уявно)	2. Ввести wrongpass 3. Натиснути "Увійти"	"Невірний email або пароль".	

Етап 3. Оформлення презентації (5 хвилин).

Створіть 2-3 слайди.

Структура:

- *Слайд 1.* Назва продукту та його короткий опис (основні функції).
- *Слайд 2.* Таблиця з тестовими сценаріями (можна винести 2 найцікавіші на слайд, інші розказати усно).
- *Слайд 3.*

Висновок. Чому ці тести важливі? Як вони допомагають гарантувати якість продукту? Які ризики вони покривають? (наприклад, ТС-02 покриває ризик "падіння" програми при діленні на нуль).

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група презентує свої сценарії (2-3 хвилини).

Загальна дискусія:

- Чи були у вас сценарії, які перевіряють не "щасливий шлях", а помилкові ситуації (негативні тести)? Чому вони важливі?
- Які тести, на вашу думку, найважливіші для будь-якого продукту?
- Чи можна створити тести, які гарантують повну відсутність помилок?

Критерії оцінювання (Завдання 1):

Повнота: створено не менше 3 сценаріїв, які покривають різні аспекти роботи програми.

Коректність: чітко прописані кроки та очікуваний результат.

Різноманітність: наявність як позитивних (happy path), так і негативних тестів (перевірка помилок).

Командна робота.

Якість презентації.

Завдання 2. Тестування простої програми на Python

Мета завдання: отримати практичний досвід написання модульних тестів з використанням бібліотеки unittest у Python, навчитися запускати тести та аналізувати результати.

Теоретичне підґрунтя

Модульне тестування – це основа автоматизованої перевірки коду. Розробники пишуть тести для своїх функцій одразу після (або навіть до) написання самого коду. Це дозволяє швидко знаходити помилки і не ламати стару функціональність при додаванні нової.

Детальна інструкція виконання

Етап 1. Підготовка середовища (2 хвилини).

Об’єднайтесь у пари.

Один з пари відкриває Replit (replit.com) і створює новий Python-репліт.

Етап 2. Написання функції для тестування (5 хвилин).

Напишіть просту функцію, яку будете тестувати. Наприклад, функцію для обчислення середнього значення списку чисел. Вставте наступний код у файл main.py:

```
def average(numbers):
    """
    Обчислює середнє арифметичне списку чисел.
    Якщо список порожній, повертає 0.
    """
    if not numbers:  # перевірка на порожній список
        return 0
    return sum(numbers) / len(numbers)
```

Етап 3. Написання модульних тестів (10 хвилин).

У тому ж файлі (або краще в новому, але в Replit можна в одному) додайте код для тестів. Для цього створіть клас, який успадковує unittest.TestCase.

```
import unittest

# ... (функція average має бути визначена вище)

class TestAverageFunction(unittest.TestCase):

    def test_average_positive_numbers(self):
        """Тестує середнє для списку позитивних чисел."""
        result = average([1, 2, 3, 4, 5])
        self.assertEqual(result, 3.0)  # (1+2+3+4+5)/5 = 15/5 = 3.0

    def test_average_mixed_numbers(self):
```

```
        """Тестує середнє для списку з додатними та від'ємними числами."""
        result = average([-10, 10])
        self.assertEqual(result, 0.0)  # (-10+10)/2 = 0
    def test_average_empty_list(self):
        """Тестує поведінку функції при передачі порожнього списку."""
        result = average([])
        self.assertEqual(result, 0)  # Очікуємо 0

    def test_average_single_number(self):
        """Тестує середнє для списку з одним числом."""
        result = average([42])
        self.assertEqual(result, 42.0)

if __name__ == '__main__':
    unittest.main()
```

Етап 4. Запуск тестів та аналіз результатів (3 хвилини).

Натисніть кнопку "Run" у Replit.

В консолі ви повинні побачити результат виконання тестів. Якщо всі тести пройшли, ви побачите щось на кшталт:

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

Якщо якийсь тест не пройде, ви побачите F замість крапки та детальний звіт про помилку.

Зробіть скріншот результату.

Етап 5. Фіксація результатів (5 хвилин).

Запишіть (скопіюйте) код програми та тестів у текстовий документ (Google Docs).

Додайте скріншот результату виконання тестів.

Підготуйте коротке пояснення: які кейси ви перевіряли і чому.

Етап 6. Презентація та обговорення (5 хвилин).

За бажанням, 1-2 пари демонструють свій код та результати.

Загальне обговорення:

▪ Чому важливо тестувати так звані "граничні випадки" (empty list, single number)?

▪ Як автоматичні тести допомагають розробнику не зламати старий код, коли він додає нові функції? (регресія).

Критерії оцінювання (Завдання 2):

Працездатність коду: функція написана правильно.

Коректність тестів: тести перевіряють різні сценарії, використовуються правильні методи assert....

Результат: тести запускаються і (бажано) проходять успішно.

Розуміння: студенти можуть пояснити, що робить кожен тест.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Аналіз тестування та забезпечення якості

Мета завдання: поглибити знання про один із видів тестування, навчитися знаходити та аналізувати інформацію, а також застосувати знання на практиці, створивши власний приклад тесту.

Вступ. У світі професійної розробки ПЗ існує безліч видів тестування. Розуміння того, чим вони відрізняються і для чого призначені, допоможе вам стати кращим розробником або QA-інженером.

Детальна інструкція виконання

Крок 1. Вибір виду тестування.

Оберіть **один** вид тестування для глибокого дослідження:

- Модульне тестування (Unit Testing)
- Інтеграційне тестування (Integration Testing)
- Системне тестування (System Testing)
- Регресійне тестування (Regression Testing)
- Приймальне тестування (Acceptance Testing)
- Навантажувальне тестування (Load/Performance Testing)

Крок 2. Дослідження.

Знайдіть відповіді на наступні питання, використовуючи мінімум 2-3 надійні джерела (SoftwareTestingHelp, Guru99, статті на DOU, відео на YouTube):

1. Мета: З якою метою проводиться цей вид тестування? Що він має виявити?

2. Об’єкт тестування: Що саме тестується? (Окрема функція? Взаємодія модулів? Вся система?)

3. Хто виконує: Розробник, QA-інженер, чи обидва?

4. Коли виконується: На якому етапі життєвого циклу розробки?

5. Переваги: Чому цей вид тестування важливий?

6. Недоліки / обмеження: Що він не може перевірити? Чи є він дорогим або складним?

Крок 3. Створення прикладу.

Придумайте **дуже просту функцію** (наприклад, функцію, яка перевіряє, чи є число парним, або функцію конвертації температури).

Напишіть для цієї функції **2-3 модульні тести** з використанням unittest (навіть якщо ви обрали інтеграційне тестування, модульні тести – це універсальний спосіб показати розуміння автоматизованої перевірки). Переконайтеся, що код працює в Replit.

Крок 4. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (250-350 слів).

Структура:

1. Вступ. Назва обраного виду тестування.

2. Основна частина: відповіді на питання з Кроку 2 (мета, об’єкт, хто, коли, переваги, недоліки). Наведіть приклад з реального життя, де цей вид тестування був би корисним.

3. Практична частина: наведіть код вашої функції та модульних тестів до неї. Поясніть, що перевіряє кожен тест.

4. Висновок: чому ви вважаєте цей вид тестування важливим для забезпечення якості ПЗ?

Варіант Б: Ментальна карта.

У центрі – назва виду тестування.

Перший рівень гілок: "Мета", "Об’єкт", "Хто проводить", "Коли", "Переваги", "Недоліки", "Приклад коду".

На гілці "Приклад коду" зробіть вставку з фрагментом коду або посиланням на Replit.

Крок 5. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина аналізу: робота демонструє розуміння суті обраного виду тестування, а не просто механічне копіювання тексту.

Коректність прикладу: код функції та тестів написано правильно, тести мають сенс.

Структурованість: логічність викладу, наявність всіх необхідних елементів.

Доказовість: використання авторитетних джерел, коректність посилань.

Оформлення: чистота, відсутність помилок.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Чому тестування є невід’ємною частиною розробки програмного забезпечення?
2. Що таке тестовий сценарій (test case)? З яких частин він складається?
3. Поясніть різницю між модульним, інтеграційним та системним тестуванням.
4. Що таке регресійне тестування? Чому воно потрібне?
5. Яка різниця між "позитивним" та "негативним" тестом? Наведіть приклад.
6. Що таке модульний тест? Якою бібліотекою в Python ми можемо його створити?
7. Чи може набір тестів гарантувати повну відсутність помилок у програмі? Чому?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Базові ресурси:

1. Software Testing Help: https://www.softwaretestinghelp.com/ – один з найкращих ресурсів для початківців у тестуванні. Величезна кількість статей, туторіалів.

2. Guru99: https://www.guru99.com/software-testing.html – ще один чудовий ресурс з простими поясненнями.

3. DOU.ua: Розділ "QA": Статті українською про тестування, поради для початківців.

Документація:

4. unittest – Unit testing framework (Python docs): https://docs.python.org/3/library/unittest.html – офіційна документація. Корисно мати під рукою для довідки.

Книги

5. Сем Канер, Джек Фолк, Енг Кек Нгуєн. "Тестування програмного забезпечення. Фундаментальні концепції менеджменту бізнес-додатків". Класична книга.

6. Ліза Кріспін, Джанет Грегорі. "Гнучке тестування: Практичний посібник для тестувальників ПЗ і гнучких команд".

Відео

7. YouTube-канали: "Guru99", "Software Testing Help", "freeCodeCamp" (є відео з основами тестування).

Інструменти:

8. Replit: https://replit.com/

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Тестування – це не руйнування, а створення якості.** Хороший тестувальник (або розробник, який тестує свій код) не намагається "завалити" програму, а допомагає зробити її кращою та надійнішою.

2. **Думайте як користувач.** Коли пишете тестові сценарії, намагайтеся передбачити, що може зробити користувач, навіть якщо це не передбачено інструкцією. Користувачі дуже креативні в пошуку способів "зламати" програму.

3. **Автоматизуйте рутину.** Модульні тести, які ви пишете на Python, – це приклад автоматизації. Комп’ютер виконає тисячі таких тестів за секунди, що значно швидше, ніж ручне тестування.

4. **Червоний -> Зелений -> Рефакторинг.** Це мантра TDD (Test-Driven Development). Спочатку пишемо тест, який не проходить (червоний), потім пишемо код, щоб тест пройшов (зелений), а потім покращуємо код (рефакторинг). Це чудова дисципліна.

5. **Не бійтеся помилок.** Знайти помилку на етапі тестування – це свято, а не катастрофа. Краще знайти її зараз, ніж завтра, коли програма вже у користувача.

ПРАКТИЧНЕ ЗАНЯТТЯ 9
СТРУКТУРА ТА ПРИНЦИПИ РОБОТИ СУЧАСНИХ ІТ-КОМПАНІЙ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників цілісного уявлення про внутрішню будову сучасних ІТ-компаній, розмаїття професійних ролей, принципи управління проєктами та особливості корпоративної культури, що впливають на ефективність роботи.

Основні завдання заняття:

1. Ознайомити з типовими організаційними структурами ІТ-компаній (функціональна, дивізіональна, матрична, плоска), ключовими відділами та посадами.

2. Дослідити на прикладі реальних компаній (Google, EPAM, SoftServe, Grammarly тощо), як організована робота, які ролі є критичними та як формується корпоративна культура.

3. Розвинути навички аналізу відкритої інформації про компанії (сайти, LinkedIn, Glassdoor) та її структурування.

4. Розвинути навички командної роботи, планування проєкту, розподілу ролей та презентації результатів.

5. Сприяти усвідомленому вибору майбутнього місця роботи та розумінню корпоративних цінностей.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Організаційна структура ІТ-компанії

Організаційна структура визначає, як розподілені завдання, ролі та відповідальність всередині компанії, а також як здійснюється координація та комунікація. В ІТ-сфері поширені такі типи структур:

Функціональна структура: співробітники групуються за виконуваними функціями (відділ розробки, відділ тестування, відділ маркетингу, HR тощо). Кожен відділ має свого керівника. *Плюси:* чітка спеціалізація, легкість управління фахівцями. *Мінуси:* ускладнена комунікація між відділами, повільне прийняття рішень.

Дивізіональна (продуктова) структура: компанія поділяється на окремі підрозділи (дивізіони), кожен з яких відповідає за конкретний продукт, проєкт

або ринок. Усередині дивізіону можуть бути свої розробники, тестувальники, маркетологи. *Плюси:* гнучкість, орієнтація на результат. *Мінуси:* можливе дублювання функцій.

Матрична структура: поєднує функціональний і проєктний підходи. Співробітник може підпорядковуватись і керівнику свого відділу, і керівнику проєкту. *Плюси:* гнучке використання ресурсів. *Мінуси:* складність звітності, потенційні конфлікти.

Плоска (flat) структура: характерна для стартапів і невеликих компаній. Має мало рівнів управління або взагалі їх не має. *Плюси:* швидка комунікація, високий рівень відповідальності. *Мінуси:* нестабільність при зростанні компанії.

2.2. Ключові ролі в IT-компанії

Розробник (Developer/Engineer): програміст, який безпосередньо створює код. Може спеціалізуватися на бекенді (Python, Java, C#), фронтенді (JavaScript, TypeScript), мобільній розробці (Swift, Kotlin), базах даних тощо.

QA-інженер (Quality Assurance): займається тестуванням програмного забезпечення, пошуком помилок, написанням тестової документації.

DevOps-інженер: відповідає за інфраструктуру, автоматизацію розгортання, безперервну інтеграцію (CI/CD), хмарні сервіси.

Бізнес-аналітик (BA): вивчає потреби бізнесу/клієнта, формулює вимоги до продукту, допомагає команді зрозуміти, що саме потрібно зробити.

Проєктний менеджер (PM) / Scrum-майстер: планує роботу, слідкує за термінами, комунікує із замовником, усуває перешкоди.

Дизайнер (UI/UX Designer): проектує зручний та естетичний інтерфейс, досліджує поведінку користувачів.

Product Owner (Власник продукту): в Agile-командах представляє інтереси замовника, визначає пріоритети в беклозі.

HR-менеджер / Talent Sourcer: займається пошуком, наймом та адаптацією співробітників, підтримує корпоративну культуру.

Sales / Marketing: продають продукт, залучають клієнтів, просувають бренд компанії.

2.3. Корпоративна культура в IT

Корпоративна культура – це набір цінностей, норм, традицій і моделей поведінки, які поділяють співробітники. Вона проявляється у всьому: від стилю спілкування до умов праці.

Елементи культури:

- Місія та цінності: наприклад, "Don’t be evil" (Google), "Customer obsession" (Amazon).
- Формат роботи: офісний, гібридний, віддалений (remote-first).
- Стиль управління: демократичний, авторитарний, самоорганізація.
- Традиції та заходи: хакатони, мітапи, корпоративні свята, тімбілдінги.
- Умови праці: гнучкий графік, ДМС, навчання за кошт компанії, комфортний офіс.

Чому це важливо. Сильна корпоративна культура підвищує лояльність співробітників, залучає талановитих фахівців, покращує комунікацію та, зрештою, впливає на якість продукту.

2.4. Інструменти управління проєктами

В IT-компаніях використовують різні інструменти для організації роботи:

- Jira: потужний інструмент для відстеження задач, багів, управління Agile-процесами (Scrum, Kanban). Стандарт для великих команд.
- Trello, Asana, Monday.com: простіші інструменти для невеликих команд, візуалізації задач.
- Confluence: вікі-система для ведення документації, зберігання знань.
- Slack, Microsoft Teams, Google Chat: інструменти для комунікації.
- GitHub, GitLab, Bitbucket: платформи для зберігання коду, код-рев’ю, CI/CD.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постерів).
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - онлайн-дошки для спільної роботи (Miro, Draw.io) – для створення схем структур.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.
  - текстовий процесор (Google Docs, Microsoft Word Online).

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Аналіз структури IT-компанії

Мета завдання: навчитися знаходити, аналізувати та систематизувати інформацію про реальні IT-компанії, розуміти зв’язок між їхньою структурою, культурою та успіхом на ринку.

Теоретичне підґрунтя

Кожна компанія має унікальну структуру, яка відображає її історію, розмір, сферу діяльності та цінності. Аналіз успішних компаній допомагає зрозуміти, як організувати роботу ефективно.

Детальна інструкція виконання

Етап 1. Формування груп та вибір компанії (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть **одну реальну IT-компанію** для аналізу. *Рекомендації:*

- *Глобальні гіганти:* Google (Alphabet), Microsoft, Amazon, Meta (Facebook).
- *Великі українські продуктові компанії:* Grammarly, GitLab (хоча вона розподілена), Jooble, Ajax Systems.
- *Великі аутсорсингові компанії:* EPAM, SoftServe, GlobalLogic, Ciklum.
- *Цікаві стартапи* (наприклад, з рейтингів DOU).

Етап 2. Збір інформації (10 хвилин).

Використовуйте різноманітні джерела:

- Офіційний сайт компанії: розділи "About us", "Careers", "Company culture", блоги.
- LinkedIn: шукайте сторінку компанії, перегляньте, скільки співробітників, які посади є, які вакансії відкриті.
- Glassdoor: відгуки співробітників, інформація про зарплати, інтерв’ю, переваги та недоліки роботи.
- DOU.ua: для українських компаній – профілі компаній, огляди ринку, статті.
- YouTube: пошукайте відео "A day in the life at [Company Name]", інтерв’ю з топ-менеджерами.

Знайдіть відповіді на наступні питання та заповніть таблицю (на папері або в спільному документі):

Критерій	Знайдена інформація
Назва компанії, рік заснування, штаб-квартира	
Основний вид діяльності (продукт/аутсорсинг/гібрид)	
Приблизна кількість співробітників	
Тип організаційної структури (функціональна, дивізіональна, плоска? – спробуйте визначити)	
Основні відділи / департаменти (перелічіть 4-5)	
Ключові ролі в команді розробки (перелічіть ті, що часто згадуються у вакансіях)	
Корпоративна культура (цінності, формат роботи, бонуси, заходи)	
Інструменти управління проєктами (якщо згадуються)	

Етап 3. Створення презентації/постеру (5 хвилин).

На основі зібраної інформації створіть 2-3 слайди або постер.

Структура:

- *Слайд 1.* Назва компанії, логотип, короткий опис (з чим асоціюється, чим відома).
- *Слайд 2.* Візуалізація структури (намалюйте просту схему: CEO -> відділи -> ключові ролі). Додайте перелік основних відділів.
- *Слайд 3.* Ключові ролі (опишіть 2-3 найцікавіші) та елементи корпоративної культури (3-5 пунктів).

Висновок: Як структура та культура допомагають цій компанії бути успішною?

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група презентує результати (2-3 хвилини).

Загальна дискусія:

- Які компанії мають більш плоску структуру, а які – ієрархічну? Як це впливає на швидкість прийняття рішень?
- Які елементи корпоративної культури вам особисто імпонують? Чому?
- Як відрізняється структура продуктової компанії (Grammarly) від аутсорсингової (EPAM)?

Критерії оцінювання (Завдання 1):

Повнота дослідження: використано різні джерела, зібрано достатньо інформації за всіма критеріями.

Глибина аналізу: спроба визначити тип структури, зробити висновки про зв’язок культури та успіху.

Візуалізація: схема структури зрозуміла, інформація структурована.

Командна робота.

Якість презентації.

Завдання 2. Рольова гра «Планування проєкту в ІТ-компанії»

Мета завдання: відчути на собі, як розподіляються ролі та обов'язки в команді під час старту нового проєкту, навчитися створювати первинний план робіт.

Теоретичне підґрунтя

У реальній компанії будь-який проєкт починається зі збору команди, визначення ролей, цілей та основних етапів. Це завдання – симуляція такого процесу.

Детальна інструкція виконання

Етап 1. Формування команд та вибір проєкту (3 хвилини).

Об'єднайтесь у групи по 2-3 особи.

Оберіть уявний проєкт. *Приклади:*

- Чат-бот для студентів (довідкова інформація, розклад).
- Сайт-портфоліо для викладача.
- Мобільний додаток для відстеження звичок.
- Інструмент для спільного планування завдань (як Trello, але простіше).

Етап 2. Розподіл ролей (5 хвилин).

Визначте, хто яку роль виконуватиме в цьому проєкті. Можна суміщати, але краще розподілити. *Можливі ролі:*

- Менеджер проєкту (PM): відповідає за терміни, комунікацію, планування.
- Розробник (Developer): відповідає за написання коду.
- Дизайнер (Designer): відповідає за зовнішній вигляд та зручність.
- Тестувальник (QA): відповідає за перевірку якості.
- Бізнес-аналітик (BA): досліджує потреби "користувачів" (студентів/викладачів).

Етап 3. Складання плану проєкту (10 хвилин).

Використовуйте наступний шаблон. Запишіть план на папері або в Google Docs.

Назва проєкту: [Назва]

Мета проєкту: [Одне речення, чого хочемо досягти]

Команда та ролі:

- [Ім'я] – Менеджер проєкту (обов'язки: ...)
- [Ім'я] – Розробник (обов'язки: ...)
- [Ім'я] – Дизайнер (обов'язки: ...)

Основні етапи (4-5):

1. Планування та дослідження (2 дні): [хто робить] аналізує потреби, [хто робить] створює прототипи. Інструменти: Miro, Google Docs.

2. Дизайн (3 дні): [хто робить] створює макети інтерфейсу. Інструменти: Figma.

3. Розробка основного функціоналу (7 днів): [хто робить] пише код, використовуючи Python/Flask. Інструменти: GitHub.

4. Тестування та виправлення помилок (3 дні): [хто робить] перевіряє роботу, [хто робить] виправляє баги. Інструменти: Trello (для відстеження багів), GitHub Issues.

5. Запуск (1 день): розгортання на сервері.

Інструменти, які будемо використовувати:

- Управління: Trello
- Код: GitHub
- Комунікація: Slack / Telegram
- Дизайн: Figma

Етап 4. Презентація та обговорення (12 хвилин).

Кожна група коротко (1 хвилина) презентує свій проєкт, ролі та план.

Загальна дискусія:

- Чи легко було розподілити ролі? Чи виникали конфлікти через обов'язки?
- Які ролі, на вашу думку, є критично важливими для успіху, а без яких можна обійтися на ранньому етапі?
- Чи реальні терміни, які ви поставили?

Критерії оцінювання (Завдання 2):

Чіткість плану: визначено мету, етапи, ролі та інструменти.

Реалістичність: план виглядає здійсненним, етапи логічні.

Розподіл ролей: всі учасники мають чітко визначені обов'язки.

Участь в обговоренні.

Командна робота.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Дослідження корпоративної культури ІТ-компанії

Мета завдання: поглибити навички самостійного дослідження ринку праці та компаній, навчитися аналізувати корпоративну культуру як фактор вибору майбутнього роботодавця.

Вступ. Коли ви будете обирати місце роботи, важливими будуть не лише технології та зарплата, а й те, в якому середовищі вам доведеться працювати. Це дослідження допоможе вам зрозуміти, на що звертати увагу.

Детальна інструкція виконання

Крок 1. Вибір компанії.

Оберіть **одну ІТ-компанію** для глибокого дослідження. Це може бути як всесвітньо відома (Google, Microsoft, Amazon), так і українська (SoftServe, EPAM, Grammarly, Ajax Systems, Rozetka, будь-який цікавий стартап).

Крок 2. Глибоке дослідження.

Використовуйте всі доступні джерела:

- Офіційний сайт (розділи "Про нас", "Кар’єра", "Культура", "Блог").
- LinkedIn (сторінка компанії, профілі співробітників, вакансії).
- Glassdoor (відгуки співробітників та колишніх співробітників – зверніть увагу на розділи "Pros", "Cons").
- DOU.ua (для українських компаній – профіль компанії, статті, інтерв’ю з засновниками).
- YouTube (відео з конференцій, інтерв’ю, "a day in the life").
- Якщо є знайомі, які там працюють – можна запитати особисто (це називається "нетворкінг").

Крок 3. Структурування звіту.

Створіть звіт (300-400 слів) у Google Docs, Microsoft Word або ментальну карту (MindMeister, Canva), який містить такі розділи:

1. Загальна інформація про компанію:
   - Назва, рік заснування, країна/місто головного офісу.

- Основний продукт/послуга (чим займається).
- Приблизна кількість співробітників, географія присутності.

2. Організаційна структура (як ви її уявляєте):

- Які основні відділи існують (на основі аналізу вакансій та сайту).
- Чи є інформація про те, як організовані команди (наприклад, продуктові команди, гільдії, глави)?

3. Корпоративна культура (найдетальніша частина):

- Цінності: Які цінності компанія декларує? (наприклад, "Be open and honest", "Customer first").
- Формат роботи: Віддалено, гібридно, в офісі? Чи є вільний графік?
- Умови праці та бонуси: Які є пільги (ДМС, оплата навчання, спортзал)? Чи проводяться хакатони, мітапи, корпоративи?
- Стиль управління: Як приймаються рішення? Чи є ментори? (можна здогадатися з відгуків).
- Атмосфера: Як співробітники відгукуються про колег, про керівництво, про ставлення до людей? (Glassdoor, DOU).

4. Інструменти управління проєктами та розробки:

- Які інструменти згадуються у вакансіях? (Jira, Confluence, GitHub, Slack, Figma тощо).

5. Приклад уявного проєкту:

- Запропонуйте, який новий проєкт (або нову функцію для існуючого продукту) могла б розробити ця компанія. Опишіть його одним абзацом.

Крок 4. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 3: офіційний сайт, LinkedIn/Glassdoor, стаття/відео).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина аналізу: робота демонструє розуміння структури та культури компанії, а не просто набір фактів.

Використання різних джерел: інформація взята не лише з сайту, а й з відгуків, статей, відео.

Структурованість: чітке слідування плану, логічний виклад.

Креативність прикладу проєкту: проєкт відповідає профілю компанії.

Оформлення: чистота, відсутність помилок, коректні посилання.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Які типи організаційних структур найчастіше зустрічаються в IT-компаніях? Опишіть їхні основні відмінності.

2. Назвіть 5-7 ключових ролей у типовій продуктовій IT-команди. Хто за що відповідає?

3. Чим відрізняються обов'язки Project Manager та Scrum Master?

4. Що таке корпоративна культура? Чому вона важлива для успіху компанії та задоволеності співробітників?

5. Які інструменти для управління проєктами є найпопулярнішими? Для чого кожен з них використовується?

6. Як відрізняється структура невеликого стартапу від структури великої корпорації на кшталт Google?

7. Які джерела інформації ви можете використати, щоб дізнатися більше про компанію, де хотіли б працювати?

8. Чому в IT-компаніях така популярна віддалена та гібридна робота? Які це має переваги та виклики?

9. Що таке "плоска структура" (flat structure)? Для яких компаній вона характерна?

10. Як корпоративна культура може вплинути на ваш вибір роботодавця в майбутньому?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Ресурси для дослідження компаній

1. Glassdoor: https://www.glassdoor.com/ – Відгуки співробітників, зарплати, інтерв'ю.

2. LinkedIn: https://www.linkedin.com/ – Профілі компаній, вакансії, нетворкінг.

3. DOU.ua: https://dou.ua/ – Спільнота програмістів України. Розділи "Компанії", "Ринок праці", "Блоги".

4. The Muse: https://www.themuse.com/ – Сайт з фокусом на культуру компаній, кар'єрні поради.

Статті та огляди

5. Atlassian Team Playbook: https://www.atlassian.com/team-playbook – Інструменти для покращення командної роботи.

6. Google re:Work: https://rework.withgoogle.com/ – Дослідження Google про ефективні команди та культуру.

7. Forbes Tech Council: Статті на тему культури в IT-компаніях.

8. Harvard Business Review: Статті про організаційну структуру та управління.

Книги

9. Патті МакКорд. "Потужні. Як побудувати культуру вільної відповідальності" (досвід Netflix).

10. Бен Хоровіц. "Про важливе. (Не)дитячі питання про бізнес" – про управління технологічними компаніями.

11. Ерік Шмідт, Джонатан Розенберг. "Як працює Google".

Відео

12. YouTube-канали компаній (наприклад, "Life at Google", "SoftServe", "EPAM") – відео про корпоративне життя.

13. Доповіді з конференцій (UA CONferences, IT Arena, DevOps Days) – часто розповідають про культуру та структуру.

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Вивчайте компанії ще до випуску.** Чим раніше ви почнете цікавитися ринком праці та окремими компаніями, тим легше вам буде зробити усвідомлений вибір після закінчення університету. Підпишіться на сторінки цікавих компаній у LinkedIn, читайте їхні блоги.

2. **Культура починається з малого.** Навіть якщо ви потрапите в компанію з прекрасною культурою, пам’ятайте, що її творять самі співробітники. Ваш внесок у доброзичливу атмосферу, готовність допомогти та відкритість до зворотного зв’язку – це вже частина культури.

3. **Не бійтеся змінювати компанії.** Ваша перша робота – це досвід. Можливо, культура першої компанії вам не підійде, і це нормально. Аналізуйте, чого вам бракує, і шукайте місце, де вам буде комфортно розвиватися.

4. **Нетворкінг – це не лише про "корисні знайомства".** Це про можливість дізнатися з перших уст, як реально працює компанія, які в неї проблеми та переваги. Відвідуйте мітапи, спілкуйтеся з доповідачами, ставте запитання.

5. **Поєднуйте мрії з реальністю.** Важливо знайти компанію, де вам подобається і продукт, і люди, і процеси. Але будьте готові, що ідеальних компаній не буває. Визначте для себе 3-5 найважливіших критеріїв (наприклад, цікаві задачі, гнучкий графік, хороший колектив) і орієнтуйтеся на них.

ПРАКТИЧНЕ ЗАНЯТТЯ 10
РІЗНОМАНІТТЯ ПРОФЕСІЙ В ІТ-ГАЛУЗІ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників цілісного уявлення про широкий спектр професій в індустрії інформаційних технологій, розуміння вимог до кожної з них, усвідомлення власних інтересів та схильностей для подальшого вибору спеціалізації.

Основні завдання заняття:

1. Ознайомити з основними категоріями ІТ-професій: розробка (frontend, backend, mobile, embedded), дані (Data Science, Data Analytics), інфраструктура (DevOps, SysAdmin), управління (PM, Scrum Master), тестування (QA), дизайн (UI/UX), безпека (Cybersecurity), бізнес-аналіз (BA) тощо.

2. Дослідити на прикладі конкретних професій їхні ключові обов'язки, необхідні технічні (hard skills) та універсальні (soft skills) навички, а також типові кар'єрні траєкторії.

3. Розвинути навички самостійного дослідження ринку праці, аналізу вакансій та вимог роботодавців.

4. Розвинути ключові навички:

– *аналітичне мислення*: порівняння різних професій, виокремлення спільного та відмінного.

– *комунікація*: представлення інформації про професію, участь в інтерв'ю.

– *самопізнання та планування*: співставлення власних інтересів з вимогами професії, початок формування індивідуальної кар'єрної траєкторії.

– *командна робота*: спільне дослідження та підготовка презентації.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Карта ІТ-професій

ІТ-галузь надзвичайно різноманітна. У ній існує не лише посада "програміст", а десятки різних ролей, які можна умовно поділити на кілька великих категорій:

1. **Розробка (Development/Engineering):**

Frontend-розробник: створює те, що бачить і з чим взаємодіє користувач у браузері. Технології: HTML, CSS, JavaScript, React, Angular, Vue.js.

Backend-розробник: відповідає за серверну частину, бази даних, логіку роботи додатку, API. Технології: Python, Java, C#, PHP, Go, SQL.

Fullstack-розробник: поєднує навички frontend та backend розробника.

Mobile-розробник: створює додатки для мобільних пристроїв (iOS – Swift, Android – Kotlin/Java, кросплатформні – Flutter, React Native).

Embedded-розробник: програмує мікроконтролери, працює з "залізом" (автомобілі, побутова техніка, IoT-пристрої). Мови: C, C++, Rust.

2. **Дані (Data):**

Data Analyst (Аналітик даних): аналізує дані, будує звіти, дашборди, шукає закономірності, допомагає бізнесу приймати рішення. Інструменти: SQL, Python (Pandas), Excel, Tableau.

Data Scientist (Науковець з даних): будує складні моделі машинного навчання для прогнозування, класифікації, рекомендацій. Потребує глибоких знань математики, статистики, Python (Scikit-learn, TensorFlow).

Data Engineer (Інженер даних): будує інфраструктуру для збору, зберігання та обробки величезних обсягів даних, щоб вони були доступні для аналітиків та дата-саєнтистів.

3. **Інфраструктура та операції (Infrastructure & Operations):**

DevOps-інженер: автоматизує процеси розробки, тестування та розгортання (CI/CD), керує хмарною інфраструктурою (AWS, Azure, GCP), налаштовує моніторинг.

Системний адміністратор (SysAdmin): підтримує роботу серверів, мереж, комп'ютерного обладнання.

4. **Управління та аналітика (Management & Analysis):**

Проєктний менеджер (PM): планує роботу команди, слідкує за термінами, комунікує із замовником.

Продакт-менеджер (Product Manager): визначає стратегію розвитку продукту, досліджує ринок і потреби користувачів, приймає рішення, які функції розробляти.

Бізнес-аналітик (BA): досліджує бізнес-процеси клієнта, збирає та формалізує вимоги до програмного забезпечення.

Scrum Master: допомагає команді дотримуватися Agile-принципів, фасилітує зустрічі, усуває перешкоди.

5. **Тестування (Quality Assurance):**

QA Engineer (Тестувальник): шукає помилки в ПЗ, створює тестові сценарії, автоматизує тести. Поділяється на Manual QA (ручне тестування) та Automation QA (автоматизоване).

6. **Дизайн (Design):**

UI/UX Designer: проектує зручний, естетичний та інтуїтивно зрозумілий інтерфейс. UX (User Experience) – досліджує поведінку користувачів, створює прототипи. UI (User Interface) – відповідає за візуальне оформлення.

7. **Безпека (Cybersecurity):**

Security Analyst / Penetration Tester (Етичний хакер): шукає вразливості в системах, проводить тестування на проникнення, захищає дані від атак.

8. **Підтримка та навчання (Support & Education):**

Technical Support Engineer: допомагає користувачам вирішувати проблеми з ПЗ.

Technical Writer: створює технічну документацію, інструкції, help-системи.

2.2. Hard Skills та Soft Skills для IT-фахівців

Hard Skills (технічні навички): конкретні, вимірювані знання, необхідні для виконання професійних обов'язків. Вони залежать від обраної професії (знання мов програмування, фреймворків, інструментів, баз даних).

Soft Skills (універсальні навички): особистісні якості та соціальні навички, які важливі для будь-якої професії:

- Комунікація: вміння чітко висловлювати думки, пояснювати складні речі просто, домовлятися з колегами та замовниками.
- Командна робота: здатність ефективно працювати в групі, ділитися знаннями, допомагати іншим.
- Критичне мислення та вирішення проблем: вміння аналізувати ситуацію, знаходити причини проблем і пропонувати ефективні рішення.
- Тайм-менеджмент та самоорганізація: вміння планувати свій час, визначати пріоритети, не відволікатися.
- Адаптивність та бажання вчитися: готовність швидко опановувати нові технології та змінювати підходи.
- Емоційний інтелект: розуміння власних емоцій та емоцій інших, емпатія.

2.3. Типовий кар'єрний шлях в IT

Більшість IT-професій мають схожі рівні розвитку:

Стажер (Intern/Trainee): щойно після курсів або університету, працює під наглядом досвідченого колеги, виконує прості завдання.

Молодший спеціаліст (Junior): має базові знання, може виконувати прості задачі самостійно, потребує код-рев'ю та допомоги в складних питаннях.

Спеціаліст (Middle): працює самостійно над більшістю задач, може навчати джуніорів, бере участь у проєктуванні.

Старший спеціаліст (Senior): має глибокі знання, вирішує складні технічні проблеми, впливає на архітектуру, менторить інших, приймає ключові рішення.

Провідний спеціаліст / Архітектор (Lead / Architect): відповідає за технічний напрямок розвитку продукту або цілої системи, координує роботу кількох команд.

Звідси можна рухатися як вглиб (технічна кар'єра), так і в управління (Team Lead, Project Manager, CTO).

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп'ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постерів).
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - текстовий процесор (Google Docs, Microsoft Word Online).
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Презентація ІТ-професії

Мета завдання: навчитися систематизувати інформацію про конкретну ІТ-професію, аналізувати вимоги ринку та представляти результати в структурованому вигляді.

Теоретичне підґрунтя

Щоб обрати професію, недостатньо знати її назву. Потрібно розуміти, чим щодня займається фахівець, які інструменти використовує, які знання потрібні та куди він може вирости. Ваше завдання – стати експертом з однієї професії.

Детальна інструкція виконання

Етап 1. Формування груп та вибір професії (5 хвилин).

Об'єднайтесь у групи по 3-4 особи.

Оберіть **одну ІТ-професію** для глибокого аналізу. *Приклади:*

- *Розробка:* Frontend Developer, Backend Developer (Python/Java/C#), Mobile Developer (iOS/Android), Game Developer.
- *Дані:* Data Analyst, Data Scientist, Data Engineer.
- *Інфраструктура:* DevOps Engineer, System Administrator.
- *Управління:* Project Manager, Product Manager, Business Analyst.
- *Тестування:* QA Manual, QA Automation.
- *Дизайн:* UI/UX Designer.
- *Безпека:* Cybersecurity Analyst, Penetration Tester.

Переконайтеся, що професії не повторюються між групами.

Етап 2. Збір інформації (10 хвилин).

Використовуйте наступні джерела для швидкого дослідження:

- Сайти з вакансіями: djinni.co, work.ua, robota.ua, LinkedIn (проаналізуйте 2-3 вакансії, випишіть вимоги).
- Освітні платформи: Coursera (статті про кар'єру), freeCodeCamp (розділ "News").
- Професійні блоги: DOU.ua, Medium (за тегами).

Знайдіть відповіді на питання та заповніть таблицю:

Критерій	**Інформація**
Назва професії	
Опис: чим займається? (2-3 речення)	
Основні обов'язки (3-5 пунктів)	
Ключові технічні навички (Hard Skills) (3-5 пунктів)	
Ключові універсальні навички (Soft Skills) (3-5 пунктів)	
Інструменти / Технології (мови, фреймворки, ПЗ)	
Типовий кар'єрний шлях (Junior -> Middle -> Senior -> ...)	
Середня зарплата (початківець / досвідчений) * (орієнтовно)	

Етап 3. Створення презентації/постеру (5 хвилин).

Створіть 2-3 слайди або постер, які яскраво та структуровано представлять професію.

Структура:

- *Слайд 1.* Назва професії, візуальне зображення (логотипи технологій), короткий опис.
- *Слайд 2.* Ключові обов'язки та необхідні навички (можна у вигляді списку або "хмари слів").
- *Слайд 3.* Кар'єрний шлях (схема "Сходи") та цікаві факти про професію.

Висновок: Чому ця професія важлива та перспективна?

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група презентує свою професію (2-3 хвилини).

Загальна дискусія:

- Яка професія здалася вам найбільш зрозумілою? Найбільш складною?
- Які навички (soft skills) є спільними для всіх професій?
- Після почутого, чи з'явилася у вас професія, яка зацікавила найбільше? Чому?

Критерії оцінювання (Завдання 1):

Повнота дослідження: зібрано інформацію за всіма критеріями.

Актуальність: навички та інструменти відповідають сучасним вимогам ринку (на основі аналізу вакансій).

Структурованість: інформація подана логічно, легко сприймається.

Командна робота.

Якість презентації.

Завдання 2. Рольова гра «День із життя ІТ-фахівця»

Мета завдання: розвинути навички комунікації, вміння ставити запитання та імпровізувати, а також "приміряти" на себе роль ІТ-фахівця.

Теоретичне підґрунтя

Найкращий спосіб зрозуміти професію – поговорити з тим, хто в ній працює. Це завдання – симуляція такого "інформаційного інтерв'ю".

Детальна інструкція виконання

Етап 1. Підготовка до інтерв'ю (5 хвилин).

Об'єднайтесь у пари. Розподіліть ролі: «Інтерв'юер» та «ІТ-фахівець».

«Фахівець» обирає професію, від імені якої він говоритиме (бажано з тих, що розглядали в Завданні 1).

«Інтерв'юер» готує 4-6 запитань. Вони мають бути відкритими (починатися з "Як", "Чому", "Розкажи"), щоб отримати розгорнуту відповідь. *Приклади запитань:*

- Розкажи, як починається твій типовий робочий день?
- Які 3 основні задачі ти вирішуєш зараз?

- З ким із колег (інших професій) ти найчастіше спілкуєшся? Про що?
- Яка найскладніша проблема була у тебе на цьому тижні? Як ти її вирішував?
- Які інструменти ти використовуєш щодня? (назви програми, сайти, бібліотеки)
- Якби в університеті був курс "Як стати [назва професії]", які 3 теми мали б бути обов'язковими?
- Що тобі найбільше подобається у твоїй роботі, а що – найменше?

Етап 2. Проведення інтерв'ю (7-8 хвилин).

«Інтерв'юер» ставить запитання. «Фахівець» відповідає, спираючись на інформацію, отриману під час першого завдання, або на власні знання/уяву. Відповіді мають бути правдоподібними.

«Інтерв'юер» робить короткі нотатки.

Етап 3. Фіксація підсумків (5 хвилин).

Разом обговоріть почуте.

Запишіть короткий підсумок (5-7 речень) у спільному документі або на аркуші, вказавши:

- Професія: [Назва]
- Основні задачі: (з відповідей фахівця)
- Ключові інструменти: (з відповідей фахівця)
- Найважливіші навички (на думку фахівця):
- Цікавий факт / Враження:

Етап 4. Презентація та обговорення (10-12 хвилин).

За бажанням, 2-3 пари зачитують свої підсумки.

Загальне обговорення:

- Які професії здаються найбільш динамічними, де щось постійно змінюється?
- Чи здивувала вас інформація про те, з ким доводиться спілкуватися фахівцям різних професій?
- Якби ви могли взяти інтерв'ю у реального фахівця, кого б ви обрали і що б запитали?

Критерії оцінювання (Завдання 2):

Якість запитань: питання різноманітні, стосуються різних аспектів роботи.

Правдоподібність відповідей: відповіді не суперечать логіці та загальним знанням про професію.

Чіткість підсумку: записи структуровані та інформативні.

Активність в обговоренні.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Дослідження ІТ-професії та кар'єрного шляху

Мета завдання: поглибити знання про обрану професію, навчитися знаходити актуальну інформацію на ринку праці, планувати власний розвиток.

Вступ. Це останнє самостійне завдання курсу має стати для вас відправною точкою у плануванні вашої кар'єри. Ви оберете одну професію і складете для себе "дорожню карту" входження в неї.

Детальна інструкція виконання

Крок 1. Вибір професії.

Оберіть **одну ІТ-професію**, яка вас найбільше зацікавила. Якщо ще не визначилися, оберіть ту, про яку хочете дізнатися більше.

Крок 2. Поглиблене дослідження.

Використовуйте різноманітні джерела, щоб скласти максимально повну картину:

Аналіз вакансій: знайдіть 3-5 актуальних вакансій для початківців (Junior/Trainee) на djinni.co, work.ua. Випишіть вимоги, що повторюються (must have), та бажані навички (nice to have).

Професійні спільноти: пошукайте інформацію про професію на DOU.ua (статті, огляди зарплат, інтерв'ю).

Освітні платформи: подивіться програми курсів для початківців з цієї професії на Prometheus, Coursera, Udemy. Це дасть розуміння, чому треба вчитися.

YouTube: подивіться відео "Day in the life", "How I became a ...", "Roadmap for ..." (дорожня карта).

Крок 3. Створення звіту.

Створіть звіт (300-400 слів) у Google Docs, Microsoft Word або ментальну карту, який містить такі розділи:

1. Опис професії:
   - Назва, коротке визначення.

- Основні обов’язки (3-5 пунктів).

2. Необхідні навички:

Технічні навички (Hard Skills). Детальний список. Розділіть на "Обов’язкові для старту" (на основі аналізу вакансій) та "Для поглибленого вивчення".

Універсальні навички (Soft Skills). Які з них є критичними для цієї професії? Чому?

3. Кар’єрний шлях:

Опишіть типову траєкторію: Trainee/Junior -> Middle -> Senior -> (Tech Lead / Architect / Manager).

Які можливі напрямки зростання? (наприклад, Backend-розробник може стати Solution Architect або Team Lead).

4. Ресурси для підготовки:

Наведіть конкретні ресурси з посиланнями, які допоможуть опанувати цю професію з нуля.

Приклади: "Курс ‘Python для початківців’ на Prometheus", "Книга ‘Clean Code’ Роберта Мартіна", "YouTube-канал ‘Тімоті’ для фронтенд-розробників", "Спільнота DOU ua".

Обов’язково додайте мінімум 3 активні посилання на різні типи ресурсів (курс, стаття, відео, книга).

5. Мої наступні кроки (Рефлексія):

Напишіть 3-5 речень про те, що вас приваблює в цій професії, а що, можливо, лякає або здається складним.

Що ви плануєте зробити найближчим часом (вже під час навчання в університеті), щоб наблизитися до цієї професії?

Крок 4. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 3-4, з гіперпосиланнями).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина та актуальність дослідження: інформація базується на реальних вимогах ринку (аналіз вакансій), а не на абстрактних уявленнях.

Практична цінність: наведені ресурси справді корисні та доступні.

Структурованість: чітке слідування плану, логічний виклад.

Рефлексія: наявність особистих висновків та планів.

Оформлення: чистота, грамотність, коректні посилання.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Які основні категорії IT-професій ви можете назвати? Наведіть по 2-3 приклади до кожної.

2. У чому різниця між Frontend, Backend та Fullstack розробником?

3. Чим відрізняється робота Data Analyst від Data Scientist?

4. Які обов'язки Project Manager? A Product Manager?

5. Назвіть 5 ключових soft skills, важливих для будь-якого IT-фахівця. Чому вони важливі?

6. Опишіть типовий кар'єрний шлях від Junior до Senior.

7. Як ви можете використовувати сайти з вакансіями (djinni, work.ua) для дослідження професій?

8. Яка IT-професія на даний момент здається вам найбільш привабливою? Чому? (чесна відповідь, без оцінювання).

9. Які ресурси (сайти, книги, відео) ви плануєте використовувати для подальшого вивчення обраної професії?

10. Як ви вважаєте, чи може людина змінити IT-професію в процесі кар'єри (наприклад, з тестувальника стати розробником)? Чому так?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Ресурси для дослідження професій:

1. DOU.ua – Ринок праці: https://dou.ua/ – Найкраще українське джерело. Огляди зарплат, статті "Як стати ...", профілі компаній.

2. djinni.co: https://djinni.co/ – Сайт пошуку роботи. Ідеальний для аналізу вимог до вакансій.

3. Coursera – Articles/Careers: https://www.coursera.org/articles – Велика кількість статей про різні IT-професії англійською та українською.

4. freeCodeCamp – News: https://www.freecodecamp.org/news/ – Тисячі статей про технології, кар'єру, навчання.

5. Glassdoor: https://www.glassdoor.com/ – Відгуки про компанії та зарплати (англ.).

Кар'єрні дорожні карти (Roadmaps):

6. Roadmap.sh: https://roadmap.sh/ – Найпопулярніший ресурс з дорожніми картами для різних IT-професій (Frontend, Backend, DevOps, QA тощо). Вкрай рекомендовано.

7. YouTube-канали: Існує безліч каналів, присвячених конкретним професіям. Шукайте за запитами "Roadmap for Frontend Developer 2025", "How to become a Data Analyst" тощо.

Книги (про кар'єру):

8. Гейл Лаакманн Мак-Доуелл. "Кар'єра програміста. Як влаштуватися на роботу в Google, Microsoft або іншу велику компанію". (Про підготовку до співбесід).

9. Мартін Клеппман. "Ви не знаєте JS" (серія книг для фронтенд-розробників).

10. Вес Маккінні. "Python for Data Analysis" (для тих, хто хоче в аналітику даних).

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Не поспішайте з вибором.** Ви тільки на початку шляху. Протягом перших курсів університету ви спробуєте різні дисципліни, можливо, напишете кілька проєктів. Цей досвід допоможе вам зробити усвідомлений вибір.

2. **Спробуйте себе в різних ролях.** Беріть участь у студентських проєктах, хакатонах. Спробуйте бути не лише розробником, а й, наприклад, презентувати проєкт (як PM) або малювати дизайн. Так ви на практиці зрозумієте, що вам ближче.

3. **Розвивайте soft skills паралельно з hard skills.** Комунікабельність, відповідальність і вміння працювати в команді цінуються роботодавцями не менше, ніж знання мов програмування. Беріть участь у дискусіях, виступайте з доповідями, заводьте знайомства.

4. **Створіть власне портфоліо.** GitHub з вашими навчальними проєктами, статті в блозі, участь у відкритому коді (open source) – це найкраще підтвердження ваших навичок для майбутнього роботодавця.

5. **Пам'ятайте: навчання ніколи не закінчується.** IT – це сфера, де технології змінюються дуже швидко. Готовність постійно вчитися нового – це не просто перевага, це необхідність. Але це й робить нашу роботу неймовірно цікавою!

ПРАКТИЧНЕ ЗАНЯТТЯ 11
М'ЯКІ НАВИЧКИ (SOFT SKILLS) ДЛЯ ІТ-ФАХІВЦІВ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників розуміння важливості м'яких навичок (soft skills) для успішної кар'єри в ІТ, усвідомлення їхньої ролі в командній роботі та професійному розвитку, а також отримання первинного практичного досвіду їх застосування.

Основні завдання заняття:

1. Ознайомити з ключовими м'якими навичками, необхідними ІТ-фахівцю: комунікація, командна робота, вирішення проблем, тайм-менеджмент, критичне мислення, емоційний інтелект, лідерство, адаптивність.

2. Дослідити, чому технічних знань недостатньо для успішної кар'єри і як soft skills впливають на ефективність роботи та задоволеність нею.

3. Розвинути практичні навички:

– *комунікації:* чітко висловлювати думки, слухати інших, аргументувати свою позицію.

– *командної роботи:* співпрацювати, розподіляти завдання, вирішувати конфлікти.

– *тайм-менеджменту:* планувати роботу, визначати пріоритети, використовувати ефективні техніки.

4. Розвинути ключові навички самоаналізу та рефлексії щодо власних сильних сторін та зон для розвитку.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Що таке soft skills і чому вони важливі в ІТ?

Soft skills (м'які навички) – це особистісні якості, соціальні навички та емоційний інтелект, які дозволяють людині ефективно взаємодіяти з іншими, вирішувати проблеми, адаптуватися до змін та керувати власним життям. На відміну від hard skills (технічних навичок), вони не залежать від конкретної технології та є універсальними.

Чому вони критично важливі для ІТ-фахівця?

Робота в командах: сучасна розробка ПЗ – це майже завжди командна робота. Потрібно вміти порозумітися з колегами, менеджерами, замовниками.

Спілкування з замовниками: потрібно вміти зрозуміти потреби замовника, пояснити технічні обмеження простою мовою, узгодити вимоги.

Вирішення конфліктів: у будь-якому колективі виникають розбіжності. Вміння конструктивно вирішувати конфлікти – запорука здорової атмосфери.

Кар'єрне зростання: для переходу на вищі позиції (Team Lead, Architect, Manager) технічних знань недостатньо. Потрібні лідерські якості, вміння керувати людьми та проєктами.

Ефективність: хороший тайм-менеджмент та самоорганізація дозволяють виконувати більше роботи за менший час і з меншим стресом.

2.2. Ключові soft skills для IT-фахівця

1. **Комунікація (Communication):**

Усна комунікація: вміння чітко і зрозуміло висловлювати свої думки, виступати перед аудиторією, брати участь у нарадах.

Писемна комунікація: вміння грамотно писати листи, технічну документацію, коментарі в коді, повідомлення в чатах.

Активне слухання: вміння не просто чути, а розуміти співрозмовника, ставити уточнюючі запитання, проявляти емпатію.

Вміння пояснювати складне простою мовою: це особливо важливо при спілкуванні з нетехнічними фахівцями (замовниками, маркетологами).

2. **Командна робота (Teamwork):**

Співпраця: готовність ділитися знаннями, допомагати колегам, працювати над спільним результатом.

Повага до думки інших: вміння визнавати, що ваша думка не є єдино правильною, і враховувати ідеї колег.

Відповідальність: розуміння, що ваш внесок впливає на успіх всієї команди.

3. **Вирішення проблем (Problem-Solving):**

Аналіз проблеми: вміння розібрати складну проблему на складові, знайти її першопричину.

Креативність: здатність знаходити нестандартні рішення.

Прийняття рішень: вміння обирати найкращий варіант з кількох альтернатив, зважуючи ризики.

4. **Тайм-менеджмент та самоорганізація (Time Management):**

Планування: вміння ставити цілі, розбивати їх на задачі, визначати пріоритети.

Уникнення прокрастинації: здатність починати роботу вчасно, не відволікатися.

Робота з дедлайнами: вміння реалістично оцінювати час на виконання задач та вкладатися в терміни.

5. **Критичне мислення (Critical Thinking):**

Аналіз інформації: вміння оцінювати достовірність джерел, відрізняти факти від припущень.

Постановка запитань: здатність ставити правильні запитання, щоб дістатися до суті.

Оцінка ризиків: вміння передбачати можливі негативні наслідки рішень.

6. **Емоційний інтелект (Emotional Intelligence, EQ):**

Самоусвідомлення: розуміння власних емоцій, сильних і слабких сторін.

Саморегуляція: вміння контролювати імпульси, керувати стресом.

Емпатія: розуміння емоцій інших людей, здатність поставити себе на їхнє місце.

7. **Адаптивність (Adaptability):**

Гнучкість: готовність змінювати плани, коли змінюються обставини або вимоги.

Бажання вчитися: відкритість до нових знань, технологій, підходів.

Стійкість до стресу: здатність зберігати продуктивність у складних ситуаціях.

8. **Лідерство (Leadership):**

Ініціативність: готовність брати на себе відповідальність, пропонувати ідеї.

Мотивація: вміння надихати інших, підтримувати командний дух.

Наставництво (менторинг): готовність ділитися досвідом, допомагати менш досвідченим колегам.

2.3. Техніки тайм-менеджменту

Метод Pomodoro. Робота розбивається на інтервали (зазвичай 25 хвилин), розділені короткими перервами (5 хвилин). Після 4 таких циклів робиться довга перерва (15-30 хвилин). Допомагає зберігати концентрацію та уникати перевтоми.

Матриця Ейзенхауера. Метод пріоритезації задач на основі їх терміновості та важливості. Задачі діляться на 4 категорії:

1. Важливі і термінові: робити негайно.
2. Важливі, але не термінові: планувати на потім.
3. Термінові, але не важливі: делегувати.
4. Не важливі і не термінові: викинути.

Принцип Парето (80/20). 20% зусиль дають 80% результату. Важливо визначити ці 20% ключових задач і зосередитися на них насамперед.

GTD (Getting Things Done): система, яка передбачає фіксацію всіх задач у зовнішній системі (нотатник, додаток), їх регулярний перегляд та організацію.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп'ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, фломастери/стікери (для візуалізації розкладів).
- програмне забезпечення (на вибір):
  - текстовий процесор (Google Docs, Microsoft Word Online) – для звітів.
  - табличний процесор (Google Sheets, Microsoft Excel Online) – для створення розкладів.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.
  - таймер (можна використати на телефоні) – для симуляції Pomodoro.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Рольова гра «Командна робота в ІТ-проєкті»

Мета завдання: на практиці відчути важливість комунікації, активного слухання та вирішення конфліктів у командній роботі під час обговорення проєкту.

Теоретичне підґрунтя

У реальних ІТ-командах щодня відбуваються обговорення, дискусії та суперечки. Вміння домовитися, почути іншого та знайти компроміс – це ключ до успішного проєкту. Сьогодні ми змоделюємо таку ситуацію.

Детальна інструкція виконання

Етап 1. Формування команд та визначення проєкту (5 хвилин).

Об'єднайтесь у групи по 4-5 осіб.

Оберіть уявний проєкт. *Приклади:*

- Розробка вебсайту для студентського клубу.
- Створення мобільного додатку для пошуку книг у бібліотеці.
- Розробка чат-бота для відповідей на часті питання абітурієнтів.

Призначте ролі. Кожен учасник має чітко розуміти свою роль та обов'язки:

- Менеджер проєкту: відповідає за те, щоб обговорення не виходило за рамки, слідкує за часом, фіксує рішення.
- Розробник (Backend): відповідає за серверну логіку, бази даних. Його хвилює технічна реалізація.
- Розробник (Frontend): відповідає за те, що бачить користувач. Його хвилює зручність та краса інтерфейсу.
- Дизайнер: відповідає за візуальний стиль, кольори, розташування елементів.
- Тестувальник (QA): відповідає за якість. Він думає про те, як користувач може "зламати" програму.

Етап 2. Симуляція командної роботи (10 хвилин).

Ваше завдання – протягом 10 хвилин обговорити ключові функції вашого проєкту та ухвалити рішення щодо дизайну або технічної реалізації.

Симулюйте ситуацію:

- Кожен по черзі пропонує ідею з точки зору своєї ролі. (Наприклад, дизайнер пропонує незвичне кольорове рішення, яке фронтенд-розробник вважає технічно складним).
- Обговоріть ідеї. Той, хто не погоджується, має аргументувати свою позицію, посилаючись на свою роль (наприклад, QA: "Це буде важко протестувати", Менеджер: "Це збільшить терміни").

Створіть уявний конфлікт. Наприклад, думки щодо головної функції розділилися. Завдання – знайти компромісне рішення, яке влаштує всіх.

- Спостерігайте, як ви спілкуєтеся: чи перебиваєте одне одного, чи вмієте слухати, чи намагаєтеся зрозуміти позицію іншого.

Етап 3. Підготовка звіту (5 хвилин).

Після симуляції запишіть короткий звіт (5-7 речень), який містить:

- Назву проєкту та ролі учасників.
- Які саме soft skills ви використовували під час обговорення? (Наприклад, "активне слухання", "аргументація", "компроміс", "врегулювання конфлікту").
- Як ці навички допомогли (або завадили) вам дійти спільного рішення?
- Що ви могли б зробити краще?

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група коротко (1-2 хвилини) презентує свій звіт.

Загальна дискусія:

- Які soft skills виявилися найважливішими під час цієї вправи?

- Чи легко було домовитися? Що було найскладнішим?
- Чи змінилося ваше розуміння важливості комунікації після цієї гри?

Критерії оцінювання (Завдання 1):

Активність: участь всіх членів групи в обговоренні.

Демонстрація навичок: видимі спроби використовувати активне слухання, аргументацію, пошук компромісу.

Якість звіту: чіткий опис використаних навичок та їх впливу на результат.

Рефлексія: усвідомлення власних сильних сторін та зон для розвитку.

Завдання 2. Вправи на розвиток тайм-менеджменту

Мета завдання: ознайомитися з практичними техніками планування часу та навчитися застосовувати їх для планування роботи над ІТ-завданням.

Теоретичне підґрунтя

ІТ-фахівці часто працюють над складними задачами, які потребують концентрації. Техніки тайм-менеджменту допомагають не потонути в деталях, вчасно здавати роботу та зберігати work-life balance.

Детальна інструкція виконання

Етап 1. Вибір завдання та ознайомлення з техніками (5 хвилин).

Об’єднайтесь у пари.

Оберіть уявне ІТ-завдання. *Приклади:*

- Написати функцію для калькулятора на Python.
- Зверстати головну сторінку сайту-портфоліо (HTML/CSS).
- Написати тест-кейси для форми реєстрації.
- Підготувати презентацію для захисту курсової роботи.

Швидко перегляньте інформацію про техніки тайм-менеджменту на MindTools або пригадайте з теоретичної частини. Оберіть одну техніку для використання: Pomodoro, Матриця Ейзенхауера або простий список справ з пріоритетами.

Етап 2. Складання розкладу (10 хвилин).

Уявіть, що на виконання цього завдання у вас є один тиждень.

Розбийте головне завдання на підзадачі (декомпозиція). *Наприклад, для функції калькулятора:*

- Дослідити, як реалізувати функцію (1 год).
- Написати код функції (2 год).
- Написати модульні тести (1 год).
- Протестувати на різних вхідних даних (1 год).
- Виправити знайдені помилки (1 год).

Складіть розклад на тиждень, використовуючи обрану техніку.

- Якщо Pomodoro: вкажіть, скільки "помідорів" (25-хвилинних сесій) ви плануєте на кожен день.
- Якщо Матриця Ейзенхауера: розподіліть підзадачі по квадрантах і визначте, що робити в першу чергу.
- Якщо простий список: створіть список на кожен день, проранжувавши завдання за важливістю (А, Б, В).

Запишіть розклад на папері або в Google Docs/Sheets.

Етап 3. Підготовка пояснення (5 хвилин).

Запишіть коротке пояснення (5-7 речень) до вашого розкладу:

- Яке завдання ви обрали?
- Яку техніку тайм-менеджменту використали і чому?
- Як ця техніка допомагає бути більш ефективним? Як вона допомагає в ІТ-роботі?

Етап 4. Презентація та обговорення (10 хвилин).

За бажанням, 2-3 пари презентують свої розклади та пояснення.

Загальна дискусія:

- Яка техніка здалася вам найбільш корисною? Чому?
- Як тайм-менеджмент впливає на рівень стресу?
- Чи плануєте ви використовувати ці техніки в навчанні?

Критерії оцінювання (Завдання 2):

Чіткість розкладу: завдання розбито на підзадачі, вказано часові рамки.

Доречність техніки: обрана техніка застосована коректно.

Обґрунтованість: пояснення демонструє розуміння, як техніка допомагає в роботі.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Аналіз м’яких навичок для ІТ-фахівця

Мета завдання: поглибити знання про одну з soft skills, зрозуміти її роль в ІТ-індустрії та скласти власний план її розвитку.

Вступ. Технічні навички можна отримати на курсах, але м’які навички потребують свідомого та тривалого розвитку. Це дослідження допоможе вам обрати одну навичку для фокусу та спланувати, як її покращувати.

Детальна інструкція виконання

Крок 1. Вибір м’якої навички.

Оберіть **одну** м’яку навичку, яка, на вашу думку, є найважливішою для вас особисто або яку ви хотіли б розвинути в першу чергу.

Варіанти: Комунікація, Командна робота, Вирішення проблем, Тайм-менеджмент, Критичне мислення, Емоційний інтелект, Адаптивність, Лідерство.

Крок 2. Поглиблене дослідження.

Знайдіть інформацію про обрану навичку, використовуючи мінімум 2-3 надійні джерела (статті на MindTools, Coursera, YouTube-лекції, книги).

Дайте відповіді на питання:

1. Визначення: Що таке ця навичка? З яких компонентів вона складається? (Наприклад, комунікація включає усне мовлення, письмо, активне слухання).

2. Важливість в ІТ: Чому саме вона критична для ІТ-фахівця? Наведіть 2-3 конкретні приклади з роботи, де вона необхідна (наприклад, "Щоб пояснити клієнту, чому його ідея технічно складна, потрібна комунікація").

3. Наслідки відсутності: Що може статися, якщо ІТ-фахівець не володіє цією навичкою? (Наприклад, поганий тайм-менеджмент -> зірвані дедлайни -> стрес у команді).

4. Способи розвитку: Знайдіть 3-4 конкретні, практичні способи розвитку цієї навички.

- *для комунікації:* виступати з доповідями, брати участь у дискусіях, вести блог, практикувати активне слухання.
- *для тайм-менеджменту:* використовувати Pomodoro, планувати тиждень у календарі, вести список справ.
- *для лідерства:* взяти на себе відповідальність за студентський проєкт, стати старостою, допомагати молодшим студентам.

Крок 3. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (300-400 слів).

Структура:

1. Вступ. Назва навички, чому ви обрали саме її.

2. Опис навички: визначення, складові.

3. Роль в ІТ: чому вона важлива, приклади застосування, наслідки відсутності.

4. План розвитку: 3-4 конкретні дії, які ви плануєте зробити найближчим часом для розвитку цієї навички.

5. Ресурси: список корисних ресурсів (книги, статті, курси, відео) з активними посиланнями (мінімум 2).

Варіант Б: Ментальна карта.

У центрі – назва навички.

Перший рівень гілок: "Визначення", "Чому важливо в ІТ", "Наслідки відсутності", "Як розвивати", "Ресурси".

На гілці "Як розвивати" – конкретні дії.

На гілці "Ресурси" – назви та іконки з посиланнями.

Крок 4. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 2, з гіперпосиланнями).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина розуміння: робота демонструє не поверхове, а глибоке розуміння обраної навички.

Конкретність: приклади застосування в ІТ є реалістичними, а план розвитку – практичним.

Практична цінність: наведені ресурси справді корисні та доступні.

Структурованість: чітке слідування плану, логічний виклад.

Оформлення: чистота, грамотність, коректні посилання.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Що таке soft skills? Чому вони важливі для IT-фахівця не менше, ніж hard skills?
2. Назвіть 5 ключових soft skills, необхідних для успішної роботи в IT-команді.
3. Які навички входять до поняття "комунікація"? Чому активне слухання таке важливе?
4. Опишіть метод Pomodoro. Як він допомагає боротися з прокрастинацією?
5. Поясніть, як працює Матриця Ейзенхауера. Наведіть приклад задачі для кожного квадранта.
6. Чому емпатія важлива для розробника, який більшу частину часу працює з кодом?
7. Як ви розумієте термін "критичне мислення"? Як воно допомагає при вирішенні складних технічних проблем?
8. Чому IT-фахівець повинен бути адаптивним і постійно вчитися?
9. Як ви можете розвивати свої лідерські якості, ще будучи студентом?
10. Яку soft skill ви вважаєте своєю найсильнішою, а яку хотіли б розвинути в першу чергу? Чому?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Ресурси про soft skills:

1. MindTools. https://www.mindtools.com/ – Один з найкращих ресурсів для розвитку кар’єрних навичок. Величезна бібліотека статей про комунікацію, лідерство, тайм-менеджмент (англ.).
2. Coursera – Soft Skills Articles. https://www.coursera.org/articles – Багато статей англійською та українською про різні soft skills в контексті IT.
3. SkillsYouNeed. https://www.skillsyouneed.com/ – Ресурс, присвячений розвитку особистих та професійних навичок (англ.).
4. DOU.ua – Блоги. Багато статей від українських IT-фахівців про soft skills, командну роботу, кар’єру.

Книги

5. Дейл Карнегі. "Як здобувати друзів і впливати на людей". Класика комунікації.

6. Стівен Кові. "7 звичок надзвичайно ефективних людей". Книга про особисту ефективність, включаючи тайм-менеджмент та пріоритезацію.

7. Деніел Гоулман. "Емоційний інтелект". Фундаментальна праця про EQ.

8. Керрі Паттерсон та ін. "Критичні розмови". Як спілкуватися, коли ставки високі.

9. Джейсон Фрайд, Девід Гайнемаєр Генссон. "Remote: Офіс не обов'язковий". Про роботу в розподілених командах та комунікацію.

Відео

10. TED Talks. Безліч виступів на теми комунікації, лідерства, щастя, продуктивності. Шукайте за тегами "communication", "leadership", "productivity".

11. YouTube-канали. "The School of Life", "Charisma on Command", "Practical Psychology".

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Soft skills можна і потрібно розвивати.** Вони не є вродженими. Якщо вам важко виступати публічно або ви відчуваєте, що погано плануєте час, це не вирок. Це просто навички, які потребують тренування.

2. **Використовуйте університетське життя як тренувальний майданчик.** Беріть участь у групових проєктах, виступайте з доповідями на семінарах, організовуйте заходи. Це безцінний досвід розвитку soft skills у безпечному середовищі.

3. **Отримуйте зворотний зв'язок.** Питайте друзів, викладачів, колег по проєкту, як вони оцінюють ваші комунікативні навички, вашу надійність, вашу здатність працювати в команді. Сприймайте це як інформацію для зростання, а не як критику.

4. **Ведіть щоденник успіхів.** Фіксуйте ситуації, де вам вдалося успішно використати soft skills (наприклад, "Сьогодні я зміг переконати команду обрати моє технічне рішення"). Це підвищує самооцінку та допомагає бачити прогрес.

5. **Пам'ятайте: над hard skills вас можуть замінити, а над soft skills – ні.** ШІ вчиться писати код, але він (поки що) не вміє будувати довірливі стосунки з клієнтом або мотивувати команду. Ваші м'які навички – це ваша унікальна конкурентна перевага.

ПРАКТИЧНЕ ЗАНЯТТЯ 12
СУЧАСНІ ТРЕНДИ В ІТ-ГАЛУЗІ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників цілісного уявлення про ключові тенденції розвитку сучасної ІТ-індустрії, розуміння їхнього впливу на технології, бізнес і суспільство, а також розвиток навичок аналізу та критичного осмислення технологічних трендів.

Основні завдання заняття:

1. Ознайомити з основними сучасними трендами в ІТ: штучний інтелект (AI) та машинне навчання (ML), хмарні технології (Cloud Computing), Інтернет речей (IoT), кібербезпека (Cybersecurity), блокчейн (Blockchain), квантові обчислення (Quantum Computing), метавсесвіт (Metaverse), Edge Computing, великі дані (Big Data) тощо.

2. Дослідити практичне застосування цих трендів у різних сферах (медицина, освіта, фінанси, розваги, транспорт) та їхній вплив на ринок праці.

3. Розвинути навички критичного аналізу технологій, оцінки їхніх переваг та потенційних ризиків.

4. Розвинути ключові навички:

- *аналітичне мислення:* виявлення суттєвих характеристик тренду, прогнозування його розвитку.
- *критичне мислення:* оцінка не лише переваг, але й ризиків, етичних аспектів.
- *комунікація:* участь у дебатах, аргументація власної позиції.
- *командна робота:* спільне дослідження та підготовка презентації.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Що таке технологічний тренд?

Технологічний тренд – це загальний напрямок розвитку технологій, який набуває значної популярності та впливу на ринку. Визначати тренди допомагають аналітичні компанії, такі як Gartner (публікує "Hype Cycle" – цикл зрілості технологій), Forrester, McKinsey. Розуміння трендів допомагає ІТ-фахівцям обирати технології для вивчення, компаніям – планувати стратегію розвитку.

2.2. Огляд ключових IT-трендів 2020-х років

1. **Штучний інтелект (Artificial Intelligence, AI) та Машинне навчання (Machine Learning, ML):**

Суть: створення систем, здатних виконувати завдання, що зазвичай потребують людського інтелекту (навчання, розпізнавання, прийняття рішень). ML – це підмножина AI, де системи "вчаться" на даних.

Застосування: чат-боти (ChatGPT, Gemini), системи рекомендацій (Netflix, Spotify), розпізнавання зображень та мови, діагностика в медицині, безпілотні автомобілі, аналіз фінансових ризиків.

Вплив: автоматизація рутинних завдань, поява нових професій (AI-тренер, промпт-інженер), загострення етичних питань (упередженість алгоритмів, конфіденційність).

2. **Хмарні технології (Cloud Computing):**

Суть: надання обчислювальних ресурсів (серверів, сховищ, баз даних, ПЗ) через інтернет за моделлю "як послуга". Основні провайдери: AWS, Microsoft Azure, Google Cloud.

Застосування: зберігання даних (Google Drive, Dropbox), хостинг вебсайтів, розробка та тестування ПЗ, аналітика великих даних, резервне копіювання.

Вплив: зниження витрат на IT-інфраструктуру, масштабованість, доступність з будь-якої точки світу, глобальна колаборація.

3. **Інтернет речей (Internet of Things, IoT):**

Суть: мережа фізичних пристроїв (датчиків, гаджетів), оснащених технологіями для збору та обміну даними через інтернет.

Застосування: "розумний дім" (термостати, освітлення, безпека), "розумне місто" (паркування, освітлення вулиць), промисловість (моніторинг обладнання), носимі пристрої (смарт-годинники).

Вплив: підвищення ефективності та зручності, збір величезних обсягів даних, нові виклики щодо безпеки та конфіденційності.

4. **Кібербезпека (Cybersecurity):**

Суть: захист комп’ютерних систем, мереж, програм і даних від цифрових атак, пошкоджень або несанкціонованого доступу.

Застосування: захист корпоративних мереж, безпека вебдодатків, шифрування даних, виявлення вторгнень, антивірусний захист, навчання співробітників.

Вплив: зростання кількості та складності кібератак робить кібербезпеку критично важливою для всіх: від держав до приватних осіб. Постійний дефіцит фахівців.

5. **Блокчейн (Blockchain):**

Суть: технологія розподіленого реєстру, де інформація зберігається у вигляді ланцюжка блоків, кожен з яких містить інформацію про попередній. Забезпечує прозорість, незмінність та децентралізацію даних.

Застосування: криптовалюти (Bitcoin, Ethereum), смарт-контракти, логістика (відстеження постачань), системи голосування, захист інтелектуальної власності.

Вплив: потенційна зміна фінансової системи, підвищення довіри до транзакцій, складність масштабування та енергоспоживання.

6. **Квантові обчислення (Quantum Computing):**

Суть: використання принципів квантової механіки (кубіти) для виконання обчислень, які неможливі для класичних комп'ютерів.

Застосування: розробка нових ліків (моделювання молекул), матеріалознавство, криптографія (злам існуючих шифрів та створення нових), оптимізація складних систем.

Вплив: потенційна революція в науці та технологіях, але технологія поки що на ранній стадії розвитку.

7. **Метавсесвіт (Metaverse):**

Суть: концепція об'єднаного віртуального простору, де люди взаємодіють між собою та з цифровими об'єктами через аватари, використовуючи VR/AR-технології.

Застосування: віртуальні зустрічі та конференції, онлайн-ігри (Fortnite, Roblox), віртуальна освіта, цифрові ринки, віртуальні тури.

Вплив: зміна способів комунікації, розваг та роботи, питання конфіденційності та регулювання.

8. **Периферійні обчислення (Edge Computing):**

Суть: обробка даних не в централізованому хмарному дата-центрі, а ближче до джерела їх генерації (на "периферії" мережі). Це зменшує затримки.

Застосування: безпілотні автомобілі, IoT-пристрої, промислова автоматизація, доповнена реальність (AR).

Вплив: критично важливо для додатків, що потребують миттєвої реакції.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постерів).
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - текстовий процесор (Google Docs, Microsoft Word Online).
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Аналіз IT-тренду

Мета завдання: навчитися аналізувати технологічні тренди, визначати їхні ключові характеристики, сфери застосування та потенційний вплив.

Теоретичне підґрунтя

Знання трендів – це не просто цікавинки. Це основа для прийняття рішень про те, які технології вивчати, в які проєкти інвестувати час, які стартапи мають потенціал. Сьогодні ми станемо аналітиками, які оцінюють перспективні напрямки.

Детальна інструкція виконання

Етап 1. Формування груп та вибір тренду (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть **один сучасний IT-тренд** для аналізу. *Приклади:*

- Штучний інтелект (AI) / Генеративний AI
- Хмарні технології (Cloud Computing)
- Інтернет речей (IoT)
- Кібербезпека (Cybersecurity)
- Блокчейн / Криптовалюти
- Квантові обчислення
- Метавсесвіт (Metaverse) / AR/VR
- Робототехніка
- Біотехнології в IT (BioTech)

Переконайтеся, що тренди не повторюються між групами.

Етап 2. Збір інформації (10 хвилин).

Використовуйте авторитетні джерела для швидкого аналізу:

- Gartner: https://www.gartner.com/en/articles – статті про топ-тренди.
- TechCrunch: https://techcrunch.com/ – новини та аналітика.
- Wired: https://www.wired.com/ – глибокі статті про технології.
- Forbes Technology: https://www.forbes.com/technology/
- DOU.ua: статті про технологічні тренди українською.

Знайдіть відповіді на питання та заповніть таблицю:

Критерій	**Інформація**
Назва тренду	
Короткий опис (що це таке?)	
Ключові технології / концепції	
Приклади застосування (2-3 приклади)	
Вплив на ІТ-галузь (нові професії, інструменти, підходи)	
Вплив на суспільство (як змінює життя людей)	
Перспективи розвитку (що прогнозують експерти на 3-5 років)	
Потенційні ризики / виклики	

Етап 3. Створення презентації (5 хвилин).

Створіть 2-3 слайди або постер, які яскраво та структуровано представлять ваш тренд.

Структура:

- *Слайд 1.* Назва тренду, візуальне зображення, коротке визначення.
- *Слайд 2.* Ключові застосування (з ілюстраціями) та вплив.
- *Слайд 3.* Перспективи та виклики.

Висновок: чому цей тренд важливий для майбутніх ІТ-фахівців?

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група презентує свій тренд (2-3 хвилини).

Загальна дискусія:

- Який тренд видався вам найбільш перспективним? Найбільш небезпечним?
- Які нові професії можуть з’явитися завдяки цим трендам?
- Як ці тренди можуть вплинути на ваше навчання та вибір спеціалізації?

Критерії оцінювання (Завдання 1):

Актуальність: використано інформацію з сучасних, авторитетних джерел.

Глибина аналізу: розглянуто не лише визначення, але й вплив, ризики, перспективи.

Наочність: слайди або постер містять візуальні елементи, інформація структурована.

Командна робота.

Якість презентації.

Завдання 2. Дебати «Переваги та ризики ІТ-тренду»

Мета завдання: розвинути навички критичного мислення, аргументації та публічного виступу, а також навчитися бачити як позитивні, так і негативні сторони технологічного прогресу.

Теоретичне підґрунтя

Будь-яка технологія має як світлу, так і темну сторони. Наприклад, AI може автоматизувати рутинну роботу, але й призвести до втрати робочих місць. Хмарні технології дають гнучкість, але створюють ризики витоку даних. Дебати допомагають побачити цю двоїстість.

Детальна інструкція виконання

Етап 1. Вибір теми та розподіл ролей (5 хвилин).

Об’єднайтесь у пари або невеликі групи по 2-3 особи.

Виберіть один ІТ-тренд, який буде темою дебатів. (Можна взяти той самий, що й у Завданні 1, або інший).

Розподіліть позиції:

- **Команда "За" (Pro):** захищає переваги тренду, показує його користь для суспільства та економіки.
- **Команда "Проти" (Con):** вказує на ризики, недоліки, етичні проблеми, потенційну шкоду.

Етап 2. Підготовка аргументів (8 хвилин).

Протягом 8 хвилин кожна команда готує **3-4 сильні аргументи** на підтримку своєї позиції.

Кожен аргумент має бути підкріплений прикладом або логічним поясненням.

Приклад для тренду "Штучний інтелект":

- **Аргумент "За":** ШІ може аналізувати медичні знімки (МРТ, рентген) швидше та точніше за лікаря, що рятує життя.
- **Аргумент "Проти":** алгоритми ШІ можуть бути упередженими, якщо їх навчали на неякісних даних, що призводить до дискримінації при прийомі на роботу або видачі кредитів.

Етап 3. Проведення дебатів (8 хвилин).

Встановіть регламент (наприклад, по 2 хвилини на виступ кожної команди + 2 хвилини на відповіді на запитання).

- Раунд 1 (Презентація): команда "За" виголошує свої аргументи.
- Раунд 2 (Контраргументи): команда "Проти" виступає зі своїми аргументами та може коротко прокоментувати аргументи опонентів.
- Раунд 3 (Дискусія): команди ставлять одна одній запитання та відповідають на них.

Етап 4. Фіксація підсумків та загальне обговорення (9 хвилин).

Після дебатів кожна пара/група записує короткий підсумок (5-7 речень), де фіксує найсильніші аргументи обох сторін.

Викладач ініціює загальне обговорення:

- Чия аргументація була більш переконливою?
- Як можна знайти баланс між розвитком технології та мінімізацією її ризиків? (наприклад, через закони, етичні кодекси, освіту).
- Чи змінилася ваша думка щодо цього тренду після дебатів?

Критерії оцінювання (Завдання 2):

Якість аргументації: логічність, переконливість, підкріплення прикладами.

Активність: участь усіх членів команди.

Навички комунікації: чіткість мовлення, вміння слухати, коректність.

Якість підсумку: чітка фіксація ключових ідей.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Дослідження впливу ІТ-тренду

Мета завдання: поглибити знання про один із ключових ІТ-трендів, навчитися аналізувати його багатогранний вплив (технологічний, соціальний, економічний) та формувати власну обґрунтовану думку.

Вступ. Технології змінюють світ навколо нас. Розуміння глибинних процесів, що стоять за гучними термінами, допоможе вам не просто бути користувачем, а стати творцем і свідомим учасником цих змін.

Детальна інструкція виконання

Крок 1. Вибір тренду.

Оберіть **один ІТ-тренд**, який вас найбільше зацікавив. Бажано, щоб він відрізнявся від того, який ви досліджували в аудиторії, щоб розширити кругозір.

Крок 2. Поглиблене дослідження.

Використовуйте мінімум 3-4 різноманітні джерела: аналітичні звіти (Gartner, Deloitte), новинні сайти (TechCrunch, The Verge), науково-популярні статті, відео на YouTube. Знайдіть відповіді на питання:

1. Технічна сутність. Як ця технологія працює на базовому рівні? (Спростіть, але збережіть суть).
2. Поточний стан. На якому етапі розвитку вона перебуває? Це вже зріла технологія, чи ще експеримент?
3. Приклади застосування. Знайдіть 3-4 конкретні приклади використання цієї технології в реальному житті (компанії, продукти, стартапи).
4. Вплив на різні сфери:
   - Бізнес: Як змінює ринки, створює нові бізнес-моделі?
   - Ринок праці: Які нові професії з'являються? Які зникають?
   - Освіта: Як впливає на навчання?
   - Медицина / Охорона здоров'я: Як впливає на медицину?
   - Повсякденне життя: Як впливає на звичайних людей?
5. Виклики та ризики. Які є технічні, етичні, правові, соціальні проблеми, пов'язані з цим трендом? (наприклад, упередженість алгоритмів, втрата робочих місць, конфіденційність, безпека).
6. Прогнози. Що експерти прогнозують щодо цього тренду на найближчі 5-10 років?

Крок 3. Створення звіту (формат на вибір).

Варіант А: Текстовий звіт (350-450 слів).

Структура:

1. Вступ. Назва тренду, чому він важливий/цікавий.
2. Основна частина: Відповіді на питання з Кроку 2 (технологія, приклади, вплив, виклики). Розбийте на підпункти.

3. Висновок: Ваша особиста думка. Чи вважаєте ви цей тренд позитивним? Чи хотіли б ви працювати в цій сфері? Чому?

Варіант Б: Ментальна карта.

У центрі – назва тренду.

Перший рівень гілок: "Технологія", "Приклади", "Вплив", "Ризики", "Прогнози".

На гілці "Вплив" можна зробити підгілки: "Бізнес", "Освіта", "Медицина" тощо.

Додавайте іконки, короткі тези, посилання.

Крок 4. Оформлення та здача.

Обов'язково вкажіть наприкінці роботи список використаних джерел (мінімум 3-4, з гіперпосиланнями).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Глибина та широта дослідження: робота охоплює різні аспекти тренду (технічний, соціальний, економічний).

Актуальність: використано сучасні джерела (останні 1-2 роки).

Критичність: наявний аналіз не лише переваг, але й ризиків.

Структурованість: чітке слідування плану, логічний виклад.

Особиста позиція: наявність власного обґрунтованого висновку.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Що таке технологічний тренд? Хто такі аналітики (наприклад, Gartner) і чим вони займаються?
2. Опишіть сутність та наведіть 3 приклади застосування штучного інтелекту.
3. Які переваги надають хмарні технології бізнесу та звичайним користувачам?
4. Що таке Інтернет речей (IoT)? Як він змінює наші міста та домівки?
5. Чому кібербезпека стала одним із найважливіших трендів сучасності?
6. Яка основна ідея технології блокчейн? Де вона застосовується, окрім криптовалют?

7. Чим квантові обчислення відрізняються від класичних? Які задачі вони можуть вирішити?

8. Що таке метавсесвіт? Які технології лежать в його основі?

9. Чому периферійні обчислення (Edge Computing) важливі для розвитку безпілотних автомобілів?

10. Як ви вважаєте, який з розглянутих трендів матиме найбільший вплив на ваше майбутнє як ІТ-фахівця? Чому?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Аналітичні та прогнозні ресурси

1. Gartner – Top Strategic Technology Trends: https://www.gartner.com/en/information-technology/insights/top-technology-trends

2. Deloitte – Tech Trends: https://www2.deloitte.com/us/en/insights/focus/tech-trends.html

3. McKinsey – Digital/McKinsey: https://www.mckinsey.com/featured-insights/digital

4. CB Insights – Research: https://www.cbinsights.com/research/

Новини технологій

5. TechCrunch: https://techcrunch.com/

6. The Verge: https://www.theverge.com/

7. Wired: https://www.wired.com/

8. MIT Technology Review: https://www.technologyreview.com/

9. DOU.ua: https://dou.ua/ – український погляд на тренди.

Книги

10. Кевін Келлі. "Невідворотне. 12 технологій, що формують наше майбутнє".

11. Мартін Форд. "Пришестя роботів: Технології та загроза майбутнього безробіття".

12. Пітер Діамандіс, Стівен Котлер. "Крах чи зліт? Як технології змінять бізнес і майбутнє".

Відео

13. YouTube-канали: "ColdFusion", "Marques Brownlee", "Lex Fridman Podcast" (інтерв’ю з експертами).

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Слідкуйте за трендами постійно.** Підпишіться на кілька технологічних ЗМІ або YouTube-каналів. Виділяйте 15-20 хвилин на день на читання новин. Це допоможе вам бути в курсі та розуміти, куди рухається індустрія.

2. **Критично ставтеся до хайпу.** Не все, що називають "проривом", є ним насправді. Вчіться відрізняти реальні технологічні досягнення від маркетингових перебільшень. Використовуйте матрицю зрілості технологій Gartner (Hype Cycle) як інструмент.

3. **Пов’язуйте тренди з власним навчанням.** Якщо ви бачите, що AI стає всюдисущим, подумайте, як ви можете інтегрувати вивчення ML у свій навчальний план. Якщо кібербезпека – ваш інтерес, шукайте відповідні курси.

4. **Не бійтеся майбутнього.** Технології змінюють ринок праці, але вони також створюють нові, часто навіть цікавіші можливості. Головне – бути гнучким і готовим вчитися протягом усього життя.

5. **Технології – це інструмент, а не мета.** Завжди пам’ятайте про людей, для яких ви створюєте технології. Найуспішніші тренди – ті, що роблять життя людей кращим, безпечнішим і зручнішим.

ПРАКТИЧНЕ ЗАНЯТТЯ 13
ПОЄДНАННЯ НАВЧАННЯ В УНІВЕРСИТЕТІ ТА ПРАКТИЧНОЇ РОБОТИ В ІТ-КОМПАНІЇ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників практичних навичок планування свого часу та кар'єри, розуміння можливостей поєднання академічного навчання з роботою в ІТ, а також розвиток навичок самопрезентації та професійного нетворкінгу.

Основні завдання заняття:

1. Ознайомити з різними формами практичної діяльності для студентів: стажування (internship), часткова зайнятість (part-time), фріланс (freelance), робота над власними проєктами, участь у відкритому коді (open source).

2. Дослідити основні виклики та стратегії успішного поєднання роботи та навчання: тайм-менеджмент, пріоритезація, боротьба з вигоранням.

3. Розвинути практичні навички:

- *тайм-менеджменту та планування:* створення реалістичного розкладу тижня.
- *самопрезентації:* вміння представити себе, свої навички та цілі.
- *нетворкінгу:* вміння встановлювати професійні контакти, проходити співбесіду.

4. Розвинути ключові навички самоорганізації, комунікації та аналізу власних ресурсів.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Чому варто починати працювати ще під час навчання?

Поєднання навчання та роботи в ІТ має багато переваг:

Практичний досвід: теорія з університету закріплюється реальними задачами. Ви вчитеся застосовувати знання.

Портфоліо: робота над реальними проєктами дає матеріал для портфоліо, яке є ключовим при пошуку роботи після випуску.

Фінансова незалежність: можливість заробляти власні гроші.

Професійні зв'язки (нетворкінг): знайомство з колегами, менторами, іншими фахівцями, які можуть допомогти в майбутньому.

Розуміння індустрії: ви зсередини бачите, як працюють IT-компанії, які там процеси, культура, інструменти.

Конкурентна перевага: після закінчення університету ви маєте не лише диплом, а й досвід роботи, що робить вас набагато привабливішим кандидатом.

2.2. Форми зайнятості для студентів

Стажування (Internship)

Суть: тимчасова оплачувана (або неоплачувана) робота в компанії, спрямована на навчання та отримання досвіду. Зазвичай триває від 2 до 6 місяців. Часто передбачає ментора.

Переваги: структуроване навчання, занурення в професію, високий шанс отримати повноцінну пропозицію після стажування.

Де шукати: сайти компаній (кар'єрні розділи), DOU.ua, LinkedIn, телеграм-канали з вакансіями для початківців.

Часткова зайнятість (Part-time)

Суть: робота на неповний робочий день (наприклад, 20-30 годин на тиждень). Дозволяє гнучко поєднувати з навчанням.

Переваги: стабільний дохід, глибше занурення в роботу, ніж на стажуванні.

Де шукати: спеціалізовані сайти (djinni.co, work.ua), знайомства.

Фріланс (Freelance)

Суть: виконання разових проєктів або задач на замовлення через онлайн-платформи (Upwork, Fiverr, Freelance.ua).

Переваги: максимальна гнучкість, можливість працювати з будь-якої точки світу, самому обирати задачі.

Недоліки: нестабільний дохід, складніше для початківців без досвіду та портфоліо.

Власні проєкти та Open Source

Суть: розробка власних застосунків, участь у проєктах з відкритим кодом на GitHub.

Переваги: розвиває навички, демонструє ініціативу, створює портфоліо. Може призвести до пропозицій роботи.

2.3. Ключові виклики та як їх долати

Брак часу: найголовніша проблема. *Рішення:* чітке планування, використання технік тайм-менеджменту (див. нижче), вміння говорити "ні" непотрібним речам.

Втома та вигорання. *Рішення:* обов’язково планувати час на відпочинок, сон, спорт, хобі. Дотримуватися балансу. Навчитися відпочивати, а не просто "не працювати".

Прокрастинація. *Рішення:* розбивати великі задачі на маленькі, використовувати Pomodoro, прибирати відволікаючі фактори.

Складність поєднання графіків. *Рішення:* з самого початку обговорювати з роботодавцем можливість гнучкого графіка, особливо в період сесії. Використовувати спільний календар (Google Calendar) для всіх справ.

2.4. Техніки тайм-менеджменту для студентів

(Детальніше див. у Практичному занятті 11)

Планування в календарі: використовуйте Google Calendar або інший додаток, щоб заздалегідь запланувати всі пари, робочі години, дедлайни та особисті справи. Бачте свій тиждень цілком.

Матриця Ейзенхауера: допомагає визначити пріоритети: що важливо і терміново, що важливо, але не терміново (сюди потрапляє більшість навчання та роботи).

Метод Pomodoro: ідеальний для виконання домашніх завдань та робочих задач, що потребують концентрації.

Time Blocking (Блокування часу): виділяйте в календарі окремі блоки для конкретних типів робіт (наприклад, "Понеділок 10:00-12:00 – робота над курсовою", "Середа 15:00-18:00 – робота в компанії").

2.5. Нетворкінг для студентів

LinkedIn: створіть профіль, заповніть його повністю (фото, навички, освіта). Додавайте викладачів, одногрупників, знайомих з ІТ. Підписуйтесь на компанії.

Профільні заходи: відвідуйте мітапи, конференції (часто є студентські квитки), хакатони. Там можна познайомитися з представниками компаній.

Спілкування: не бійтеся підходити і знайомитися. Ставте розумні питання. Після заходу можна додати людину в LinkedIn і написати: "Привіт, було приємно познайомитися на [назва заходу]".

Стажування: нетворкінг часто допомагає дізнатися про відкриті позиції до того, як вони з’являться у відкритому доступі.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.

• для аудиторної роботи: аркуші паперу А4, ручки, фломастери/кольорові олівці (для створення постерів).

• програмне забезпечення (на вибір):

o сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).

o календарі (Google Calendar) – для створення розкладів.

o текстовий процесор (Google Docs, Microsoft Word Online).

o інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Симуляція планування робочого тижня

Мета завдання: навчитися на практиці планувати свій час, враховуючи навчання, роботу та відпочинок, використовуючи техніки тайм-менеджменту.

Теоретичне підґрунтя

Багато студентів вважають, що поєднати навчання і роботу неможливо. Але з правильним плануванням це цілком реально. Це завдання допоможе вам побачити, як це може виглядати на практиці.

Детальна інструкція виконання

Етап 1. Формування груп та визначення умов (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Уявіть, що ви – студенти 2-3 курсу, які вирішили почати працювати в IT.

Визначте параметри для планування:

Навчання: приблизно 25-30 годин на тиждень (пари, самостійна робота, підготовка до занять).

Робота: уявна роль в IT-компанії з частковою зайнятістю. Виберіть один з варіантів:

▪ *Варіант А:* стажер-розробник (Python). Графік: 3 рази на тиждень по 4 години + 8 годин у суботу. Разом 20 годин.

▪ *Варіант Б:* тестувальник (QA) на part-time. Графік: щодня по 3-4 години. Разом 20-25 годин.

▪ *Варіант В:* фріланс-верстальник. Графік: вільний, але треба виконати 2-3 невеликі замовлення на тиждень.

Етап 2. Складання розкладу (10 хвилин).

Використовуйте аркуш паперу або онлайн-інструмент (Google Calendar, Excel). Створіть таблицю тижня (з понеділка по неділю, з 8:00 до 22:00).

Розподіліть на тижні:

- Фіксовані події: пари в університеті (придумайте приблизний розклад, наприклад, 4 пари на день з 9:00 до 14:00 у будні). Робочі години (відповідно до обраного варіанту).
- Блоки для самостійної роботи: коли ви робитимете домашні завдання, готуватиметесь до семінарів.
- Блоки для сну та відпочинку: обов'язково залиште час на сон (7-8 годин щодня), харчування, прогулянки, хобі, зустрічі з друзями. Це так само важливо, як і робота!
- Буферний час: залишіть кілька "вікон" на непередбачувані справи.

Етап 3. Обґрунтування техніки тайм-менеджменту (5 хвилин).

Визначте, яку техніку тайм-менеджменту ви використовували (або плануєте використовувати), щоб виконувати цей розклад.

Приклади:

- "Ми використовували принцип Time Blocking, виділивши в календарі окремі блоки для навчання, роботи та відпочинку."
- "Для виконання домашніх завдань ми плануємо використовувати метод Pomodoro, щоб підвищити концентрацію."
- "Пріоритезувати завдання допомагатиме Матриця Ейзенхауера, особливо в періоди дедлайнів."

Етап 4. Створення презентації (5 хвилин).

Створіть 2-3 слайди або постер, які візуалізують ваш план.

Структура:

- *Слайд 1.* Опис ролі (студент + [посада в IT]), загальне навантаження (годин на тиждень).
- *Слайд 2.* Розклад тижня у вигляді таблиці або кольорової діаграми (різними кольорами позначено навчання, роботу, відпочинок).
- *Слайд 3.* Використана техніка тайм-менеджменту та пояснення, чому вона допоможе. Короткий висновок: чи вважаєте ви цей розклад реалістичним?

Етап 5. Презентація та обговорення (15 хвилин).

Кожна група презентує свій розклад (2-3 хвилини).

Загальна дискусія:

- Які основні виклики ви побачили під час планування?

- Чи вистачило часу на відпочинок?
- Що робити, якщо в університеті з’являються додаткові завдання або на роботі трапляються аврали? (Важливість буферного часу).

Критерії оцінювання (Завдання 1):

Реалістичність: розклад виглядає здійсненним, враховано час на сон та відпочинок.

Структурованість: чітке розмежування навчання, роботи, особистого часу.

Використання технік: усвідомлений вибір техніки тайм-менеджменту.

Командна робота.

Якість презентації.

Завдання 2. Рольова гра «Нетворкінг у ІТ»

Мета завдання: розвинути навички самопрезентації, вміння вести діалог з представником компанії та отримати перший досвід проходження "інтерв’ю" на стажування.

Теоретичне підґрунтя

В ІТ дуже важливо вміти себе "продати". Перше враження, яке ви справите на HR-а або технічного спеціаліста, може вирішити долю вашої кандидатури. Це завдання – безпечне тренування таких навичок.

Детальна інструкція виконання

Етап 1. Підготовка до інтерв’ю (5 хвилин).

Об’єднайтесь у пари. Розподіліть ролі:

Студент: шукає стажування в ІТ-компанії. Має базові знання (наприклад, трохи знає Python, HTML/CSS, або тестування). Готує коротку самопрезентацію (1-2 хвилини).

Представник компанії: HR-менеджер або технічний лідер (Tech Lead). Його мета – зрозуміти, чи підходить кандидат.

Етап 2. Проведення інтерв’ю (7-8 хвилин).

Самопрезентація студента (2 хв):

- Ім’я, курс, спеціальність.
- Що вже вивчив (технології, мови).
- Чому цікавить саме ця компанія / сфера.

- Які має цілі (чого хоче навчитися на стажуванні).

Питання від представника компанії (5-6 хв): представник ставить 3-4 запитання. *Приклади:*

- Розкажи про свій останній навчальний проєкт. Яку задачу вирішував? З якими труднощами зіткнувся?
- Як ти плануєш поєднувати стажування з навчанням в університеті?
- Які технології ти б хотів вивчити в першу чергу і чому?
- Як ти дізнаєшся про новини в IT? (які сайти, блоги читаєш)
- Чи є в тебе досвід роботи в команді (наприклад, над груповим проєктом)?

Студент відповідає на питання. Представник може робити нотатки.

Етап 3. Підведення підсумків (3-4 хвилини).

Разом обговоріть, як пройшло інтерв'ю.

Запишіть підсумок (5-7 речень) у спільному документі:

- Що в самопрезентації було сильним?
- Які питання були найскладнішими?
- Поради для ефективного нетворкінгу: (наприклад, "бути впевненим, але не нахабним", "уважно слухати питання", "говорити про свої досягнення, але не перебільшувати", "цікавитися компанією").

Етап 4. Презентація та обговорення (10 хвилин).

За бажанням, 2-3 пари діляться своїми висновками та порадами.

Загальна дискусія:

- Які навички самопрезентації найважливіші?
- Як підготуватися до такого інтерв'ю в реальному житті?
- Що робити, якщо не знаєш відповіді на технічне питання? (Чесність і бажання навчитися часто цінуються більше, ніж "плавання" у відповідях).

Критерії оцінювання (Завдання 2):

Якість самопрезентації: чіткість, структурованість, доречність.

Якість запитань: питання від "представника" є змістовними.

Вміння слухати та відповідати: відповіді "студента" по суті, демонструють розуміння.

Чіткість підсумку: зафіксовано конкретні поради.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

План поєднання навчання та роботи в IT

Мета завдання: створити реалістичний індивідуальний план дій на найближчі 1-2 роки, який включає пошук першого досвіду роботи в IT та ефективне поєднання його з навчанням.

Вступ. Це завдання – ваш особистий стратегічний план. Воно допоможе вам не просто мріяти про роботу в IT, а спланувати конкретні кроки для досягнення цієї мети, враховуючи ваші поточні ресурси (час, знання).

Детальна інструкція виконання

Крок 1. Вибір форми роботи та цілі.

Визначте, яка форма роботи для вас є найбільш бажаною та реалістичною на найближчі 1-2 роки:

- Стажування (Internship) в українській або міжнародній компанії.
- Часткова зайнятість (Part-time) на позиції Junior.
- Фріланс на біржах (Upwork, Freelance.ua).
- Робота над власним стартапом / pet-проєктом.
- Участь у Open Source проєктах.

Крок 2. Дослідження ринку та вимог.

Знайдіть 3-5 реальних вакансій на стажування або part-time для початківців на djinni.co, work.ua, LinkedIn.

Випишіть, які технології та навички є обов’язковими (must have) для цих позицій.

Оцініть, яких навичок вам бракує.

Крок 3. Створення плану.

Створіть документ (або ментальну карту) з наступними розділами:

1. Моя ціль: (чітко сформулюйте, чого хочете досягти. Наприклад: "Знайти стажування Frontend-розробника до кінця 2-го курсу").

2. Обрана форма роботи: (опишіть, чому ви обрали саме її).

3. План розвитку навичок (до виходу на роботу):

- Які технології треба вивчити в першу чергу? (Спираючись на аналіз вакансій).
- Які курси / книги / ресурси для цього використаєте? (Додайте посилання).
- Який pet-проєкт плануєте зробити для портфоліо?

4. План пошуку роботи:

- Які компанії вас цікавлять? (Складіть список з 5-10 компаній).
- Як ви плануєте виходити на контакт? (Наприклад, через LinkedIn, через відгуки на вакансії, через знайомих).
- Коли плануєте почати активний пошук?

5. План поєднання роботи та навчання (після виходу):

- Розклад тижня: Створіть реалістичний розклад (як у Завданні 1), враховуючи ваше реальне навчальне навантаження та бажану кількість робочих годин (наприклад, 20 годин/тиждень).
- Техніка тайм-менеджменту: Яку техніку ви будете використовувати і чому?
- Стратегія відпочинку та боротьби з вигоранням: Що ви будете робити, щоб відновлювати сили?

6. Стратегії нетворкінгу:

- Що ви вже зробили / плануєте зробити для розширення професійних зв'язків? (Наприклад, створити профіль на LinkedIn, відвідати 2 мітапи, взяти участь у хакатоні).

Крок 4. Оформлення та здача.

Обов'язково вкажіть наприкінці роботи список використаних джерел (мінімум 2-3, з гіперпосиланнями на статті, курси, сайти з вакансіями).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Реалістичність та практичність: план спирається на реальні вимоги ринку та враховує особисті обставини.

Конкретність: усі пункти містять конкретні дії, а не загальні фрази.

Структурованість: чітке слідування плану, логічний виклад.

Рефлексія: план враховує особисті сильні сторони та зони для розвитку.

Використання джерел: коректні посилання на актуальні ресурси.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Чому варто починати працювати в IT ще під час навчання? Назвіть 3-4 переваги.

2. Які існують форми зайнятості для студентів в ІТ? Опишіть переваги та недоліки кожної.

3. Які основні виклики поєднання роботи та навчання? Як їх долати?

4. Назвіть 3 техніки тайм-менеджменту, які є найбільш корисними для студента, що працює. Поясніть чому.

5. Що таке нетворкінг? Як студент може його ефективно використовувати для пошуку роботи?

6. Яку інформацію має містити профіль у LinkedIn для студента, який шукає стажування?

7. Що таке "пет-проєкт" (pet project) і чому він важливий для початківця?

8. Як підготуватися до інтерв'ю на стажування?

9. Що робити, якщо ви розумієте, що не встигаєте і поєднання роботи та навчання призводить до вигорання?

10. Який ваш особистий план дій на найближчі 1-2 роки щодо отримання першого досвіду в ІТ?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Ресурси про тайм-менеджмент та самоорганізацію:

1. MindTools – Time Management: https://www.mindtools.com/pages/main/time-management.htm

2. Coursera: "Work Smarter, Not Harder: Time Management for Personal & Professional Productivity": https://www.coursera.org/learn/work-smarter-not-harder (безкоштовний аудит)

3. Todoist Blog: https://todoist.com/blog – багато статей про продуктивність.

Ресурси про пошук роботи та стажування:

4. DOU.ua – Розділ "Ринок праці": https://dou.ua/ – огляди зарплат, статті про пошук роботи, база компаній.

5. Djinni.co – Career Guide: https://djinni.co/ – корисні статті про пошук роботи в ІТ.

6. freeCodeCamp – "How to Get an IT Internship": https://www.freecodecamp.org/news/how-to-get-an-it-internship/

7. Coursera – "How to Find a Job or Internship": https://www.coursera.org/articles/find-a-job-or-internship

Ресурси про нетворкінг та LinkedIn:

8. LinkedIn Learning: Курси з нетворкінгу та створення профілю (часто доступні через університет).

9. The Muse – Networking: https://www.themuse.com/advice/networking – статті про професійний нетворкінг.

10. Forbes – Networking Tips: https://www.forbes.com/sites/forbescoachescouncil/... – статті на тему.

Книги

11. Девід Аллен. "Як упорядкувати справи. Мистецтво продуктивності без стресу" (Getting Things Done).

12. Кал Ньюпорт. "Цифровий мінімалізм" та "Глибока робота".

13. Лора Вандеркам. "Що роблять успішні люди у вихідні".

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Почніть з малого.** Не намагайтеся одразу знайти full-time роботу. Почніть зі стажування або невеликих фріланс-замовлень. Головне – зробити перший крок.

2. **Будьте чесними з роботодавцем.** На співбесіді одразу кажіть, що ви студент і маєте навчання. Обговоріть можливість гнучкого графіка, особливо на час сесії. Чесність цінується.

3. **Використовуйте університетські ресурси.** Багато університетів мають центри кар’єри, домовленості з компаніями про стажування. Дізнайтеся, чи є таке у вашому.

4. **Не нехтуйте навчанням.** Так, робота дає досвід, але фундаментальні знання, які ви отримуєте в університеті (математика, алгоритми, методології), стануть вам у нагоді в довгостроковій перспективі для кар’єрного зростання.

5. **Слідкуйте за балансом.** Найцінніший ресурс – це ваше здоров’я. Якщо ви відчуваєте, що не встигаєте і постійно втомлені, перегляньте свій графік. Можливо, варто трохи зменшити робоче навантаження, ніж довести себе до вигорання.

6. **Створіть LinkedIn профіль вже сьогодні.** Не відкладайте. Навіть якщо ви поки що не шукаєте роботу, це чудовий спосіб почати будувати свою професійну присутність в інтернеті.

ПРАКТИЧНЕ ЗАНЯТТЯ 14
СТАЖУВАННЯ, ІНТЕРНАТУРА ТА СТВОРЕННЯ ПОРТФОЛІО ДЛЯ ІТ-ФАХІВЦЯ

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників чіткого розуміння того, як отримати перший практичний досвід в ІТ, як правильно підготувати документи для подачі на стажування (резюме, портфоліо) та як презентувати себе потенційному роботодавцю.

Основні завдання заняття:

1. Ознайомити з поняттями стажування (internship) та інтернатури, їхніми цілями, структурою та відмінностями від повноцінної роботи.

2. Дослідити реальні вимоги ІТ-компаній до кандидатів на стажування, проаналізувати процес відбору.

3. Розвинути практичні навички:

– *створення резюме (CV):* структурування інформації, виділення ключових навичок, опис проєктів.

– *створення портфоліо:* відбір та опис проєктів, демонстрація компетенцій.

– *самопрезентації:* вміння представити свої досягнення у вигідному світлі.

4. Розвинути ключові навички самоорганізації, аналітичного мислення, комунікації та здатності до рефлексії (отримання та надання зворотного зв’язку).

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Стажування (Internship) та Інтернатура: що це і для чого?

Стажування (internship) – це тимчасова (зазвичай від 2 до 6 місяців) робота в компанії, основною метою якої є навчання студента або випускника-початківця практичним навичкам під керівництвом досвідчених менторів. Інтернатура – часто є синонімом, але іноді позначає більш структуровані програми для талановитих студентів, які можуть тривати довше і включати ротацію між відділами.

Основні цілі стажування:

• отримання практичного досвіду роботи в реальних проєктах.

• застосування теоретичних знань, отриманих в університеті.

• розуміння внутрішніх процесів IT-компанії (Agile/Scrum, інструменти, комунікація).

• налагодження професійних зв'язків (нетворкінг).

• підвищення шансів на отримання повноцінної роботи (full-time offer) після закінчення стажування.

2.2. Як знайти стажування?

Кар'єрні сторінки компаній: більшість великих компаній (EPAM, SoftServe, GlobalLogic, Ciklum, Google, Microsoft) мають окремі розділи на сайтах з вакансіями для студентів та випускників.

Сайти пошуку роботи: djinni.co (фільтр "Trainee/Intern"), work.ua, robota.ua, LinkedIn.

Університетські центри кар'єри: багато університетів співпрацюють з компаніями та інформують студентів про відкриті позиції.

Професійні заходи: хакатони, мітапи, конференції – чудове місце для знайомства з представниками компаній.

Прямі контакти: якщо у вас є знайомі, які вже працюють в IT, не соромтеся звертатися до них за порадою або "рефералом" (рекомендацією).

2.3. Типовий процес відбору на стажування

1. Подача заявки: ви надсилаєте резюме (CV) та, можливо, супровідний лист (cover letter).

2. Скринінг резюме: HR-менеджер переглядає резюме та відбирає кандидатів, які відповідають базовим критеріям.

3. Тестове завдання: кандидатам пропонують виконати невелике практичне завдання, щоб оцінити їхні навички. Для розробників це може бути написання простої програми або функції, для QA – створення тест-кейсів.

4. Співбесіда (інтерв'ю): зазвичай складається з двох етапів:

- HR-інтерв'ю: оцінка мотивації, soft skills, загальних знань про компанію.
- Технічне інтерв'ю: спілкування з технічним спеціалістом (Team Lead, Senior Developer). Можуть запитувати теорію (алгоритми, структури даних), пропонувати вирішити задачі, обговорювати виконане тестове завдання.

2.4. Резюме (CV) для IT-початківця: структура та поради

Резюме – це ваша візитівка. Воно має бути чітким, лаконічним та адаптованим під конкретну вакансію.

Структура резюме:

1. Контактна інформація: ім’я та прізвище, місто, телефон, email, посилання на LinkedIn, GitHub (обов’язково!).

2. Короткий профіль (Summary): 2-3 речення про вас, ваші цілі та ключові навички. (Наприклад: "Студент 2-го курсу спеціальності ‘Комп’ютерні науки’. Маю базові знання Python та SQL. Шукаю можливість стажування для отримання практичного досвіду в розробці backend-систем.")

3. Технічні навички (Hard Skills): список технологій, мов програмування, інструментів, якими ви володієте. Групуйте за категоріями (наприклад, "Мови програмування: Python, SQL; Інструменти: Git, VS Code").

4. Проєкти (Projects): найважливіший розділ для початківця. Опишіть 2-3 навчальні або особисті проєкти. Для кожного вкажіть: назву, короткий опис, використані технології, вашу роль та результат (чого навчилися, що зробили). Додайте посилання на код (GitHub) або демо, якщо можливо.

5. Освіта (Education): назва університету, факультет, спеціальність, рік початку навчання та очікуваний рік закінчення. Можна вказати середній бал (якщо він високий).

6. М’які навички (Soft Skills): 3-5 ключових навичок, важливих для IT (наприклад, "командна робота", "комунікабельність", "швидке навчання", "відповідальність").

7. Мови: рівень володіння українською, англійською (та іншими мовами).

8. Додатково (Optional): сертифікати, участь у хакатонах, волонтерство, хобі.

2.5. Портфоліо: більше, ніж просто резюме

Портфоліо – це збірка ваших найкращих робіт, яка наочно демонструє ваші навички. Для IT-фахівця портфоліо часто є важливішим за резюме.

- GitHub: для розробників GitHub – це і є портфоліо. Важливо мати кілька добре задокументованих проєктів з README-файлами, які пояснюють, що це за проєкт, як його встановити та використовувати.
- Сайт-портфоліо: для дизайнерів, фронтенд-розробників, аналітиків даних (з прикладами звітів) може бути корисним мати власний вебсайт з описами проєктів, скріншотами, посиланнями.

Що має бути в описі проєкту в портфоліо:

- назва проєкту.
- контекст / проблема (чому ви взялися за цей проєкт?).
- ваша роль (якщо це був командний проєкт).
- використані технології та інструменти.
- ключові функції / особливості (що саме ви зробили).

- результат / висновки (чого навчилися, що можна покращити).
- посилання на код (GitHub) або живу демо-версію.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки.
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - текстовий процесор (Google Docs, Microsoft Word Online) – для створення резюме.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.
  - редактори коду (VS Code, або онлайн-редактори) – для створення чернетки сайту-портфоліо.

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Аналіз можливостей стажування

Мета завдання: навчитися знаходити, аналізувати та порівнювати пропозиції стажувань в ІТ-компаніях, розуміти вимоги ринку до початківців.

Теоретичне підґрунтя

Перш ніж подавати заявку, важливо зрозуміти, чого саме компанії очікують від стажерів. Аналіз вакансій допомагає визначити, які навички варто підтягнути, а які проєкти додати в портфоліо.

Детальна інструкція виконання

Етап 1. Формування груп та вибір компаній (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть **1-2 ІТ-компанії** для аналізу. *Приклади:*

- Великі українські компанії: EPAM, SoftServe, GlobalLogic, Ciklum, Luxoft, Intellias.
- Продуктові компанії: Grammarly, Jooble, Ajax Systems, GitLab.

- Міжнародні гіганти: Google (студентські програми), Microsoft, Amazon.

Знайдіть на їхніх сайтах розділи "Careers", "Jobs", "Internships" або "Студентам".

Етап 2. Збір інформації (10 хвилин).

Знайдіть конкретну вакансію на стажування (наприклад, "Intern/Junior Software Engineer", "QA Intern", "Frontend Intern").

Проаналізуйте її за наступними критеріями та заповніть таблицю:

Критерій	**Компанія 1: [Назва]**	**Компанія 2: [Назва] (якщо є)**
Назва програми/посади		
Ключові технології (must have)	(наприклад, Python, SQL, ООР)	
Бажані технології (nice to have)		
Вимоги до soft skills	(наприклад, англійська, командна робота)	
Етапи відбору	(наприклад: резюме -> тестове -> технічне інтерв'ю -> HR)	
Тривалість стажування		
Переваги для стажерів	(наприклад, ментор, оплата, гнучкий графік, перспектива)	

Етап 3. Створення презентації (5 хвилин).

Створіть 2-3 слайди або постер, які узагальнюють ваше дослідження.

Структура:

- *Слайд 1.* Назва компанії та програми стажування.
- *Слайд 2.* Ключові вимоги до кандидата (технічні + soft skills).
- *Слайд 3.* Процес відбору та переваги.

Висновок: Чи варто подаватися? Що потрібно підтягнути, щоб мати шанс?

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група презентує результати (2-3 хвилини).

Загальна дискусія:

- Які навички (технічні та м'які) є найпоширенішими вимогами?
- Чи відрізняються вимоги продуктових компаній від аутсорсингових?
- Що з переліку вимог ви вже маєте, а над чим треба працювати?

Критерії оцінювання (Завдання 1):

Актуальність: проаналізовано реальні вакансії на сайтах компаній.

Повнота: зібрано інформацію за всіма критеріями.

Практичність: зроблено висновки, які можна використати для саморозвитку.

Командна робота.

Якість презентації.

Завдання 2. Створення чернетки резюме для IT-стажування

Мета завдання: отримати перший досвід написання резюме, навчитися структурувати інформацію про себе та свої навички, а також практикувати навички надання та отримання конструктивного зворотного зв'язку.

Теоретичне підґрунтя

Хороше резюме значно підвищує ваші шанси бути поміченим HR-менеджером. Важливо не просто перелічити факти, а подати їх так, щоб підкреслити вашу відповідність посаді.

Детальна інструкція виконання

Етап 1. Вибір ролі та підготовка (5 хвилин).

Об'єднайтесь у пари.

Оберіть уявну роль, на яку ви будете писати резюме. *Приклади:*

- стажер-розробник Python (Junior Python Developer Intern)
- стажер-тестувальник (QA Intern)
- стажер-фронтенд розробник (Frontend Intern)
- стажер-аналітик даних (Data Analyst Intern)

Ознайомтеся зі структурою резюме з теоретичної частини.

Етап 2. Написання резюме (10 хвилин).

Кожен студент індивідуально (але радячись із партнером) складає чернетку резюме на аркуші паперу або в Google Docs.

Обов'язкові елементи резюме:

- контакти та посилання (вигадайте email, LinkedIn, GitHub).
- короткий профіль (2-3 речення).
- технічні навички (список технологій, які, на вашу думку, потрібні для обраної ролі).

Проєкти: опишіть 1-2 уявні навчальні проєкти. *Приклад для Python-розробника:*

Назва проєкту: бот для Telegram з курсом валют

Технології: Python, python-telegram-bot API, requests

Опис: розробив бота, який за запитом користувача надсилає актуальний курс долара та євро. Використовував публічне API ПриватБанку. Навчився працювати з API та обробляти JSON-відповіді.

- Освіта (ваш університет, спеціальність).
- М’які навички та мови.

Етап 3. Взаємний зворотний зв’язок (5 хвилин).

Обміняйтеся резюме з партнером.

Прочитайте резюме один одного та дайте конструктивний зворотний зв’язок. Відповідаючи на питання:

- Що в резюме є сильним? (наприклад, "класний опис проєкту, видно, що ти робив").
- Що можна покращити? (наприклад, "додай посилання на GitHub", "опиши проєкт детальніше", "перевір граматику").

Запишіть отримані поради.

Етап 4. Фіксація результатів (5 хвилин).

Запишіть фінальну версію резюме (або основні правки) та короткий підсумок зворотного зв’язку (3-5 речень) у спільному документі.

Етап 5. Презентація та обговорення (5 хвилин).

За бажанням, 1-2 пари діляться своїми резюме та отриманими порадами.

Загальне обговорення:

- Що було найскладнішим у написанні резюме?
- Які поради виявилися найкориснішими?
- Чому опис проєктів важливіший за список технологій?

Критерії оцінювання (Завдання 2):

Структура резюме: відповідність рекомендованій структурі, наявність всіх ключових розділів.

Якість опису проєктів: опис є конкретним, зрозумілим, показує застосування технологій.

Зворотний зв’язок: поради є конструктивними, конкретними, спрямованими на покращення.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Створення портфоліо для IT-фахівця

Мета завдання: створити прототип професійного портфоліо, яке наочно демонструє навички, проєкти та потенціал для майбутнього роботодавця.

Вступ. Портфоліо – це ваш головний інструмент для пошуку роботи. Воно має розповідати історію про вас як фахівця. Це завдання допоможе вам зробити перший крок до створення такого інструменту.

Детальна інструкція виконання

Крок 1. Вибір професії та формату.

Визначте, для якої IT-професії ви створюєте портфоліо (наприклад, Frontend Developer, Python Developer, QA Engineer, UI/UX Designer, Data Analyst). Виберіть формат, в якому ви будете його подавати:

- **Варіант А (Текстовий документ):** детальний документ у Google Docs або Word, який містить резюме та розширені описи проєктів.
- **Варіант Б (Ментальна карта):** візуальне представлення вашого портфоліо у вигляді мапи з гілками: "Про мене", "Резюме", "Проєкти", "Навички", "Контакти". Це може бути структура для майбутнього сайту.
- **Варіант В (Чернетка вебсайту):** створіть просту HTML-сторінку, яка буде каркасом вашого сайту-портфоліо. Вона має містити заголовки, списки навичок та описи проєктів (сам сайт може бути незверстаним "до кінця", але структура має бути зрозумілою).

Крок 2. Створення контенту портфоліо.

Ваше портфоліо має містити наступні елементи:

1. Профіль / Про себе: короткий текст (3-5 речень) про вас, вашу мотивацію, цілі та ключові компетенції.
2. Резюме: включіть сюди оновлену версію резюме з аудиторного завдання.
3. Проєкти: опишіть 2-3 уявні або реальні навчальні проєкти. Для кожного проєкту обов’язково вкажіть:
   - Назва проєкту.
   - Зображення / Скріншот (якщо можливо, хоча б уявний): можна додати placeholder.
   - Короткий опис (що це, для чого).
   - Використані технології (список).

- Ваша роль та конкретний внесок: що саме ви зробили? (наприклад, "розробив базу даних", "створив адаптивний інтерфейс", "написав модульні тести").
- Посилання на код (GitHub): додайте посилання на репозиторій (можна на уявний профіль).
- Висновки / Чого навчилися: (наприклад, "Навчився працювати з REST API та обробляти асинхронні запити").

4. Навички (Skills): чіткий перелік технічних (hard skills) та м'яких (soft skills) навичок.

5. Контакти: email, посилання на LinkedIn, GitHub.

Крок 3. Оформлення та здача.

Обов'язково вкажіть наприкінці роботи список використаних джерел (мінімум 2-3, з гіперпосиланнями на статті про створення портфоліо, шаблони тощо).

Завантажте готовий файл (PDF, DOCX, HTML-файл, JPG/PNG для ментальної карти) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Повнота: наявність всіх обов'язкових елементів (профіль, резюме, проєкти, навички, контакти).

Якість опису проєктів: описи детальні, конкретні, показують роль автора та використані технології.

Структурованість: інформація подана логічно, легко сприймається.

Візуальна привабливість (для обраного формату): документ/карта/сторінка виглядають охайно, читабельно.

Використання джерел: коректні посилання на корисні ресурси.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. У чому різниця між стажуванням (internship) та повноцінною роботою (full-time job)?
2. Назвіть основні етапи відбору кандидатів на стажування в IT-компанії.
3. Для чого потрібне тестове завдання? Що воно показує роботодавцю?
4. Які обов'язкові розділи мають бути в резюме IT-початківця?

5. Чому опис проєктів у резюме є важливішим за простий перелік технологій?

6. Що таке портфоліо? Чим воно відрізняється від резюме?

7. Яку інформацію має містити опис проєкту в портфоліо?

8. Чому GitHub є важливою частиною портфоліо для розробника?

9. Де можна шукати інформацію про стажування? Назвіть 3-4 джерела.

10. Як ви можете покращити своє резюме та портфоліо вже під час навчання на 1-2 курсі?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Ресурси про резюме та портфоліо:

1. Canva – Resume Templates: https://www.canva.com/resumes/templates/ – Безкоштовні шаблони для гарного оформлення резюме.

2. freeCodeCamp – "How to Build a Portfolio": https://www.freecodecamp.org/news/how-to-build-a-portfolio/

3. Coursera – "How to Create a Portfolio": https://www.coursera.org/articles/how-to-create-a-portfolio

4. LinkedIn Learning: Курси з написання резюме та створення портфоліо.

5. DOU.ua – Статті про резюме: https://dou.ua/ (шукайте за тегом "резюме" або "CV").

Ресурси про стажування:

6. EPAM Careers: https://www.epam.com/careers (шукайте студентські програми)

7. SoftServe Careers: https://www.softserveinc.com/en-us/careers (розділ для студентів)

8. GlobalLogic Careers: https://www.globallogic.com/careers/

9. Google Students: https://careers.google.com/students/

10. Microsoft Internships: https://careers.microsoft.com/students

Книги

11. Гейл Лаакманн Мак-Доуелл. "Кар’єра програміста. Як влаштуватися на роботу в Google, Microsoft або іншу велику компанію". (Розділи про резюме та проходження співбесід).

12. Остін Клеон. "Показуйте свою роботу! 10 способів змусити світ побачити вашу творчість". (Про важливість демонстрації своїх проєктів).

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Почніть створювати портфоліо прямо зараз.** Не чекайте, поки ви станете "експертом". Візьміть свої навчальні лабораторні роботи, курсові проєкти, оформіть їх гарно на GitHub. Це вже матеріал для портфоліо.

2. **Адаптуйте резюме під кожну вакансію.** Не надсилайте одне й те саме резюме в усі компанії. Підкреслюйте ті навички та проєкти, які найбільше відповідають вимогам конкретної вакансії.

3. **Будьте чесними.** Не приписуйте собі навичок, яких у вас немає. Це дуже швидко виявиться на співбесіді. Краще чесно сказати: "Я цього ще не знаю, але дуже хочу навчитися".

4. **Посилання на GitHub – обов'язкове.** Переконайтеся, що ваш GitHub профіль виглядає охайно: є фото, заповнена інформація, проєкти мають README-файли. Порожній GitHub може зіграти проти вас.

5. **Практикуйтеся в самопрезентації.** Розкажіть про себе та свої проєкти друзям, викладачам. Чим більше ви практикуєтеся, тим впевненіше почуватиметеся на реальній співбесіді.

6. **Не бійтеся відмов.** Це нормальна частина процесу. Кожна відмова – це досвід і можливість отримати зворотний зв'язок. Аналізуйте, що можна покращити, і рухайтеся далі.

ПРАКТИЧНЕ ЗАНЯТТЯ 15
САМООСВІТА, КАР'ЄРНЕ ПЛАНУВАННЯ ТА РОЗВИТОК У СФЕРІ КОМП'ЮТЕРНИХ НАУК

1. МЕТА ЗАНЯТТЯ

Метою практичного заняття є формування у студентів-першокурсників розуміння необхідності безперервного професійного розвитку в IT-галузі, навичок самостійного навчання, стратегічного кар'єрного планування та вміння орієнтуватися в розмаїтті освітніх ресурсів.

Основні завдання заняття:

1. Ознайомити з концепцією "навчання протягом усього життя" (Lifelong Learning) та її критичною важливістю для IT-фахівця.
2. Дослідити різноманітні ресурси для самоосвіти: онлайн-платформи (Coursera, edX, Prometheus, freeCodeCamp), технічну літературу, професійні блоги та спільноти, подкасти, конференції.
3. Розвинути практичні навички:
   - *кар'єрне планування:* постановка SMART-цілей, створення індивідуального плану розвитку (IDP).
   - *самоосвіта:* оцінка якості та вибір релевантних навчальних ресурсів.
   - *адаптивність:* розуміння необхідності постійного оновлення знань.
4. Розвинути ключові навички саморефлексії, аналітичного мислення, комунікації та здатності до критичної оцінки інформації.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ (ДЛЯ САМОПІДГОТОВКИ)

2.1. Чому самоосвіта є критично важливою в IT?

Сфера інформаційних технологій розвивається з неймовірною швидкістю. Мови програмування, фреймворки, інструменти, методології, які є популярними сьогодні, можуть застаріти через 3-5 років. Університетська освіта дає фундамент, але саме здатність до самоосвіти (self-learning) визначає довгострокову успішність IT-фахівця.

Принцип "Навчання протягом усього життя" (Lifelong Learning): Це усвідомлене, добровільне та постійне прагнення до отримання нових знань і навичок протягом усього професійного життя.

2.2. Джерела для самоосвіти в IT: карта ресурсів

1. Онлайн-курси та платформи:

- Prometheus (Україна): безкоштовні курси українською від провідних викладачів та компаній.
- Coursera, edX: міжнародні платформи з курсами від топових університетів (Stanford, MIT, Harvard). Частина матеріалів безкоштовна (аудит), сертифікати платні.
- freeCodeCamp, The Odin Project: повністю безкоштовні, практично орієнтовані платформи для вивчення веб-розробки.
- Udemy: платформа з величезною кількістю курсів (часто зі знижками). Якість варіюється, тому варто читати відгуки.
- Stepik, Codecademy: інтерактивні платформи для вивчення програмування.

2. **Книги та технічна література:**

- класичні "must read" Книги "Clean Code" Роберта Мартіна, "Code Complete" Стіва Макконнелла, "Design Patterns" Банди Чотирьох, "Грокаємо алгоритми" Адітьї Бхаргави.
- сучасні книги з конкретних технологій (видавництва O’Reilly, Manning, Packt).
- важливо: читати книги в оригіналі (англійською) або в якісних перекладах.

3. **Документація та технічні блоги:**

- Офіційна документація мов програмування та фреймворків (наприклад, python.org, react.dev) – найактуальніше джерело.
- Технічні блоги: freeCodeCamp News, Medium (теги: "programming", "data science", "devops"), Dev.to, Habr, DOU.ua (статті українською).

4. **Відео та подкасти:**

- YouTube-канали: CS50, freeCodeCamp, Traversy Media, Fireship, Lex Fridman Podcast, "Тімоті" (український канал про фронтенд).
- Подкасти: "Фронтенд-подкаст", "Розробка: The Podcast", "Syntax", "Software Engineering Daily". Чудово для навчання під час дороги або спорту.

5. **Професійні спільноти та заходи:**

- Конференції: IT Arena, Devoxx, DevOps Days, DOU Day.
- Мітапи (зустрічі): локальні зустрічі програмістів для обміну досвідом. Шукайте на Meetup.com, в телеграм-каналах.

- Хакатони: змагання, де за короткий час (24-48 год) треба створити працюючий прототип. Це інтенсивне навчання та нетворкінг.
- Open Source: участь у проєктах з відкритим кодом на GitHub. Найкращий спосіб навчитися працювати в команді та читати чужий код.

2.3. Кар'єрне планування в IT

Кар'єрне планування – це процес визначення професійних цілей та шляхів їх досягнення. Важливо не просто "плисти за течією", а усвідомлено будувати свою кар'єру.

SMART-цілі: цілі мають бути:

- Specific (Конкретними): "Вивчити Python" vs. "Пройти курс 'Python для початківців' на Prometheus та написати 3 проєкти".
- Measurable (Вимірними): як дізнатися, що ціль досягнута? (наприклад, "Отримати сертифікат", "Мати 3 проєкти на GitHub").
- Achievable (Досяжними): ціль має бути реалістичною, враховуючи ваші поточні знання та час.
- Relevant (Релевантними): ціль має відповідати вашим довгостроковим кар'єрним планам.
- Time-bound (Обмеженими в часі): Мати чіткий дедлайн (наприклад, "до 1 червня 2025 року").

Індивідуальний план розвитку (Individual Development Plan, IDP): документ, який описує ваші кар'єрні цілі, навички, які потрібно розвинути, конкретні кроки (навчання, проєкти) та ресурси для досягнення цілей.

Кар'єрні шляхи:

- Технічний шлях: Junior -> Middle -> Senior -> Team Lead / Architect.
- Менеджерський шлях: Junior -> Team Lead -> Engineering Manager / Project Manager -> CTO.
- Шлях експерта: Junior -> Middle -> Senior -> Principal Engineer / Fellow.

2.4. Як залишатися в курсі змін?

- Підпишіться на технологічні новини (TechCrunch, The Verge, Wired, DOU).
- Слідкуйте за лідерами думок у LinkedIn та Twitter (X).
- Читайте аналітичні звіти (Gartner, Forrester, State of JS, Stack Overflow Developer Survey).
- Регулярно переглядайте вимоги до вакансій, щоб бачити, які технології набирають популярність.

3. ОБЛАДНАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Для виконання роботи необхідно:

- персональний комп’ютер з доступом до мережі Інтернет.
- для аудиторної роботи: аркуші паперу А4, ручки.
- програмне забезпечення (на вибір):
  - сервіси для створення презентацій (Google Slides, Canva, Microsoft PowerPoint Online).
  - текстовий процесор (Google Docs, Microsoft Word Online) – для створення планів.
  - інструменти для створення ментальних карт (MindMeister, Canva) – для самостійної роботи.
  - браузер для доступу до ресурсів (Coursera, freeCodeCamp, YouTube).

4. ЗАВДАННЯ ДЛЯ АУДИТОРНОЇ РОБОТИ

Завдання 1. Аналіз ресурсів для самоосвіти

Мета завдання: навчитися орієнтуватися в різноманітних джерелах знань для ІТ-фахівця, критично оцінювати їхні переваги та недоліки та обирати найбільш ефективні для власних потреб.

Теоретичне підґрунтя

Інформаційне перевантаження – одна з проблем сучасного світу. Важливо вміти відфільтровувати якісний контент від сміття. Це завдання допоможе вам створити власну "карту ресурсів".

Детальна інструкція виконання

Етап 1. Формування груп та вибір типу ресурсу (5 хвилин).

Об’єднайтесь у групи по 3-4 особи.

Оберіть **один тип ресурсу** для глибокого аналізу. *Приклади:*

- Варіант А: онлайн-платформи з курсами (Coursera, edX, Prometheus, Udemy).
- Варіант Б: технічна література (книги, офіційна документація).
- Варіант В: відеоресурси (YouTube-канали, відеокурси, подкасти).
- Варіант Г: професійні спільноти (конференції, мітапи, Meetup.com, DOU.ua, LinkedIn).

▪ Варіант Д: практичні платформи (freeCodeCamp, The Odin Project, Codecademy, LeetCode, GitHub).

Переконайтеся, що типи не повторюються між групами.

Етап 2. Дослідження ресурсу (10 хвилин).

Використовуйте ваші знання, досвід (якщо є) та швидкий пошук в інтернеті, щоб проаналізувати обраний тип ресурсу. Заповніть таблицю:

Критерій	Ваші висновки
Назва типу ресурсу	(наприклад, "Онлайн-платформи з курсами")
Конкретні приклади	(наприклад, Coursera, Prometheus, Udemy)
Опис ресурсу (що пропонує, для кого)	
Переваги (3-5 пунктів)	(наприклад, структурованість, можливість отримати сертифікат, викладачі з топ-університетів)
Недоліки / Обмеження (3-5 пунктів)	(наприклад, платні сертифікати, відсутність живого спілкування, необхідна самомотивація)
Найкраще застосування (для чого цей ресурс ідеально підходить?)	(наприклад, для вивчення теорії, для отримання сертифіката, для систематизації знань)
Приклад конкретного матеріалу	(назва курсу, книги, каналу)

Етап 3. Створення презентації (5 хвилин).

Створіть 2-3 слайди або постер, які узагальнюють ваше дослідження.

Структура:

▪ *Слайд 1.* Тип ресурсу, приклади (логотипи).

▪ *Слайд 2.* Таблиця "Переваги vs Недоліки".

▪ *Слайд 3.* Рекомендації для студентів: коли і як найкраще використовувати цей ресурс, для яких цілей. Наведіть 1-2 конкретні поради.

Етап 4. Презентація та обговорення (20 хвилин).

Кожна група презентує результати (2-3 хвилини).

Загальна дискусія:

▪ Який тип ресурсу, на вашу думку, є найефективнішим для початківця? Чому?

▪ Як комбінувати різні типи ресурсів для максимального результату? (наприклад, теорія з книги + практика на freeCodeCamp + нетворкінг на мітапі).

▪ Як ви перевіряєте якість інформації в інтернеті?

Критерії оцінювання (Завдання 1):

Повнота аналізу: розглянуто переваги, недоліки, наведено конкретні приклади.

Критичність: виявлено не лише плюси, а й мінуси ресурсу.

Практичність рекомендацій: поради є корисними та реалістичними.

Командна робота.

Якість презентації.

Завдання 2. Створення індивідуального плану розвитку

Мета завдання: навчитися формулювати кар'єрні цілі та розбивати їх на конкретні кроки, використовуючи техніку SMART та отримуючи зворотний зв'язок.

Теоретичне підґрунтя

Мрії стають цілями, коли ми записуємо їх і плануємо шлях досягнення. Це завдання – перший крок до створення вашого професійного компаса.

Детальна інструкція виконання

Етап 1. Вибір професії та підготовка (5 хвилин).

Об'єднайтесь у пари.

Оберіть уявну ІТ-професію, для якої ви будете складати план. *Приклади:*

- Frontend Developer
- Python Backend Developer
- Data Analyst
- DevOps Engineer
- QA Engineer

Етап 2. Складання плану (10 хвилин).

Кожен студент індивідуально (але радячись із партнером) складає чернетку плану розвитку на аркуші паперу або в Google Docs.

Структура плану:

1. Моя цільова професія: [Назва]

2. Моя довгострокова мета (на 3-5 років): (SMART-ціль, наприклад, "Стати Middle Python Developer в продуктовій компанії до кінця 4-го курсу").

3. Мої короткострокові цілі (на 6-12 місяців): (2-3 цілі, кожна має бути конкретною та вимірною).

- *Приклад 1:* "Пройти спеціалізацію 'Python for Everybody' на Coursera до 1 вересня".

- *Приклад 2:* "Написати 2 веб-проєкти на Flask/Django та завантажити їх на GitHub до грудня".
- *Приклад 3:* "Покращити рівень англійської до B2, займаючись на платформі Lingoda 3 рази на тиждень".

4. Ресурси для досягнення: (список конкретних курсів, книг, платформ з посиланнями, які допоможуть досягти кожної цілі).

5. План дій на тиждень: (скільки часу ви плануєте витрачати на навчання щодня/щотижня).

Етап 3. Взаємний зворотний зв'язок (5 хвилин).

Обміняйтеся планами з партнером.

Оцініть план партнера за критеріями SMART: Чи є цілі конкретними? Вимірними? Реалістичними?

Дайте 2-3 конкретні поради, як можна покращити план. Наприклад: "Можливо, варто додати дедлайн до цієї цілі", "А ти впевнений, що зможеш виділяти 20 годин на тиждень? Може, почни з 10?".

Запишіть отримані поради.

Етап 4. Фіксація результатів (5 хвилин).

Запишіть фінальну версію плану (або основні правки) та короткий підсумок зворотного зв'язку (3-5 речень) у спільному документі.

Етап 5. Презентація та обговорення (5 хвилин).

За бажанням, 1-2 пари діляться своїми планами та отриманими порадами.

Загальне обговорення:

- Чи легко було сформулювати реалістичні цілі?
- Що виявилося найскладнішим у плануванні?
- Чому важливо періодично переглядати свій план розвитку?

Критерії оцінювання (Завдання 2):

Реалістичність: цілі виглядають досяжними для студента.

Конкретність: використання SMART-підходу (є дедлайни, конкретні дії).

Якість ресурсів: наведені конкретні, релевантні ресурси.

Зворотний зв'язок: поради є конструктивними та корисними.

Співпраця в парі.

5. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Створення плану самоосвіти та кар'єрного розвитку

Мета завдання: створити власний, персональний план розвитку, який стане дорожньою картою на найближчі роки, враховуючи ваші інтереси, цілі та ресурси.

Вступ. Це підсумкове завдання всього курсу. Ви маєте зібрати всі знання, отримані протягом семестру, і створити документ, який дійсно допоможе вам у майбутньому. Поставтеся до нього серйозно – це ваш особистий план дій.

Детальна інструкція виконання

Крок 1. Вибір професії та самоаналіз.

Оберіть ІТ-професію, яка вас найбільше цікавить (з тих, що ми розглядали на занятті 10). Проведіть короткий самоаналіз:

- Що вам подобається в цій професії?
- Які ваші сильні сторони можуть допомогти в ній?
- Яких знань/навичок вам наразі бракує?

Крок 2. Дослідження вимог ринку.

Знайдіть 3-5 вакансій для початківців (Junior/Trainee) за обраною професією на djinni.co або LinkedIn. Випишіть:

- Які технології є обов'язковими (must have)?
- Які технології є бажаними (nice to have)?
- Які soft skills найчастіше згадуються?

Крок 3. Створення плану розвитку.

Створіть детальний план (у текстовому документі або ментальній карті), який містить наступні розділи:

1. Моя цільова професія та кар'єрна мета:
   - Назва професії.
   - Моя довгострокова мета (на 3-5 років) за SMART.
   - Моя короткострокова мета (на 1 рік) за SMART.
2. План розвитку технічних навичок (Hard Skills):

Складіть список технологій, які вам потрібно вивчити (на основі аналізу вакансій).

Для кожної ключової технології вкажіть конкретний ресурс для вивчення (назва курсу, книги з посиланням) та приблизний термін.

Приклад:

- Технологія: Python (основи)
- Ресурс: Курс "Python для початківців" на Prometheus [посилання].
- Термін: 2 місяці.
- Технологія: Git
- Ресурс: Інтерактивний туторіал "Learn Git Branching" [посилання].
- Термін: 1 тиждень.
- Практика: Проєкт "Консольний калькулятор" на GitHub.

3. **План розвитку м’яких навичок (Soft Skills):**

Оберіть 2-3 soft skills, які є найважливішими для вашої професії та які ви хочете розвинути.

Для кожної навички вкажіть, як ви плануєте її розвивати (наприклад, "брати активну участь у групових проєктах в університеті", "виступати з доповідями", "займатися з викладачем англійської").

4. **Стратегії нетворкінгу та професійної присутності:**

- Що ви зробите для створення професійного профілю? (наприклад, "оформлю LinkedIn профіль до кінця місяця", "почну вести GitHub").
- Які заходи плануєте відвідати? (наприклад, "DOU Day", "IT Arena").
- На які професійні спільноти підпишетесь? (наприклад, "Telegram-канал ‘Dev Ukraine’", "LinkedIn-група ‘Python Ukraine’").

5. **Методи адаптації до змін:**

- Як ви будете слідкувати за новинами в IT? (наприклад, "читати TechCrunch щотижня", "підписатися на розсилку ‘State of Dev’").
- Як часто ви плануєте переглядати та оновлювати цей план? (наприклад, "кожні 6 місяців").

Крок 4. Оформлення та здача.

Обов’язково вкажіть наприкінці роботи список використаних джерел (мінімум 3-4, з гіперпосиланнями на вакансії, курси, статті, книги).

Завантажте готовий файл (PDF, DOCX, JPG, PNG) або надайте посилання на онлайн-документ/ментальну карту з доступом на читання у відповідний розділ на платформі Moodle.

Критерії оцінювання самостійної роботи:

Реалістичність та обґрунтованість: план базується на аналізі реальних вимог ринку та враховує особисті обставини.

Конкретність: усі пункти містять конкретні дії, ресурси, терміни (SMART-підхід).

Повнота: план охоплює технічні навички, м’які навички, нетворкінг, адаптацію до змін.

Практична цінність: наведені ресурси справді корисні та доступні.

Рефлексія: план демонструє усвідомлення власних сильних сторін та зон для розвитку.

Оформлення та джерела: чистота, грамотність, коректні посилання.

6. ПИТАННЯ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ТА САМОПЕРЕВІРКИ

1. Чому концепція "навчання протягом усього життя" (Lifelong Learning) є критично важливою для IT-фахівця?

2. Назвіть 5 різних типів ресурсів для самоосвіти в IT та наведіть приклади кожного.

3. Які переваги та недоліки онлайн-курсів (Coursera, Prometheus) порівняно з читанням технічних книг?

4. Як ви можете використовувати YouTube для ефективного навчання, а не просто для розваги?

5. Що таке SMART-цілі? Розшифруйте абревіатуру та наведіть приклад SMART-цілі для IT-студента.

6. Що таке індивідуальний план розвитку (IDP)? З яких основних частин він складається?

7. Чому важливо періодично переглядати та оновлювати свій кар’єрний план?

8. Як участь у професійних спільнотах (конференції, мітапи, open source) допомагає в розвитку кар’єри?

9. Як ви можете залишатися в курсі останніх технологічних тенденцій?

10. Які три головні кроки ви зробите найближчим часом для реалізації свого плану розвитку?

7. РЕКОМЕНДОВАНІ ІНТЕРНЕТ-ДЖЕРЕЛА

Платформи для навчання:

1. Prometheus (Україна): https://prometheus.org.ua/
2. Coursera: https://www.coursera.org/
3. freeCodeCamp: https://www.freecodecamp.org/
4. The Odin Project: https://www.theodinproject.com/

5. edX: https://www.edx.org/

6. Udemy: https://www.udemy.com/

7. YouTube-канали: freeCodeCamp, Traversy Media, Fireship, CS50, NetNinja.

Ресурси для кар'єрного планування:

8. DOU.ua – Ринок праці: https://dou.ua/

9. Djinni.co – Career Guide: https://djinni.co/

10. LinkedIn Learning: Курси з кар'єрного розвитку.

11. Roadmap.sh: https://roadmap.sh/ – Дорожні карти для різних IT-спеціалізацій (обов'язково до перегляду!).

Ресурси для нетворкінгу:

12. Meetup.com: https://www.meetup.com/

13. LinkedIn: https://www.linkedin.com/

Книги про навчання та кар'єру:

14. Кал Ньюпорт. "Глибока робота".

15. Барбара Оклі. "Навчитися вчитися".

16. Керол Двек. "Гнучка свідомість".

8. ПОРАДИ (ЗАМІСТЬ ВИСНОВКУ)

1. **Ваш план – це живий документ.** Не варто писати його і забути на рік. Повертайтеся до нього щомісяця, аналізуйте прогрес, коригуйте цілі. Життя вносить свої корективи, і це нормально.

2. **Будьте проактивними.** Не чекайте, поки хтось прийде і запропонує вам знання або роботу. Шукайте самі, питайте, пробуйте, помиляйтеся, але не зупиняйтеся.

3. **Створюйте звичку вчитися щодня.** Навіть 30 хвилин на день, присвячені вивченню нової технології, читанню статті або роботі над проєктом, за рік перетворяться на сотні годин і зроблять вас експертом.

4. **Знайдіть однодумців.** Вчитися наодинці важко. Шукайте друзів, які теж захоплені IT, створюйте спільні проєкти, обговорюйте прочитане. Це і мотивація, і нетворкінг.

5. **Святкуйте маленькі перемоги.** Завершили складний курс? Написали перший великий проєкт? Отримали перший коміт на GitHub? Похваліть себе! Це важливі кроки на вашому шляху.

ПІСЛЯМОВА

Завершення цього циклу з 15 практичних занять – це не просто виконання навчального плану, а ваш перший усвідомлений крок у глобальну IT-індустрію. Протягом курсу ви пройшли шлях від розуміння абстрактних алгоритмів та історії обчислювальної техніки до планування власної кар'єри та створення професійного портфоліо.

Ці заняття заклали фундамент вашого професійного світогляду. Ви дізналися, що Комп'ютерні науки – це не лише написання коду, а глибоке розуміння архітектури систем, принципів модульності та взаємодії компонентів. Ви переконалися, що якісний програмний продукт неможливий без ретельного тестування та дотримання міжнародних стандартів якості, таких як ACM та IEEE.

Одним із найважливіших висновків нашої роботи стало розуміння того, що сучасний IT-фахівець – це не «самотній геній», а командний гравець. Опановані вами методології Agile, Scrum та Kanban продемонстрували, як ефективна комунікація та чітке планування дозволяють створювати складні продукти в умовах змін. Ви також усвідомили критичну роль soft skills: емпатії, тайм-менеджменту та етики. Пам'ятайте, що ваші алгоритми та рішення впливають на життя людей, тому професійна відповідальність має бути вашим постійним орієнтиром.

Що далі?

Світ технологій змінюється швидше, ніж будь-яка інша галузь. Штучний інтелект, хмарні обчислення, блокчейн та квантові технології, які ми розглядали як тренди, завтра стануть вашою щоденною роботою. Тому вашим головним інструментом має стати концепція Lifelong Learning – навчання протягом усього життя.

Поради на майбутнє:

1. Продовжуйте практикувати. Ваші лабораторні роботи та «пет-проєкти» на GitHub – це живі докази вашої майстерності.
2. Будуйте нетворкінг. Спілкуйтеся з колегами, відвідуйте конференції та розвивайте свій LinkedIn вже зараз.
3. Шукайте баланс. Поєднуючи навчання з першою роботою чи стажуванням, не забувайте про самовідновлення та запобігання вигоранню.
4. Будьте проактивними. Не чекайте на готові знання – шукайте їх у документації, професійних спільнотах та на онлайн-платформах.

Університетська освіта дала вам «карту знань», але саме ви обираєте маршрут. Не бійтеся складних викликів, адже кожна помилка – це лише крок до глибшого розуміння.

Віримо, що знання та навички, отримані під час цих практичних занять, стануть надійним стартовим майданчиком для вашої успішної кар’єри. Ласкаво просимо до професії!

РЕКОМЕНДОВАНІ ДЖЕРЕЛА

А. Навчальні посібники та підручники

1. Гришанович Т. О., Глинчук Л. Я. Основи об'єктно-орієнтованого програмування : навч. посіб. Луцьк : ВНУ ім. Лесі Українки, 2022. 120 с. URL: https://evnuir.vnu.edu.ua/handle/123456789/20320.

2. Гришанович Т. О. Алгоритми та структури даних : навч. посіб. Луцьк : ВНУ ім. Лесі Українки, 2021. 150 с. URL: https://evnuir.vnu.edu.ua/handle/123456789/19978.

3. Кузьменко І. М., Дацюк О. А. Базові алгоритми та структури даних : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2022. 137 с. URL: https://ela.kpi.ua/handle/123456789/48256.

4. Марченко О. І., Вітковська І. І. Компоненти програмної інженерії. Ч. 1. Вступ до програмної інженерії : лаб. практикум. Київ : КПІ ім. Ігоря Сікорського, 2022. 50 с. URL: https://ela.kpi.ua/items/685a5b22-4c39-40ea-9b2a-3584e3c11256.

5. Глинчук Л. Я., Гришанович Т. О. Програмування : підручник. Луцьк : ВНУ ім. Лесі Українки, 2022. 160 с. URL: https://evnuir.vnu.edu.ua/handle/123456789/20320.

6. Висоцька В. А., Оборська О. В. Python: алгоритмізація та програмування : навч. посіб. Львів : Новий Світ-2000, 2023. 514 с. URL: https://ns2000.com.ua/alhorytmizatsiia-ta-prohramuvannia-python-navchal-nyy-posibnyk/.

7. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Ч. 1 : навч. посіб. Львів : Новий Світ-2000, 2024. 336 с. URL: https://ns2000.com.ua/alhorytmizatsiia-ta-prohramuvannia-chastyna-1-navchal-nyy-posibnyk/.

8. Бакушевич Я. М., Капаціла Ю. Б. Інформатика та комп'ютерна техніка : навч. посіб. / Я. М. Бакушевич, Ю. Б. Капаціла. Львів : Магнолія, 2021. 312 с.

9. Баженов В. А., Шинкаренко Г. А. Інформатика. Комп'ютерна техніка. Комп'ютерні технології: підручник. 2-ге вид. Київ : Каравела, 2023. 496 с. URL: https://caravela.com.ua/index.php?route=product/product&product_id=155.

10. Добровольський Ю. Г. Стандартизація в інженерії програмного забезпечення [Електронний ресурс]: навч. посіб. Чернівці : ЧНУ ім. Ю. Федьковича, 2022. 140 с. URL: https://libguide.sumdu.edu.ua/web/site/category.html?id=117.

11. Трофименко О. Г., Манаков С. Ю., Ларін Д. Г. Основи програмної інженерії : навч.-метод. посіб. Одеса : Фенікс, 2022. URL: https://libguide.sumdu.edu.ua/web/site/category.html?id=117.

Б. Електронні ресурси та інформаційні портали

12. DOU – Спільнота розробників України : портал. URL: https://dou.ua.

13. AIN.UA – Медіа про технології, IT-бізнес та стартапи : портал. URL: https://ain.ua.

14. dev.ua – Медіа про технології та інновації в Україні : портал. URL: https://dev.ua.

15. ITC.ua – Популярний ресурс про інформаційні технології : портал. URL: https://itc.ua.

16. Prometheus – Найбільша платформа онлайн-курсів в Україні : освітня платформа. URL: https://prometheus.org.ua.

17. IT-курси від Prometheus+ : каталог IT-курсів платформи Prometheus. URL: https://prometheus.org.ua/courses-catalog/.

18. Електронний навчально-науковий архів КПІ ім. Ігоря Сікорського (ELAKPI) : репозитарій. URL: https://ela.kpi.ua.

19. Інституційний репозитарій Волинського національного університету імені Лесі Українки : репозитарій. URL: https://evnuir.vnu.edu.ua.

20. Kyivstar Business Hub. Ресурси для розвитку та IT-навчання. *Kyivstar Business Hub*. – 2024. URL: https://hub.kyivstar.ua/articles/11-resursiv-dlya-it-navchannya-ta-rozvitku.

21. Бібліотека факультету комп'ютерних наук та кібернетики КНУ ім. Тараса Шевченка : каталог наукових видань. URL: https://www.csc.knu.ua/uk/library.

22. Кафедра комп'ютерних наук та кібербезпеки ВНУ ім. Лесі Українки. Посібники та підручники : каталог видань кафедри. URL: https://cs.vnu.edu.ua/?page_id=1519.

23. Бібліотека Університету Григорія Сковороди в Переяславі. Нові книги з комп'ютерних наук : бібліографічний огляд. 2022. URL: https://library.uhsp.edu.ua/2022/03/21/biblioteka-informuye-pro-novi-knygy/.

24. Навчально-методичний портал КПІ «Сікорський» : система дистанційного навчання. URL: https://campus.kpi.ua.

25. Путівник по бібліографічних ресурсах з інженерії програмного забезпечення / наук. б-ка СумДУ. Суми: СумДУ, 2022. URL: https://libguide.sumdu.edu.ua/web/site/category.html?id=117.

Навчальне видання

ВСТУП ДО ФАХУ

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.