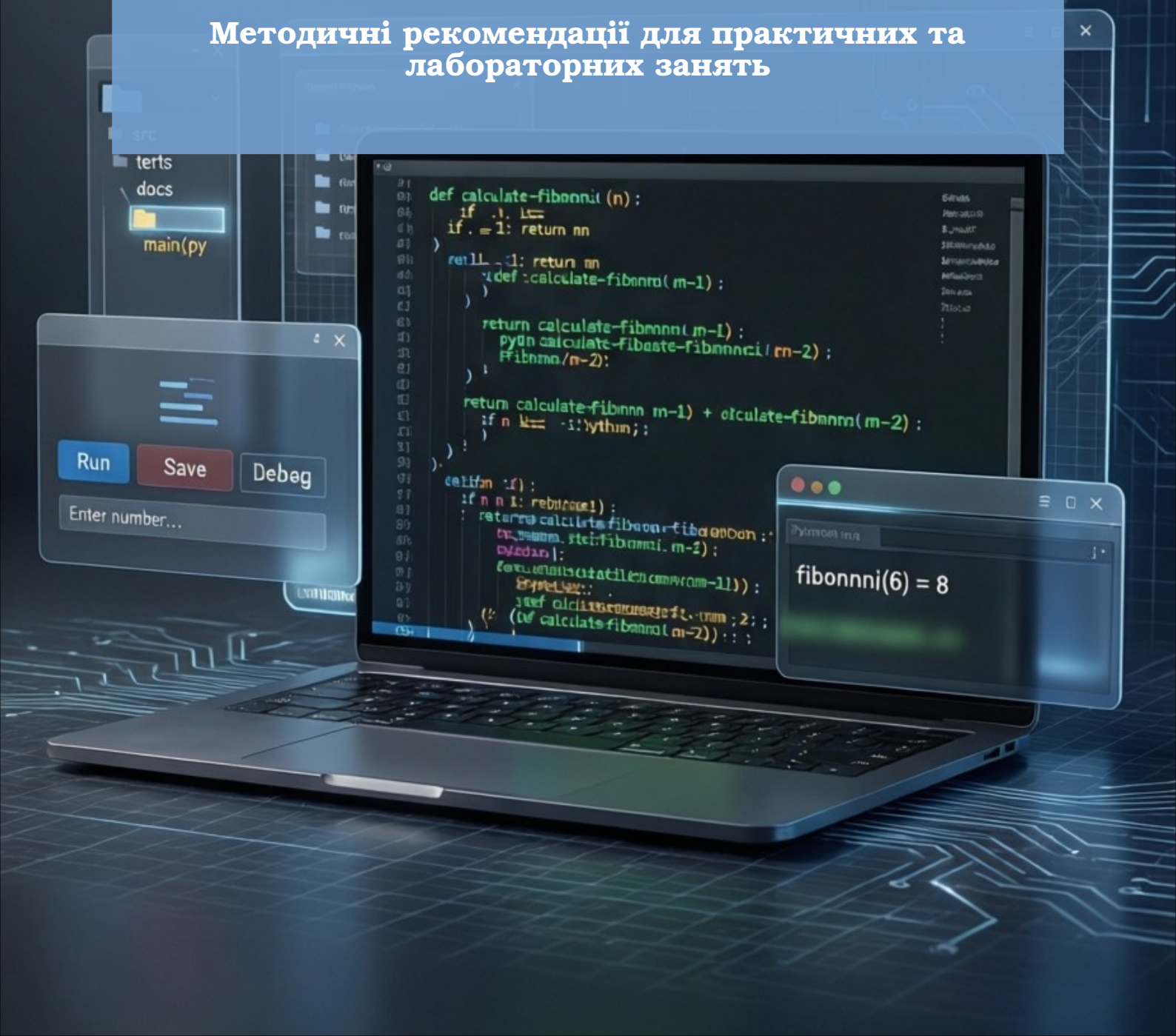
Педяш В.В., Йона Л.Г., Мазур Г.Д., Русу О.П.

PYTHON-ПРОГРАМУВАННЯ

Методичні рекомендації для практичних та лабораторних занять



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Міжнародний гуманітарний університет
Кафедра комп’ютерної інженерії та інноваційних технологій

Педяш В.В., Йона Л.Г., Мазур Г.Д., Русу О.П.

PYTHON-ПРОГРАМУВАННЯ

Методичні рекомендації для практичних та лабораторних занять
здобувачів вищої освіти зі спеціальностей:
A4.09 Середня освіта (Інформатика),
F2 Інженерія програмного забезпечення,
F3 Комп'ютерні науки,
F7 Комп'ютерна інженерія,
F5 Кібербезпека та захист інформації,
G5 Електроніка, електронні комунікації, приладобудування та радіотехніка

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету (протокол № 12 від 23 червня 2025 року)

Педяш В.В., Йона Л.Г., Мазур Г.Д., Русу О.П.

Python-програмування: Методичні рекомендації для практичних та лабораторних занять (частина 1) [Електронне видання] / Педяш В.В., Йона Л.Г., Мазур Г.Д., Русу О.П. — Кафедра комп'ютерної інженерії та інноваційних технологій Міжнародного гуманітарного університету. — Одеса, 2025. — 137 с.

Методичні рекомендації до практичних та лабораторних занять з дисципліни «Python-програмування» розроблено відповідно до навчального плану підготовки здобувачів вищої освіти. Посібник охоплює 14 лабораторних робіт, спрямованих на формування базових і середніх навичок програмування мовою Python — від встановлення та налаштування середовища розробки та використання базових конструкцій мови до об'єктно-орієнтованого програмування, роботи з файлами, візуалізації даних, мережевого програмування, взаємодії з API та створення графічного інтерфейсу користувача.

Матеріали містять методичні вказівки до виконання практичних і лабораторних завдань, приклади програмного коду, вимоги до оформлення звітів і рекомендації щодо тестування програм. Кожна тема передбачає проведення практичного заняття з викладом ключових положень, яке забезпечує підготовку до виконання відповідної лабораторної роботи.

Посібник призначено для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальностями А4.09 Середня освіта (Інформатика), F2 Інженерія програмного забезпечення, F3 Комп'ютерні науки, F7 Комп'ютерна інженерія, F5 Кібербезпека та захист інформації, G5 Електроніка, електронні комунікації, приладобудування та радіотехніка.

Завдання виконуються з використанням сучасних середовищ розробки (PyCharm, Jupyter Notebook), стандартних засобів мови Python та бібліотек прикладного програмування (NumPy, Pandas, Matplotlib, TensorFlow/Keras, requests, socket, tkinter). Такий підхід забезпечує гнучкість навчального процесу та формування практичних навичок програмування й аналізу даних, необхідних для подальшого вивчення сучасних інформаційних технологій і виконання кваліфікаційних робіт.

ЗМІСТ

ВСТУП

Сучасна інженерна та наукова діяльність у галузі інформаційних технологій неможлива без ґрунтовного розуміння принципів програмування та вміння застосовувати їх на практиці. Мова програмування Python завдяки простоті синтаксису, універсальності та розвиненій екосистемі бібліотек стала одним із ключових інструментів для розв'язання широкого спектра прикладних задач – від базових обчислень і обробки даних до мережевого програмування, аналізу даних, штучного інтелекту та створення графічних інтерфейсів користувача. Курс «Python-програмування» спрямований на формування стійкої практичної основи, необхідної для подальшого опанування фахових дисциплін і виконання кваліфікаційних робіт.

Збірник методичних рекомендацій призначено для використання під час проведення практичних і лабораторних занять з дисципліни «Python-програмування» та охоплює 14 навчальних робіт, що послідовно розкривають основні теми курсу: від інсталяції та налаштування середовища розробки, використання математичних функцій і базових конструкцій мови до об'єктно-орієнтованого програмування, роботи з файлами, аналізу та візуалізації табличних даних, мережевих протоколів, взаємодії з API, створення графічних інтерфейсів користувача та застосування елементів нейронних мереж для апроксимації функцій.

Кожна навчальна робота побудована за єдиною логічною структурою, що забезпечує системність і послідовність навчального процесу. Практичне заняття базується на розділі «Ключові положення», у якому викладено необхідні теоретичні відомості, пояснення синтаксису мови та особливостей використання бібліотек, наведено приклади програмного коду та узагальнювальні висновки. Опрацювання цього матеріалу формує цілісне розуміння теми та забезпечує підготовку студентів до виконання практичних завдань.

Лабораторне заняття логічно продовжує практичну частину та виконується відповідно до розділу «Лабораторне завдання». У ньому подано структуровану покрокову інструкцію з виконання конкретних завдань — від отримання вхідних даних за допомогою користувацького введення або зовнішніх джерел (файлів, API) до реалізації алгоритмів, тестування програм і оформлення результатів. Під час виконання робіт передбачається використання сучасних середовищ розробки (PyCharm, Jupyter Notebook), стандартної бібліотеки мови Python і поширених пакетів прикладного та наукового програмування (NumPy, Pandas, Matplotlib, TensorFlow/Keras, requests, socket, tkinter).

Запропонована організація навчального матеріалу забезпечує логічну наступність між теоретичною та практичною складовими курсу: знання, отримані під час практичних занять, закріплюються та поглиблюються у процесі виконання лабораторних робіт. Такий підхід сприяє формуванню

практичних навичок програмування, розвитку інженерного мислення, уміння самостійно аналізувати задачі та обґрунтовано обирати способи їх розв’язання.

Набуті в результаті вивчення дисципліни компетентності створюють підґрунтя для подальшого опанування курсів, пов’язаних із розробкою програмного забезпечення, кібербезпекою, аналізом даних і машинним навчанням, а також для участі у науково-дослідницькій діяльності та виконання випускних кваліфікаційних робіт.

Лабораторна робота № 1

Тема: Інсталяція Pycharm IDE в ОС Windows

Мета: Отримання навичок з інсталяції та налаштування середовища розробки програми Python

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Інтегроване середовище розробки (IDE - Integrated Development Environment) являє собою комплексний програмний продукт, що об'єднує в собі всі необхідні інструменти для повного циклу створення програмного забезпечення. Сучасні IDE включають текстовий редактор з розширеним підсвічуванням синтаксису та підтримкою множини мов програмування, компілятор або інтерпретатор для перетворення вихідного коду в виконувану форму, засоби налагодження (дебагер) для пошуку та усунення помилок, систему автодоповнення коду з контекстними підказками, інтеграцію з системами контролю версій для відстеження змін у коді, засоби профілювання для аналізу продуктивності програм та інструменти для створення графічних інтерфейсів користувача. Використання інтегрованого середовища розробки кардинально підвищує продуктивність програміста завдяки автоматизації рутинних операцій, миттєвому виявленню синтаксичних помилок у реальному часі, інтелектуальному автодоповненню коду з урахуванням контексту, зручним засобам навігації по проекту та можливості швидкого рефакторингу коду. Особливо важливою є можливість інтеграції з різноманітними інструментами розробки, системами збірки проектів та хмарними сервісами, що дозволяє створювати комплексні рішення в рамках єдиного робочого середовища.

PyCharm є одним з найпотужніших професійних кросплатформних IDE, розробленим відомою чеською компанією JetBrains, яка спеціалізується на створенні інструментів для розробки програмного забезпечення. Це середовище спеціально оптимізоване для мови програмування Python та надає розширені можливості для розробки різноманітних Python-додатків, від простих скриптів до складних веб-додатків та систем машинного навчання. PyCharm включає інтелектуальне автодоповнення коду, що аналізує структуру проекту та бібліотеки для надання найбільш релевантних підказок, потужні засоби рефакторингу для безпечної зміни структури коду, комплексний аналіз коду в реальному часі з виявленням потенційних проблем та надання рекомендацій щодо покращення якості коду, підтримку популярних веб-фреймворків Django та Flask з спеціалізованими шаблонами та налаштуваннями, інтеграцію з різноманітними базами даних включаючи PostgreSQL, MySQL, SQLite та MongoDB, а також тісну інтеграцію з системами контролю версій Git, Mercurial та Subversion. Існують дві основні версії PyCharm: Community Edition - безкоштовна версія з базовими функціями, достатніми для розробки чистого Python коду, навчання та створення невеликих проектів, та Professional Edition - платна версія з розширеними можливостями для професійної веб-розробки,

наукових обчислень, роботи з базами даних, підтримкою JavaScript, HTML, CSS та інших веб-технологій. PyCharm також надає унікальні потужні інструменти рефакторингу, включаючи перейменування змінних у всьому проекті, витягування методів та класів, оптимізацію імпортів та автоматичне виправлення стилю коду відповідно до стандартів PEP 8.

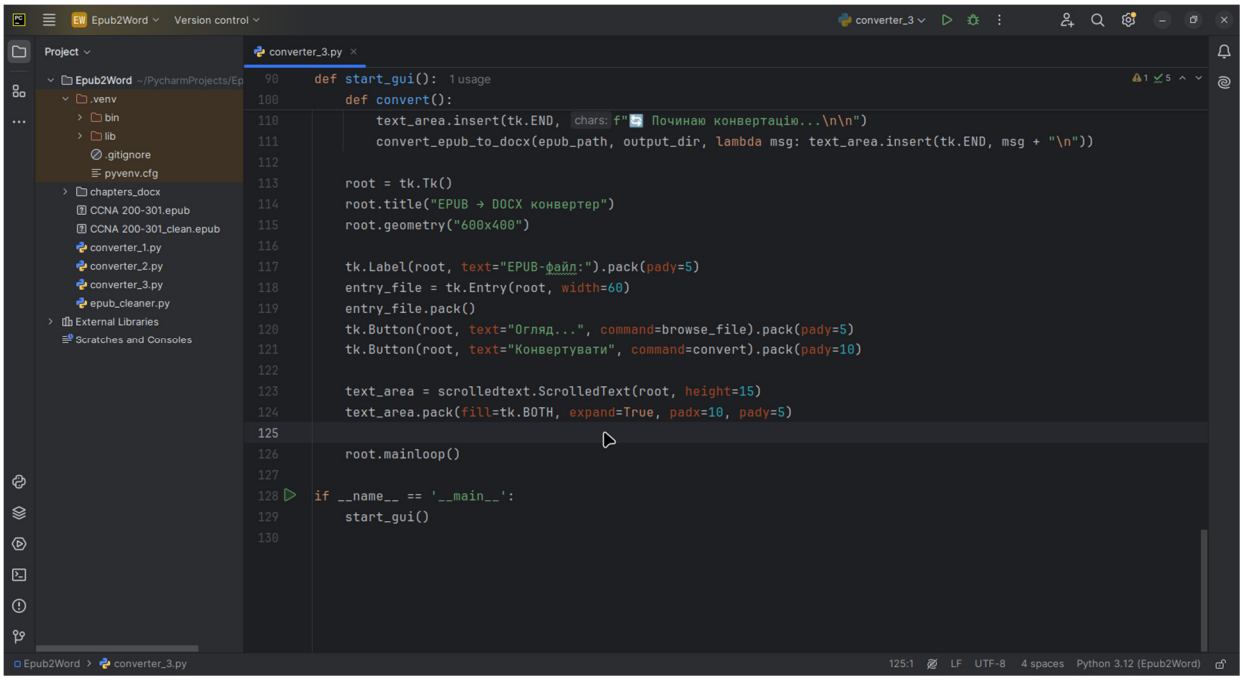


Рисунок 1.1 – Зовнішній вигляд вікна IDE PyCharm

Python як інтерпретована мова програмування має фундаментальну особливість - для виконання Python-програм необхідна наявність спеціального програмного компонента, який називається інтерпретатором Python. На відміну від компільованих мов, де вихідний код перетворюється в машинний код перед виконанням, Python-код інтерпретується та виконується рядок за рядком у момент запуску програми. PyCharm як IDE не містить власного інтерпретатора Python і не може функціонувати без попередньо встановленого в системі інтерпретатора Python, тому встановлення Python є обов'язковою передумовою для роботи з PyCharm. Після встановлення Python та PyCharm, середовище розробки автоматично сканує файлову систему комп'ютера для виявлення всіх встановлених інтерпретаторів Python різних версій та дозволяє гнучко налаштовувати їх використання для конкретних проектів, що особливо важливо при роботі з проектами, які вимагають різних версій Python або специфічних наборів бібліотек. При встановленні Python в операційній системі Windows критично важливо звернути увагу на спеціальну опцію "Add Python to PATH" під час процесу інсталяції, оскільки вона забезпечує глобальну доступність команди python з командного рядка Windows та дозволяє запускати Python-скрипти з будь-якого каталогу системи. Стандартна інсталяція інтерпретатора Python включає не лише сам інтерпретатор, але й повну стандартну бібліотеку Python з сотнями модулів для роботи з файлами, мережею, математичними обчисленнями та іншими завданнями, менеджер пакетів pip для встановлення

додаткових бібліотек з Python Package Index (PyPI), базову документацію та приклади коду. Важливою особливістю Windows є можливість одночасного встановлення та використання декількох версій Python, що дозволяє розробникам працювати з проектами, створеними для різних версій мови.

Процес встановлення програмного забезпечення в операційній системі Windows має усталену методологію, яка базується на використанні спеціальних виконуваних файлів з розширенням .exe, що містять так званий майстер встановлення або інсталятор. Цей майстер являє собою інтерактивний інструмент, який крок за кроком проводить користувача через всі етапи конфігурації програми: вибір каталогу встановлення з можливістю зміни стандартного шляху, створення ярликів на робочому столі та в меню "Пуск" для зручного доступу до програми, автоматичну реєстрацію програми в системному реєстрі Windows, налаштування асоціацій файлів для автоматичного відкриття певних типів файлів у встановленій програмі, а також конфігурацію системних змінних середовища при необхідності. Сучасні інсталятори також включають засоби перевірки системних вимог, автоматичного завантаження додаткових компонентів та створення точок відновлення системи для можливості безпечного видалення програми в майбутньому. Після успішного встановлення як Python, так і PyCharm, необхідно здійснити важливий етап первинного налаштування - конфігурацію інтерпретатора Python для конкретного проекту. PyCharm використовує складний алгоритм автоматичного сканування файлової системи для пошуку всіх встановлених інтерпретаторів Python, включаючи стандартні системні інсталяції, віртуальні середовища та conda-середовища, і представляє їх користувачеві у зручному інтерфейсі для вибору. Правильне налаштування інтерпретатора є критично важливим для забезпечення коректної роботи всіх функцій IDE, включаючи точне автодоповнення коду з урахуванням доступних бібліотек, своєчасну перевірку синтаксису та семантики коду, ефективне виконання та налагодження програм, а також інтеграцію з системами управління пакетами.

Фундаментальною концепцією роботи в PyCharm є організація коду у вигляді проектів, що передбачає логічне об'єднання всіх пов'язаних файлів, ресурсів, конфігурацій та налаштувань в єдину цілісну структуру. Кожен проект в PyCharm має власний кореневий каталог, який містить всі файли проекту та підкаталоги, власні налаштування інтерпретатора Python з можливістю вибору конкретної версії та віртуального середовища, індивідуальну конфігурацію залежностей і зовнішніх бібліотек через файли requirements.txt або pipfile, персоналізовані параметри запуску та налагодження програм, налаштування стилю коду та форматування, а також конфігурацію інтеграції з системами контролю версій. Така організація дозволяє повністю ізолювати різні проекти один від одного, запобігаючи конфліктам між різними версіями бібліотек та налаштуваннями, і забезпечує максимальну зручність управління навіть найскладнішими програмними системами з множиною компонентів та залежностей. Python-програми традиційно зберігаються у текстових файлах з розширенням .py, які містять вихідний код, написаний строго відповідно до синтаксису мови Python з дотриманням правил відступів

та структури. Для виконання програми PyCharm використовує попередньо налаштований інтерпретатор Python, передаючи йому повний шлях до файлу програми разом з необхідними параметрами командного рядка, а результати виконання програми, включаючи текстовий вивід, повідомлення про помилки та діагностичну інформацію, відображаються в спеціальній консолі виконання, розташованій у нижній частині інтерфейсу IDE та обладнаній засобами для взаємодії з програмою під час її виконання.

Комплексне використання PyCharm як професійного інтегрованого середовища розробки надає розробникам Python значні переваги у продуктивності та якості створюваного коду. Автоматизація великої кількості рутинних операцій, таких як форматування коду, оптимізація імпортів, автоматичне додавання необхідних імпортів та виправлення простих помилок, дозволяє програмістам зосередитися на вирішенні основних завдань замість витрачання часу на технічні деталі. Розвинена система підказок та автодоповнення коду з контекстним аналізом не лише прискорює процес написання коду, але й зменшує кількість помилок завдяки перевірці типів даних та валідації синтаксису в реальному часі. Потужні засоби навігації по проекту, включаючи швидкий пошук файлів, класів, методів та змінних, дозволяють ефективно орієнтуватися навіть в найскладніших проектах з тисячами файлів. Інтегровані засоби контролю версій забезпечують безшовну роботу з Git та іншими системами, дозволяючи відстежувати зміни, створювати гілки, виконувати злиття та вирішувати конфлікти прямо в середовищі розробки без необхідності використання зовнішніх інструментів.

2 КЛЮЧОВІ ПИТАННЯ

1. Що таке інтегроване середовище розробки (IDE) та які основні компоненти воно включає?
2. Чому необхідно встановлювати Python окремо від PyCharm і яка послідовність інсталяції цих компонентів?
3. Які існують версії PyCharm та в чому полягають основні відмінності між Community Edition та Professional Edition?
4. Яку роль відіграє опція "Add Python to PATH" при встановленні Python в Windows і чому вона є важливою?
5. Як PyCharm виявляє встановлені в системі інтерпретатори Python та як налаштувати інтерпретатор для конкретного проекту?
6. Що являє собою концепція проекту в PyCharm та які компоненти входять до структури проекту?
7. Які основні етапи проходить майстер встановлення (installer) при інсталяції програм в Windows?
8. Як створити новий Python-файл у проекті PyCharm та запустити програму на виконання?
9. Які переваги надає використання IDE порівняно з простими текстовими редакторами для розробки програм?

10. Де відображаються результати виконання Python-програми в інтерфейсі PyCharm та які додаткові можливості надає консоль виконання?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Для початку роботи з мовою Python необхідно встановити інтерпретатор Python в операційну систему. Слід враховувати, що PyCharm є лише інтегрованим середовищем розробки (IDE) і не містить власного інтерпретатора, а використовує зовнішній Python для виконання програм. Тому спочатку встановлюється Python, а вже після цього — середовище PyCharm. Під час першого запуску PyCharm автоматично знаходить встановлений інтерпретатор і пропонує використовувати його для створення та запуску проєктів. Такий порядок встановлення забезпечує коректну роботу середовища та спрощує керування різними версіями Python.

3.1 Завантаження та встановлення Python у Windows

3.1.1 Щоб завантажити та встановити Python, відвідайте офіційний веб-сайт Python https://www.python.org/downloads/ і виберіть **свою** версію. Вибираємо останню актуальну версію Python 3.6.3.

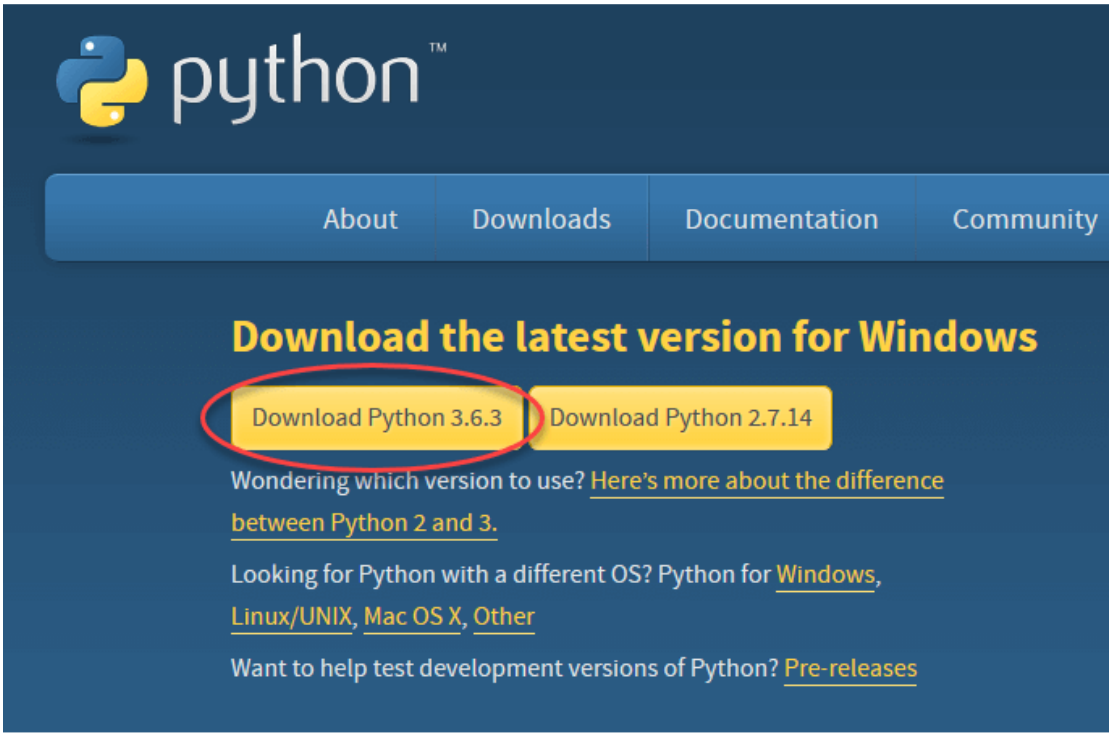


Рисунок 1.2 – Вибір версії Python на офіційному сайті для завантаження під ОС Windows

3.1.2 Після завершення завантаження запустіть файл .exe, щоб інсталювати Python. Тепер натисніть "**Install Now**" (Встановити зараз).

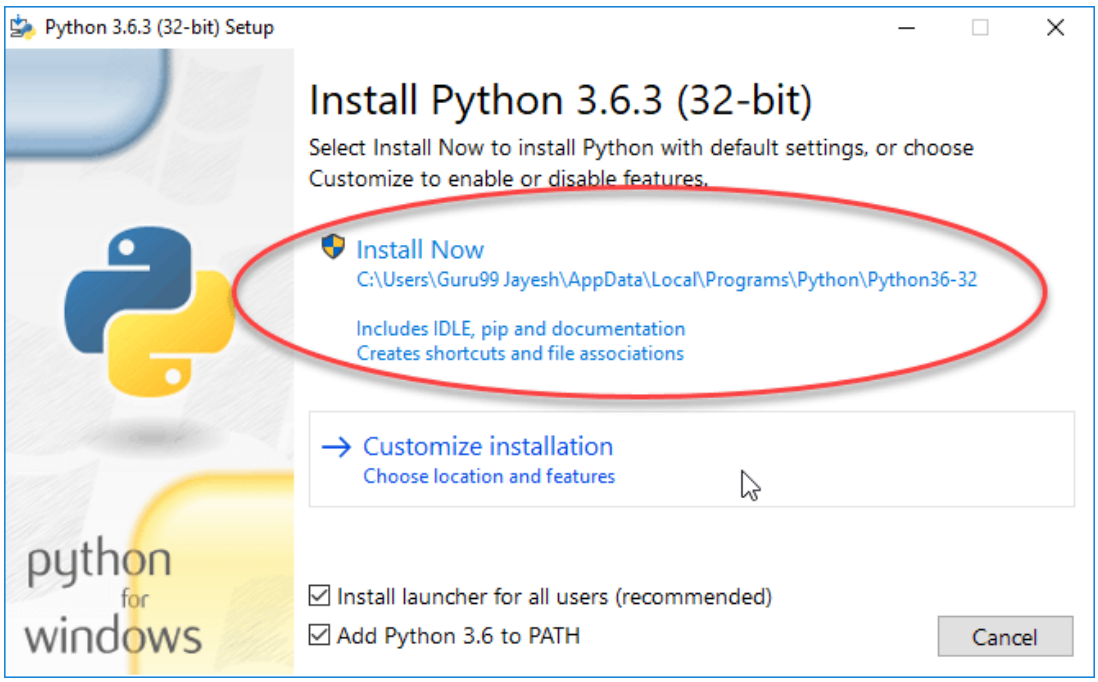


Рисунок 1.3 – Вибір типу установки Python у вікні інсталяції

3.1.3 На цьому етапі ви можете побачити встановлення Python.

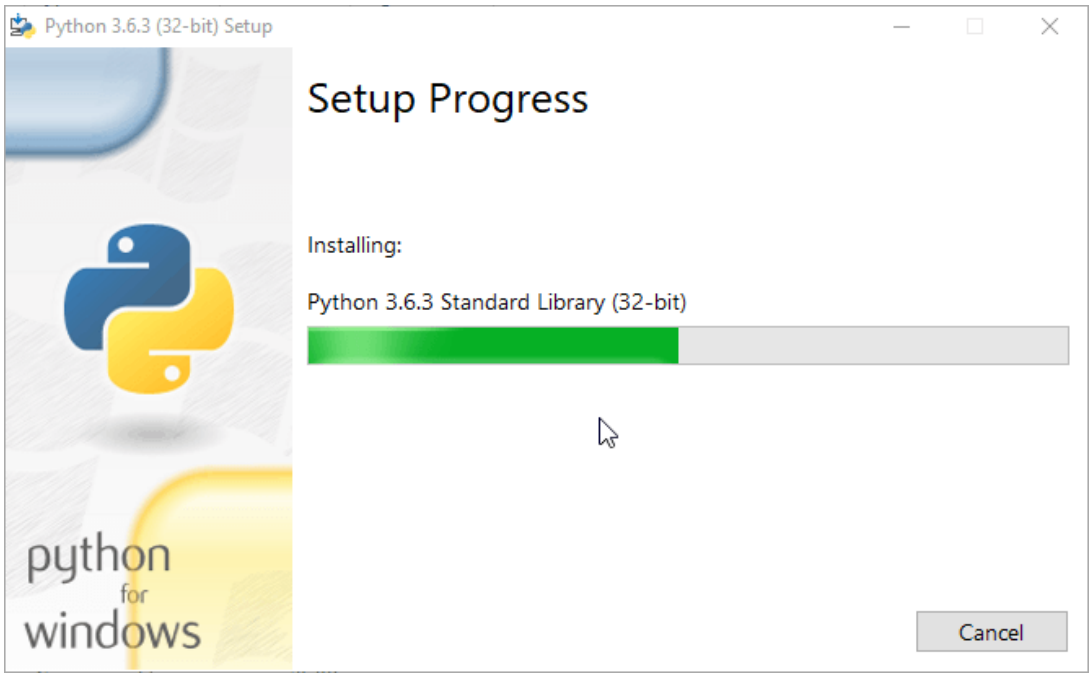


Рисунок 1.4 – Процес інсталяції інтерпретатора Python

3.1.4 Після завершення ви побачите екран із повідомленням про успішну інсталяцію. Тепер натисніть “**Close**” (Закрити).

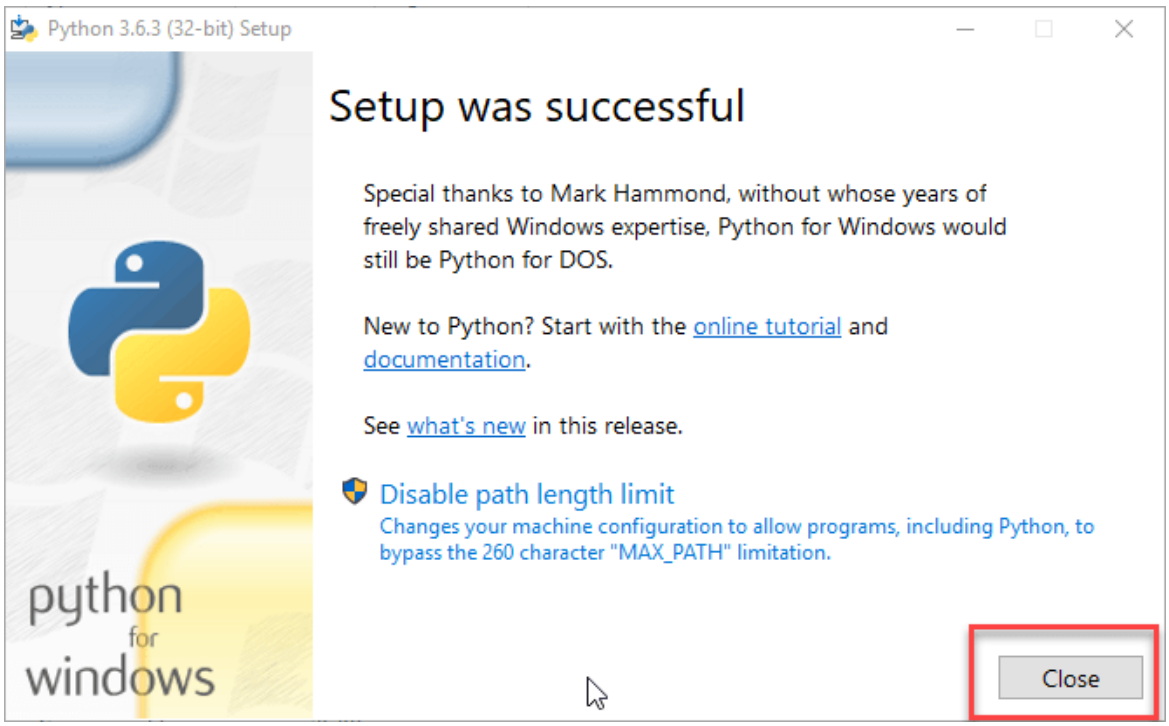


Рисунок 1.5 – Вікно завершення інсталяції інтерпретатора Python

3.2 Інсталяція Pycharm IDE у Windows

3.2.1 Щоб завантажити PyCharm, відвідайте веб-сайт https://www.jetbrains.com/pycharm/download/ і натисніть посилання “DOWNLOAD” (ЗАВАНТАЖИТИ) » в розділі "Community" (Спільнота).

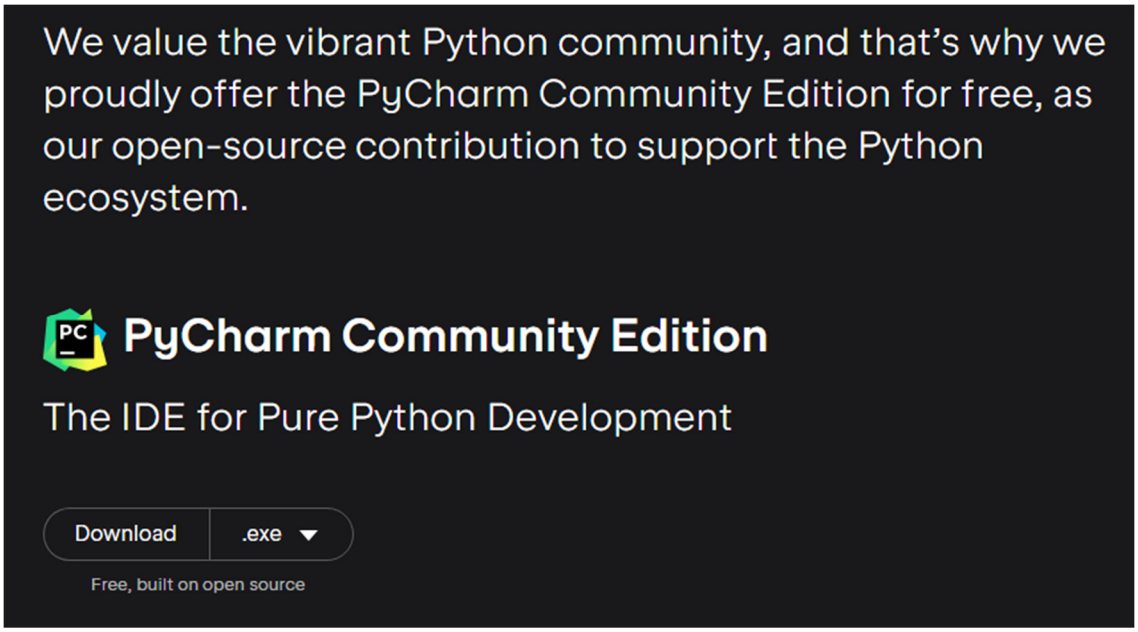


Рисунок 1.6 – Фрагмент вікна сторінки сайту для завантаження PyCharm

3.2.2 Після завершення завантаження, запустіть exe для встановлення PyCharm. Має запуститися майстер налаштування. Натисніть "Next" (Далі).

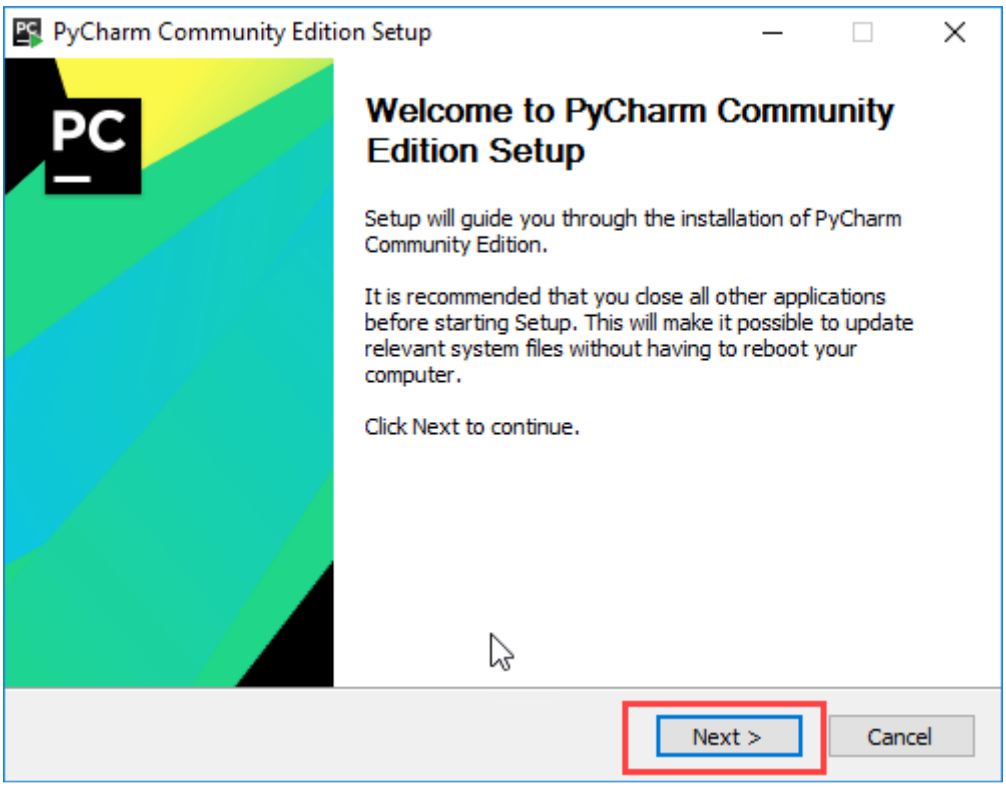


Рисунок 1.7 – Початкове вікно інсталятора PyCharm

3.2.3 На наступному екрані змініть шлях встановлення, якщо потрібно. Натисніть "Next" (Далі).

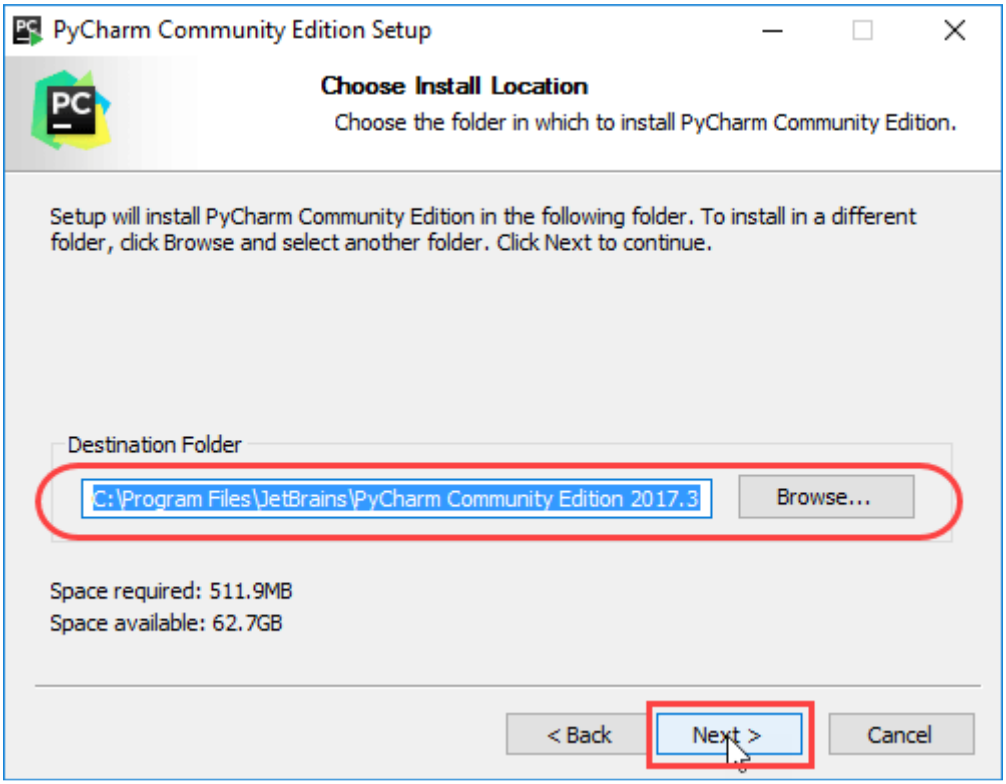


Рисунок 1.8 – Вибір каталогу для встановлення IDE PyCharm

3.2.4 На наступному кроці ви можете створити ярлик на робочому столі, якщо хочете, і натисніть "Next" (Далі).

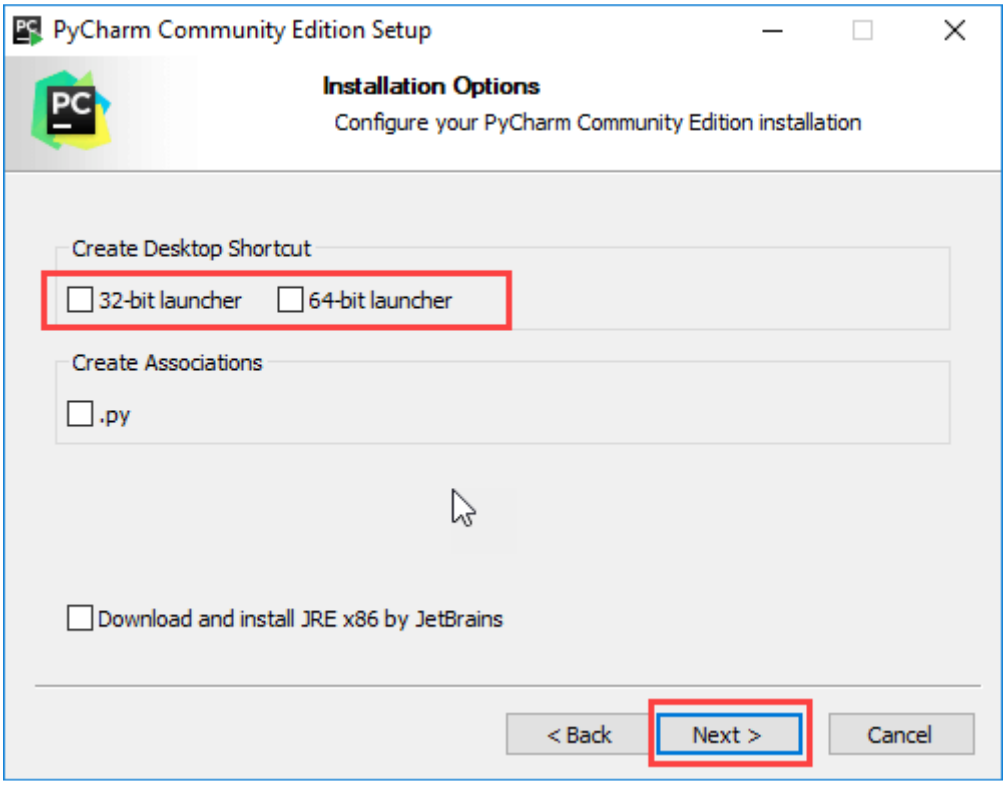


Рисунок 1.9 – Вікно вибору створення ярликів

3.2.5 Вибір папки в меню «Пуск». Збережіть запропоновану назву папки "JetBrains" і натисніть " Install" (Встановити).

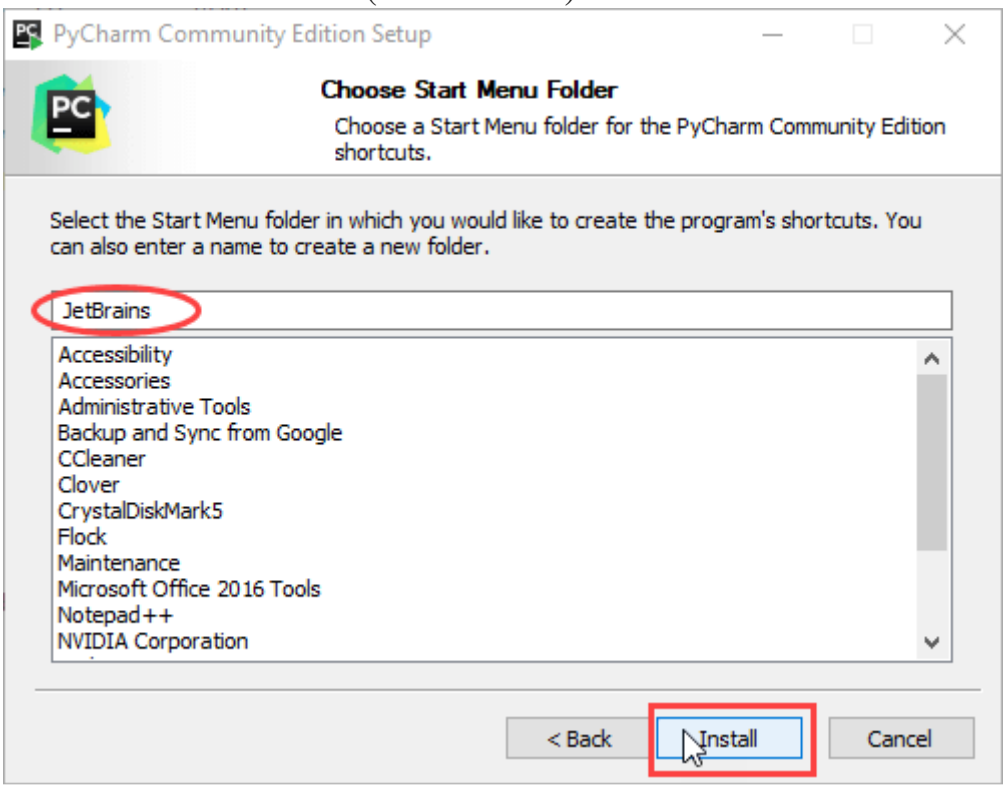


Рисунок 1.10 – Створення групи JetBrains в меню ОС

3.2.6 Дочекайтеся закінчення встановлення.

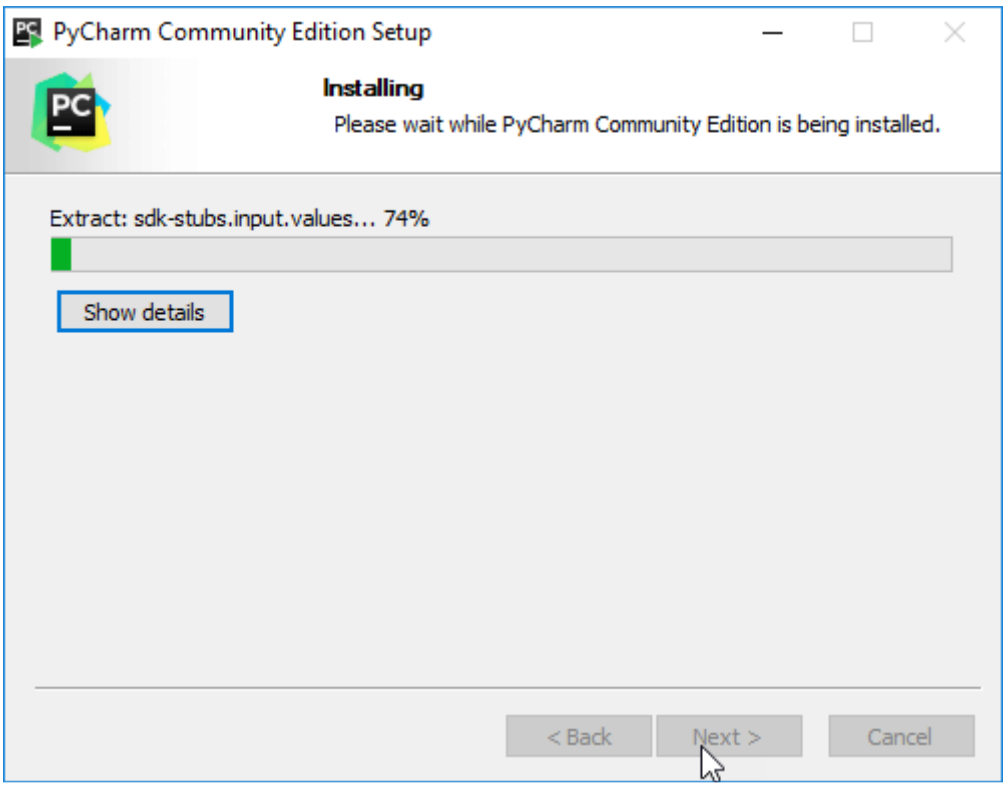


Рисунок 1.11 – Процес інсталяції IDE PyCharm

3.2.7 Після завершення встановлення ви повинні отримати екран повідомлення про те, що PyCharm встановлено. Якщо ви хочете продовжити та запустити його, спочатку натисніть поле " Run PyCharm Community Edition" (Запустити PyCharm Community Edition), а потім натисніть "Finish" (Готово).

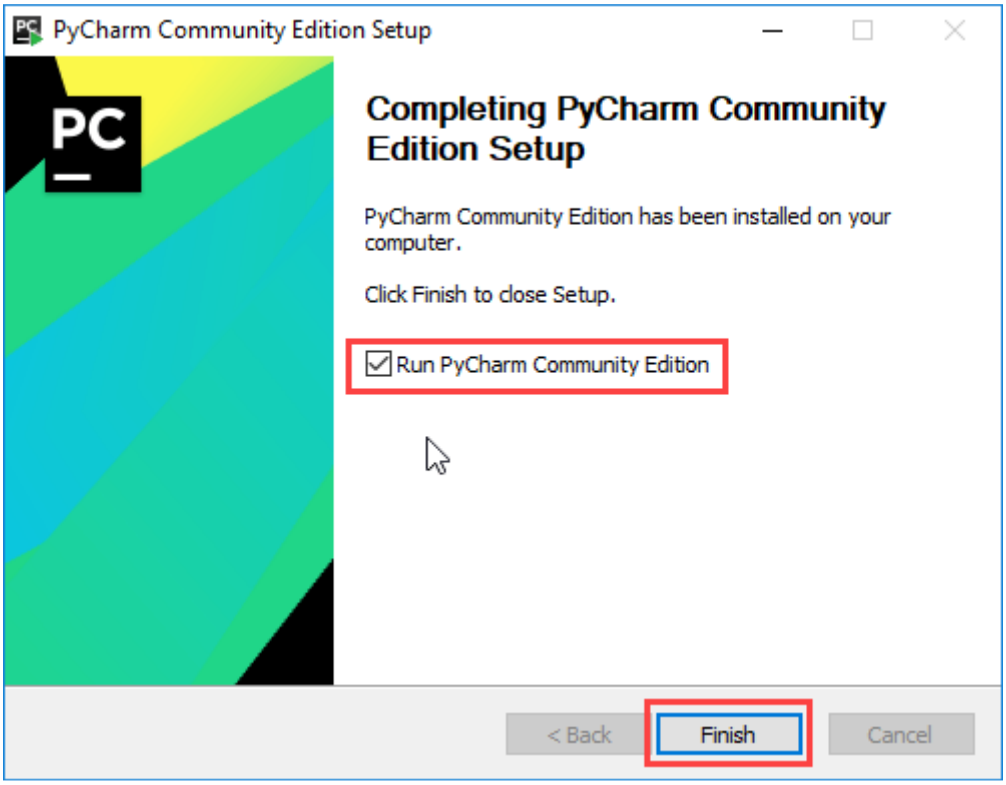


Рисунок 1.12 – Вікно закінчення процесу встановлення IDE PyCharm

3.2.8 Після натискання кнопки «Готово» з’явиться наступний екран.

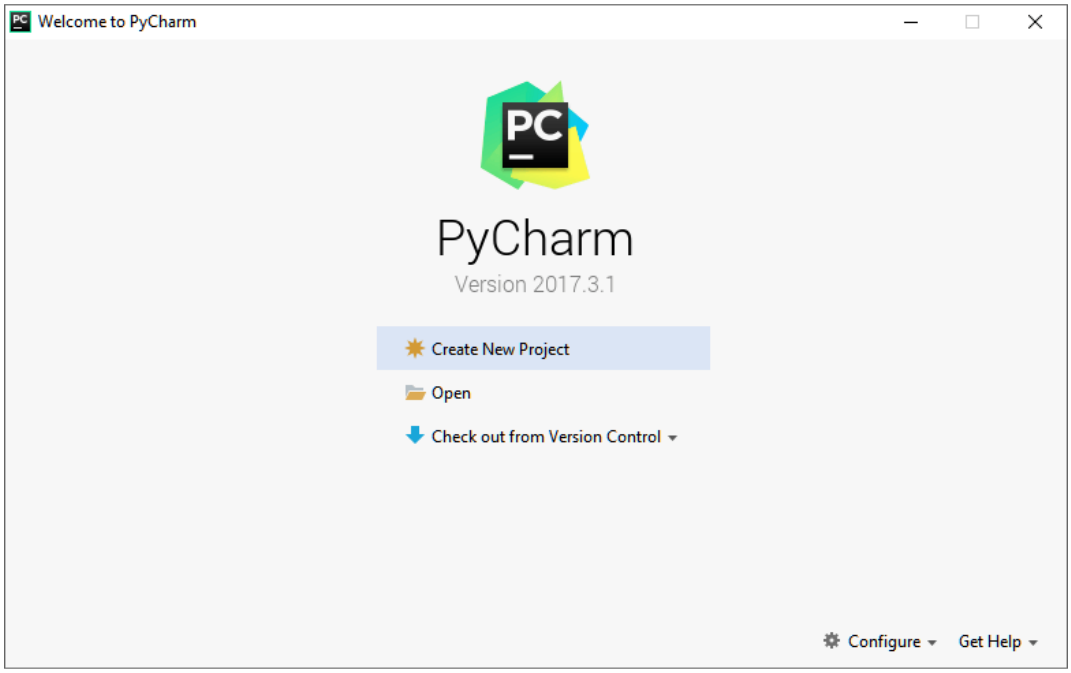


Рисунок 1.13 – Початкове вікно IDE PyCharm

3.3 Створення проекту в PyCharm

3.3.1 Відкрийте редактор PyCharm. Ви можете побачити вступний екран для PyCharm. Щоб створити новий проект, натисніть "Create New Project" (Створити новий проект).

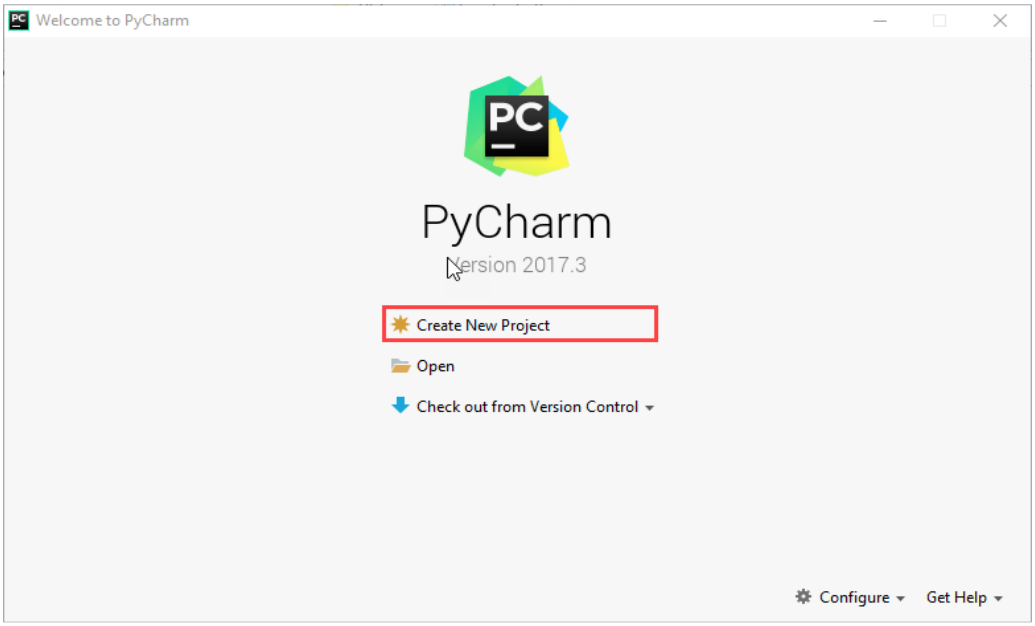


Рисунок 1.14 – Створення нового проекту

3.3.2 Після запуску PyCharm у стартовому вікні натисніть кнопку «Create New Project». У вікні створення проєкту (рис. 1.15) в полі Location ① задайте шлях до каталогу, у якому буде збережено проєкт. За замовчуванням пропонується папка PyCharmProjects, однак за потреби можна обрати будь-яке

інше місце. Обов'язково змініть стандартну назву проєкту untitled на унікальну, наприклад FirstProject.

У розділі Project Interpreter ② перевірте, чи вибрано коректний інтерпретатор Python, встановлений раніше (наприклад, python.exe з каталогу Python36-32). Якщо інтерпретатор не було визначено автоматично, виберіть його вручну.

Після завершення налаштувань натисніть кнопку «Create» ③, щоб створити проєкт.

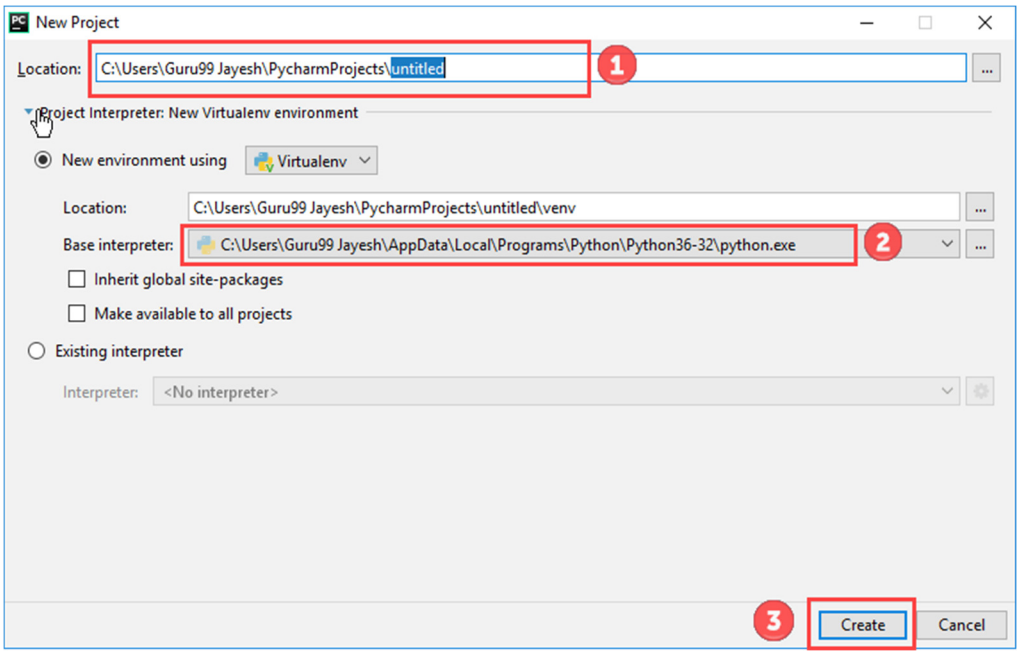


Рисунок 1.15 – Вибір опцій нового проекту

3.3.3 Тепер перейдіть до меню "File" (Файл) і виберіть "New". Далі виберіть "Python File" (Файл Python).

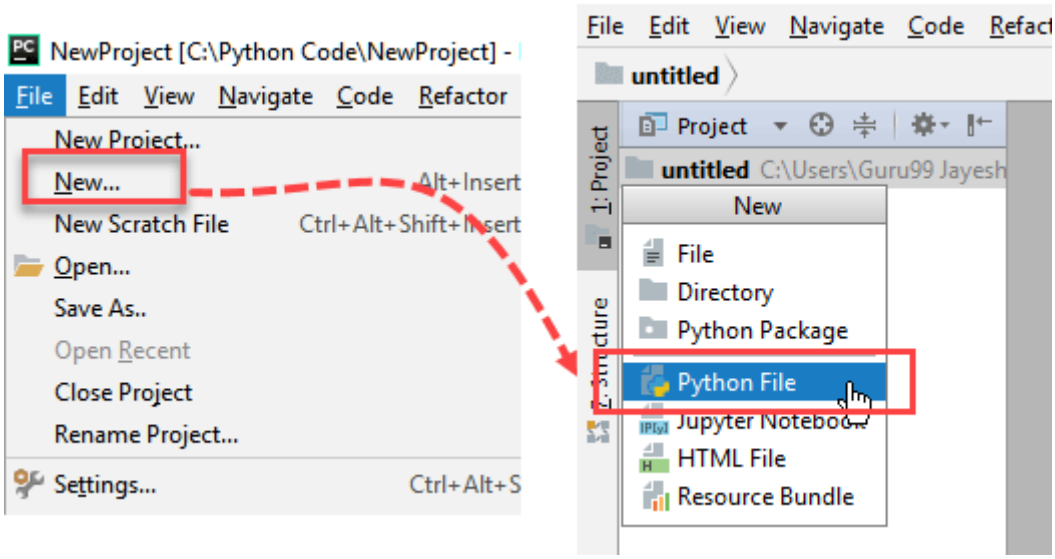


Рисунок 1.16 – Створення файла скрипта Python

3.3.4 З’явиться нове спливаюче вікно. Тепер введіть назву потрібного файлу (тут ми даємо «HelloWorld») і натисніть «ОК».

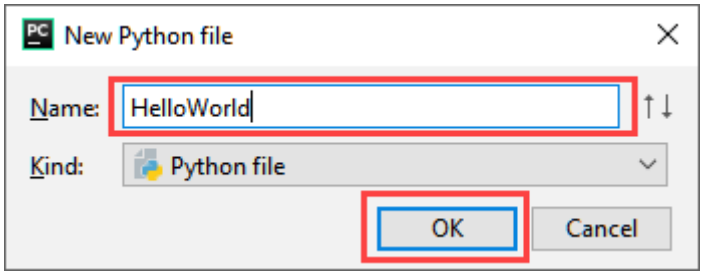


Рисунок 1.17 – Вікно вводу назви файла

3.3.5 Тепер наберіть просту програму – print («Hello World!»).

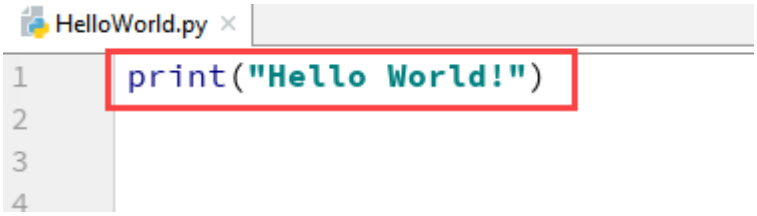


Рисунок 1.18 – Текст першої програми

3.3.6 Тепер перейдіть до меню "Run" (Виконати) та виберіть "Run", щоб запустити програму.

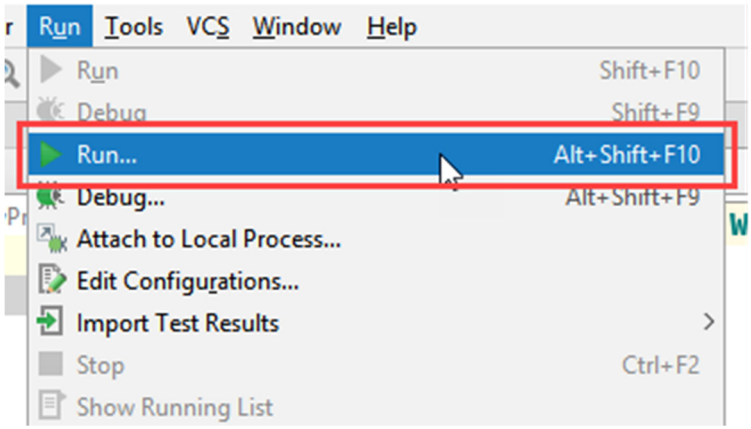


Рисунок 1.19 – Виконання програми у PyCharm через меню Run

3.3.7 Ви можете побачити результат вашої програми внизу екрана.

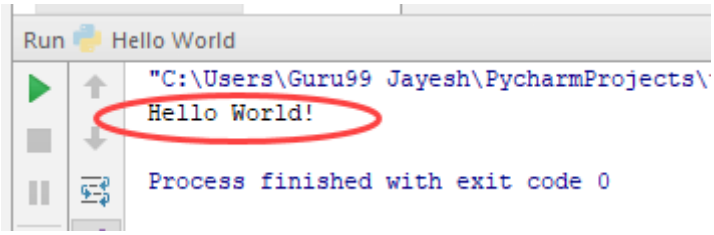


Рисунок 1.20 – Виведення результату виконання програми у консолі PyCharm

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У цій частині послідовно описано етапи виконання лабораторного завдання: завантаження та інсталяція інтерпретатора Python, завантаження та встановлення середовища PyCharm Community Edition, створення нового проекту з налаштуванням інтерпретатора, розробка та запуск першої програми print("Hello, World!"), а також перевірка працездатності середовища через автодоповнення, підсвічування синтаксису та налагоджувач. Кожен крок задокументовано скріншотами інтерфейсу та виводу консолі.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов'язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 2

Тема: Робота з математичними функціями та виразами в Python

Мета: Навчитися використовувати математичні функції з модуля math, працювати з користувацьким введенням та виводом результатів обчислень.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Python як мова програмування надає потужні засоби для виконання математичних обчислень завдяки вбудованому модулю math, який містить широкий спектр математичних функцій та констант. Модуль math є частиною стандартної бібліотеки Python і не вимагає додаткового встановлення, але потребує явного імпорту в програму за допомогою інструкції import math або через селективний імпорт конкретних функцій. Цей модуль реалізує математичні функції, визначені стандартом C, та забезпечує високу точність обчислень з числами з плаваючою комою. Використання модуля math дозволяє виконувати складні математичні операції, які неможливо реалізувати за допомогою базових арифметичних операторів Python, включаючи тригонометричні функції, логарифми, степеневі функції, факторіали та операції з константами як π та e.

Основні категорії функцій модуля math включають тригонометричні функції для роботи з кутами та періодичними залежностями, такі як sin(), cos(), tan() для основних тригонометричних відношень, asin(), acos(), atan() для обернених тригонометричних функцій, а також degrees() та radians() для конвертації між градусами та радіанами. Логарифмічні та експоненціальні функції представлені функціями log() для натурального логарифму, log10() для десяткового логарифму, exp() для експоненційної функції та pow() voor степеневих операцій з підвищеною точністю. Функції для роботи з цілими числами включають factorial() для обчислення факторіала, gcd() для знаходження найбільшого спільного дільника та функції округлення ceil(), floor() та trunc(). Модуль також надає важливі математичні константи, такі як math.pi для числа π, math.e для основи натурального логарифму та math.inf для представлення нескінченності.

Робота з користувацьким введенням в Python здійснюється за допомогою вбудованої функції input(), яка зупиняє виконання програми та очікує введення даних користувачем з клавіатури. Функція input() завжди повертає рядок (тип str), незалежно від того, що ввів користувач, тому для виконання математичних обчислень необхідно явно перетворювати введені дані до відповідних числових типів за допомогою функцій int() для цілих чисел або float() для чисел з плаваючою комою. Правильна обробка користувацького введення вимагає врахування можливих помилок введення, таких як введення нечислових значень або порожніх рядків, що може призвести до виникнення винятків ValueError або TypeError під час спроби конвертації типів. Для забезпечення

надійності програми рекомендується використовувати конструкції обробки винятків try-except або попередню валідацію введених даних.

Вивід результатів обчислень в Python реалізується за допомогою вбудованої функції print(), яка може приймати множину аргументів різних типів та автоматично перетворює їх у рядкове представлення для відображення на екрані. Функція print() підтримує різноманітні параметри форматування, включаючи sep для визначення роздільника між аргументами, end для зміни символу завершення рядка та file для перенаправлення виводу в інший потік. For більш точного контролю над форматуванням числових результатів можна використовувати метод format() рядків або f-рядки (f-strings), які дозволяють задавати кількість знаків після коми, ширину поля виводу та інші параметри відображення. Особливо важливим є контроль точності виводу для чисел з плаваючою комою, оскільки математичні обчислення можуть давати результати з великою кількістю знаків після коми, що ускладнює читання результатів.

Математичні вирази в Python можуть включати комбінації арифметичних операторів, функцій з модуля math та дужок для визначення порядку обчислень. Python дотримується стандартного порядку виконання математичних операцій: спочатку виконуються функції та операції в дужках, потім степеневі операції, далі множення та ділення зліва направо, і нарешті додавання та віднімання зліва направо. При роботі з складними математичними виразами важливо правильно розставляти дужки для забезпечення коректного порядку обчислень та уникнення логічних помилок. Використання змінних для збереження проміжних результатів обчислень може покращити читабельність коду та спростити налагодження програми.

Типи даних у математичних обчисленнях Python включають цілі числа (int), числа з плаваючою комою (float) та комплексні числа (complex). Цілі числа мають необмежену точність у Python 3, тоді як числа з плаваючою комою використовують стандарт IEEE 754 подвійної точності і можуть мати обмеження в точності представлення десяткових дробів. При змішуванні різних числових типів в обчисленнях Python автоматично виконує приведення типів згідно з ієрархією: int → float → complex. Розуміння особливостей роботи з числами з плаваючою комою критично важливе для уникнення помилок округлення та забезпечення точності математичних обчислень.

Обробка помилок при математичних обчисленнях може включати різні типи винятків: ZeroDivisionError при діленні на нуль, ValueError при передачі некоректних аргументів функціям (наприклад, логарифм від від'ємного числа), OverflowError при перевищенні максимально допустимих значень для чисел з плаваючою комою. Деякі математичні функції можуть повертати спеціальні значення, такі як math.nan (не число) або math.inf (нескінченність), які потребують окремої обробки в програмі. Розуміння цих особливостей дозволяє створювати більш надійні програми, здатні коректно обробляти крайні випадки та повідомляти користувачу про проблеми у вхідних даних або обчисленнях.

Таблиця 2.1 – Функції модуля math в Python

Функція	Опис	Приклад використання
Тригонометричні функції		
math.sin(x)	Синус кута в радіанах	math.sin(math.pi/2)→ 1.0
math.cos(x)	Косинус кута в радіанах	math.cos(0)→ 1.0
math.tan(x)	Тангенс кута в радіанах	math.tan(math.pi/4)→ 1.0
math.asin(x)	Арксинус (результат в радіанах)	math.asin(1)→ 1.5707963267948966
math.acos(x)	Арккосинус (результат в радіанах)	math.acos(0)→ 1.5707963267948966
math.atan(x)	Арктангенс (результат в радіанах)	math.atan(1)→ 0.7853981633974483
math.degrees(x)	Переведення радіанів в градуси	math.degrees(math.pi)→ 180.0
math.radians(x)	Переведення градусів в радіани	math.radians(180)→ 3.141592653589793
Логарифмічні функції		
math.log(x)	Натуральний логарифм (ln)	math.log(math.e)→ 1.0
math.log(x, base)	Логарифм з основою base	math.log(100, 10)→ 2.0
math.log10(x)	Десятковий логарифм	math.log10(1000)→ 3.0
math.log2(x)	Двійковий логарифм	math.log2(8)→ 3.0
math.exp(x)	Експонента (e^x)	math.exp(1)→ 2.718281828459045
Степеневі функції		
math.pow(x, y)	Піднесення x до степеня y	math.pow(2, 3)→ 8.0
math.sqrt(x)	Квадратний корінь	math.sqrt(16)→ 4.0
math.isqrt(x)	Цілочисельний квадратний корінь	math.isqrt(16)→ 4
Функції округлення		
math.ceil(x)	Округлення вгору	math.ceil(4.2)→ 5
math.floor(x)	Округлення вниз	math.floor(4.8)→ 4
math.trunc(x)	Відсікання дробової частини	math.trunc(4.8)→ 4
Факторіали		
math.factorial(x)	Факторіал числа	math.factorial(5)→ 120
math.gamma(x)	Гамма-функція	math.gamma(5)→ 24.0
Найбільший спільний дільник		
math.gcd(a, b)	Найбільший спільний дільник	math.gcd(12, 18)→ 6
Абсолютне значення		
math.fabs(x)	Абсолютне значення (float)	math.fabs(-5.5)→ 5.5
Додаткові функції		
math.hypot(x, y)	Гіпотенуза прямокутного трикутника	math.hypot(3, 4)→ 5.0
math.fmod(x, y)	Залишок від ділення (float)	math.fmod(10.5, 3)→ 1.5
math.modf(x)	Відокремлення цілої та дробової частин	math.modf(3.14)→ (0.14000000000000012, 3.0)
Константи		
math.pi	Число π	math.pi→ 3.141592653589793
math.e	Основа натурального логарифма	math.e→ 2.718281828459045
math.tau	Подвійне значення π (2π)	math.tau→ 6.283185307179586
math.inf	Нескінченність	math.inf→ inf
math.nan	Не число (Not a Number)	math.nan→ nan

2 КЛЮЧОВІ ПИТАННЯ

1. Як імпортувати модуль math в Python?
2. Які основні математичні функції містить модуль math?
3. Як отримати числові дані від користувача за допомогою функції input()?
4. Як перетворити текстові дані від користувача в числа з плаваючою комою?
5. Який порядок виконання математичних операцій в Python?
6. Як правильно вивести результати обчислень за допомогою функції print()?
7. Які типи даних використовуються в математичних обчисленнях Python?
8. Які математичні винятки можуть виникнути під час обчислень?
9. Як форматувати вивід чисел з плаваючою комою?
10. Як забезпечити правильну роботу програми при некоректному введенні даних користувачем?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Оберіть один із варіантів завдань з наведеної нижче таблиці відповідно до вашого номера у списку.
2. Напишіть програму на Python, яка буде обчислювати значення заданого математичного виразу.
3. Програма повинна запитувати у користувача значення змінних x та y за допомогою функції input().
4. Використовуйте необхідні функції з модуля math для обчислення виразу.
5. Виведіть результат обчислення на екран за допомогою функції print().

Таблиця 2.2 – Варіанти індивідуальних завдань

Варіант	Завдання	Варіант	Завдання
1	$\frac{\tan(x)+y^3}{\cos(x-y)\cdot\sqrt[3]{x+y}}$	16	$\frac{\log_2(x+y)+\cos(xy)}{\sinh(x-y)\cdot\sqrt{x^2+y^2}}$
2	$\frac{e^x+\ln(y)}{x\cdot y+\sin(x-y)}$	17	$\frac{\log_{10}(xy)\cdot\tanh(x+y)}{x^2-y^2+\sqrt[3]{x+y}}$
3	$\frac{\sqrt{x^2+y^2}}{\arctan(x/y)+x\cdot y}$	18	$\frac{\arcsin(x/10)\cdot y^{1/3}+\ln(x+y)}{x\cdot\tan(y)-\sqrt{x^2-y^2}}$
4	$\frac{x^y+\cos(x)}{\sinh(y)-x^2}$	19	$\frac{\cos(x/y)+e^{xy}}{\sinh(x^2)-\sqrt{y}}$

5	$\frac{\log_{10}(x+y)\cdot\sqrt[4]{x-y}}{e^{x/y}+\arcsin(x/10)}$	20	$\frac{x^y-\cosh(x-y)}{\log_2(xy)+\sqrt[4]{x+y}}$
6	$\frac{\sin(x+y)\cdot\ln(x^2+y^2)}{x^3-y^3+\sqrt{xy}}$	21	$\frac{e^x\cdot\sin(y)+\log_{10}(xy)}{\tan(x-y)+\sqrt{x+y}}$
7	$\frac{\arccos(x/10)+y^{1/3}}{x\cdot\log(y)+e^{x-y}}$	22	$\frac{\sin(x+y)\cdot e^{x-y}}{x^3+y^3-\arctan(y/x)}$
8	$\frac{x\cdot\sinh(y)+\sqrt{x^2-y^2}}{\tan(x+y)-\ln(xy)}$	23	$\frac{\cosh(x/y)-\sqrt[3]{x^2-y^2}}{x^y+\arcsin(x/10)}$
9	$\frac{\cos(x^2)+y^3}{\arctan(y/x)\cdot\sqrt[3]{x+y}}$	24	$\frac{\sqrt{xy}\cdot\tanh(x^2)}{\cos(y)+\log(x-y)}$
10	$\frac{e^{xy}-\sin(x+y)}{x^2+y^2+\log(x-y)}$	25	$\frac{x^2\cdot\ln(y)+\tanh(x+y)}{e^{xy}-\sqrt[4]{x-y}}$
11	$\frac{\sqrt{x+y}\cdot\tanh(x-y)}{x^y+\arcsin(y/10)}$	26	$\frac{x\cdot\arccos(y/10)-y^{1/5}}{\sinh(x+y)+\sqrt[3]{x^2+y^2}}$
12	$\frac{x^3\cdot\arccos(y/10)+\tan(x+y)}{e^x+y^2-\sqrt{xy}}$	27	$\frac{\log_2(x-y)+\sin(x^2+y^2)}{x\cdot\cosh(y)+\sqrt[5]{xy}}$
13	$\frac{\sin(x^2+y^2)+\ln(x-y)}{x\cdot\cosh(y)+\sqrt[4]{xy}}$	28	$\frac{\sqrt{x+y}\cdot\cos(xy)}{\sinh(x^2)+y^3-\arctan(x/y)}$
14	$\frac{(x+y)^3-\cosh(x-y)}{x\cdot y+\tan(x/y)}$	29	$\frac{\sqrt{x}\cdot y^3+\arctan(x/y)}{e^{x-y}-\sin(xy)}$
15	$\frac{\cosh(x)-y^{1/4}}{x\cdot\sin(y)+\sqrt[3]{xy}}$	30	$\frac{(x^2+\sin(y))\cdot\sqrt{x-y}}{(x-y)^3+\log(x+y)}$

Приклад коду python

```
# Введення значень x та y
x = float(input("Введіть значення x: "))
y = float(input("Введіть значення y: "))

# Обчислення виразу (приклад для варіанту 1)
result = ((x**2 + math.sin(y)) * math.sqrt(x-y)) / ((x-y)**3 + math.log(x+y))

# Виведення результату
print(f"Результат обчислення: {result}")
```

Не забудьте замінити формулу у прикладі на ту, що відповідає вашому варіанту завдання.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи № 2 студент реалізує програму на мові Python, яка обчислює значення індивідуального математичного виразу залежно від введених користувачем значень змінних x та y. Програма використовує функції з модуля math, отримує вхідні дані через input(), перетворює їх у числовий формат і виводить результат за допомогою print(). Код супроводжується коментарями та оформлений згідно зі стандартами читабельності.

Результати виконання задокументовано на основі принаймні трьох тестових прикладів з різними значеннями x та y, для кожного з яких наведено вхідні дані, отриманий результат і скріншот виконання. У завершальній частині проводиться аналіз отриманих результатів: перевіряється їх відповідність очікуваним значенням, обговорюються можливі розбіжності або особливості, пов'язані з областю визначення функцій або обмеженнями чисел з плаваючою комою.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов'язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 3

Тема: Розробка програми з використанням умовного оператора if-elif-else в Python

Мета: Навчитися застосовувати оператор if-elif-else для розгалуження програми залежно від вхідних даних, працювати з математичними функціями та користувацьким введенням.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Оператори розгалуження в мові програмування Python відіграють ключову роль у створенні інтелектуальних програм, які можуть приймати рішення на основі аналізу вхідних даних. Вони дозволяють програмі виконувати різні шляхи обчислень залежно від певних умов, що робить код гнучким та адаптивним до різних ситуацій. Без операторів розгалуження програми були б лінійними та жорстко визначеними, не здатними реагувати на зміни вхідних параметрів або зовнішніх умов. У реальному світі більшість процесів мають умовний характер: якщо температура вища за певний поріг, увімкнути кондиціонер; якщо баланс менший за нуль, заборонити операцію зняття коштів. Саме оператори розгалуження дозволяють закладати таку логіку прийняття рішень у програмний код, роблячи програми практично корисними та інтелектуальними.

Умовний оператор if-elif-else є однією з ключових конструкцій мови Python, яка дозволяє реалізовувати розгалуження в програмах. Він використовується для виконання різних блоків коду залежно від істинності або хибності певних умов. Оператор if перевіряє першу умову, і якщо вона істинна, виконується відповідний блок коду. Якщо ж умова хибна, перевіряється наступна умова, вказана в elif. Якщо жодна з умов не виконується, виконується блок else. Така конструкція дозволяє гнучко керувати логікою програми, особливо коли потрібно обробляти різні діапазони значень або типи даних. Використання кількох elif дозволяє створювати складні структури прийняття рішень, де кожна гілка відповідає певному діапазону або конкретному значенню вхідних даних.

Для реалізації програми з розгалуженням важливо правильно організувати умови, щоб вони не перетиналися і охоплювали всі можливі випадки. Умови повинні перевірятися послідовно, і кожна наступна умова має сенс лише тоді, коли попередні були хибними. У багатьох задачах, особливо при роботі з математичними функціями, вхідні дані потрібно перевіряти на належність до певних інтервалів. Для цього використовуються логічні вирази, що містять оператори порівняння, наприклад, <, <=, >, >=, ==, !=. Також важливо враховувати, що вирази в elif перевіряються лише тоді, коли попередні умови були False. Правильна організація умов забезпечує коректну роботу програми і запобігає ситуаціям, коли жодна з умов не виконується або

виконується декілька умов одночасно, що може призвести до непередбачуваного результату.

Робота з користувацьким введенням у Python вимагає особливої уваги, оскільки функція input() завжди повертає рядок. Тому перед використанням у числових обчисленнях необхідно перетворити введене значення до числового типу, наприклад, float або int. У випадку роботи з математичними функціями, такими як sqrt, sin, cos, log тощо, потрібно підключити модуль math за допомогою інструкції import math. При використанні функцій з цього модуля слід пам'ятати про область визначення — наприклад, не можна обчислювати квадратний корінь з від'ємного числа без комплексних чисел. У разі некоректного введення або виходу за область визначення функції програма може викликати помилку, тому важливо передбачати перевірки або обробку винятків. Також слід враховувати, що користувач може ввести некоректні дані, тому рекомендується додавати додаткові перевірки на валідність введених значень.

Виведення результатів у зручному форматі забезпечується за допомогою функції print(), якій можна передавати форматовані рядки за допомогою f-рядків. Це дозволяє вставляти значення змінних прямо в текст повідомлення, що робить вивід більш зрозумілим для користувача. Також важливо пам'ятати про типи даних, що виводяться, і при необхідності виконувати округлення чисел або форматування виводу. Використання умовного оператора разом із функціями введення-виведення та математичними обчисленнями дозволяє створювати повноцінні програми, які можуть аналізувати вхідні дані та давати коректну відповідь у залежності від умов. Такий підхід є основою для побудови більш складних алгоритмів та додатків. У реальному програмуванні оператори розгалуження використовуються практично в кожному проекті, від простих калькуляторів до складних систем прийняття рішень, що робить їх незамінним інструментом у арсеналі програміста.

Приклад коду

```
import math

# Введення значення x
x = float(input("Введіть значення x: "))

# Обчислення функції (приклад для варіанту 1)
if x < -5:
    result = 2 * x**2 - 3
elif -5 <= x < 5:
    result = math.sqrt(x + 10)
else:  # x >= 5
    result = math.sin(x) + 1

# Виведення результату
print(f"Результат обчислення: {result}")
```

2 КЛЮЧОВІ ПИТАННЯ

1. Яке призначення умовного оператора if-elif-else в мові Python та як він реалізує розгалуження програми?
2. Як відбувається перевірка умов у конструкції if-elif-else і в якому порядку виконуються блоки коду?
3. Які оператори порівняння використовуються в умовах та як правильно формувати логічні вирази для перевірки діапазонів значень?
4. Як здійснюється робота з користувацьким введенням даних через функцію input() та які перетворення типів необхідні для числових обчислень?
5. Як підключається та використовується модуль math для обчислення математичних функцій у програмах з умовними операторами?
6. Які особливості області визначення математичних функцій потрібно враховувати при використанні умовних операторів?
7. Як правильно організувати структуру умов, щоб уникнути перетину діапазонів та забезпечити повне покриття можливих вхідних даних?
8. Як здійснюється форматований вивід результатів обчислень за допомогою f-рядків та функції print()?
9. Які можливі помилки можуть виникнути при роботі з умовними операторами та як їх уникнути (наприклад, ділення на нуль, вихід за межі області визначення)?
10. Як умовні оператори інтегруються з іншими елементами програми, такими як цикли, функції та структури даних, для створення складних алгоритмів?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Оберіть один із варіантів завдань з наведеної нижче таблиці відповідно до вашого номера у списку.
2. Напишіть програму на Python, яка буде обчислювати значення функції залежно від введеного значення x.
3. Програма повинна запитувати у користувача значення змінної x за допомогою функції input().
4. Використовуйте оператор if-elif-else для визначення відповідного проміжку та обчислення відповідного виразу.
5. Використовуйте необхідні функції з модуля math для обчислення виразу.
6. Виведіть результат обчислення на екран за допомогою функції print().

Таблиця 3.1 – Варіанти індивідуальних завдань

Варіант	Завдання
1	$y=\begin{cases}2x^2+5x-3, & \text{якщо} x<-10\\ 3x-7, & \text{якщо} -10\le x\le 10\\ x^3-4x+5, & \text{якщо} x>10\end{cases}$
2	$y=\begin{cases}x^2-2x+1, & \text{якщо} x<-5\\ 4x+9, & \text{якщо} -5\le x\le 15\\ 2x^3-6, & \text{якщо} x>15\end{cases}$
3	$y=\begin{cases}x^3+2x^2-4, & \text{якщо} x<-12\\ 5x-3, & \text{якщо} -12\le x\le 8\\ x^2-5x+8, & \text{якщо} x>8\end{cases}$
4	$y=\begin{cases}\frac{x^3}{2}+x, & \text{якщо} x<-8\\ 2x^2-4, & \text{якщо} -8\le x\le 6\\ \frac{x^2}{3}+7, & \text{якщо} x>6\end{cases}$
5	$y=\begin{cases}x^2+3x-1, & \text{якщо} x<-15\\ 6x+2, & \text{якщо} -15\le x\le 5\\ x^3-x^2+3, & \text{якщо} x>5\end{cases}$
6	$y=\begin{cases}\sin(x), & \text{якщо} x<-7\\ x^2-2x, & \text{якщо} -7\le x\le 7\\ 3x^3+x, & \text{якщо} x>7\end{cases}$
7	$y=\begin{cases}\ln(\lvert x\rvert+1), & \text{якщо} x<-12\\ 4x-5, & \text{якщо} -12\le x\le 4\\ \frac{x}{2}+3, & \text{якщо} x>4\end{cases}$
8	$y=\begin{cases}x^2+4, & \text{якщо} x<-18\\ 5x-1, & \text{якщо} -18\le x\le 2\\ \frac{1}{x+1}+x^3, & \text{якщо} x>2\end{cases}$

9	$y=\begin{cases} x^3-x^2+5, & \text{якщо} x<-3 \\ 3x^2+x, & \text{якщо} -3\le x\le 12 \\ x^4+x-4, & \text{якщо} x>12 \end{cases}$
10	$y=\begin{cases} 2x+9, & \text{якщо} x<-14 \\ \dfrac{x^2}{2}-3, & \text{якщо} -14\le x\le 6 \\ x^3+2x-1, & \text{якщо} x>6 \end{cases}$
11	$y=\begin{cases} \dfrac{x^3}{3}+5, & \text{якщо} x<-10 \\ 7x-4, & \text{якщо} -10\le x\le 8 \\ 3x^2+1, & \text{якщо} x>8 \end{cases}$
12	$y=\begin{cases} e^x, & \text{якщо} x<-20 \\ x^2+x, & \text{якщо} -20\le x\le 0 \\ x^3-7, & \text{якщо} x>0 \end{cases}$
13	$y=\begin{cases} \ln(x^2+1), & \text{якщо} x<-6 \\ 4x+2, & \text{якщо} -6\le x\le 9 \\ 2x^3-5x, & \text{якщо} x>9 \end{cases}$
14	$y=\begin{cases} \cos(x)+x, & \text{якщо} x<-11 \\ 3x-1, & \text{якщо} -11\le x\le 7 \\ x^2+2x+3, & \text{якщо} x>7 \end{cases}$
15	$y=\begin{cases} x^2+3x-5, & \text{якщо} x<-13 \\ 5x+7, & \text{якщо} -13\le x\le 10 \\ x^4-x^2+2, & \text{якщо} x>10 \end{cases}$
16	$y=\begin{cases} \dfrac{x^2}{x+1}, & \text{якщо} x<-16 \\ 6x^2-9x, & \text{якщо} -16\le x\le 3 \\ x^3-5x+1, & \text{якщо} x>3 \end{cases}$

17	$y=\begin{cases}\log_2(\lvert x\rvert+1), & \text{якщо} x<-4\\ 2x^3+3x, & \text{якщо} -4\le x\le 8\\ x^2+7x-2, & \text{якщо} x>8\end{cases}$
18	$y=\begin{cases}x^3+6x-3, & \text{якщо} x<-7\\ 5x^2+1, & \text{якщо} -7\le x10\\ 2x-4, & \text{якщо} x>10\end{cases}$
19	$y=\begin{cases}4x+7, & \text{якщо} x<-9\\ \frac{x^3}{2}-x+6, & \text{якщо} -9\le x\le 5\\ x^2-3x+8, & \text{якщо} x>5\end{cases}$
20	$y=\begin{cases}\frac{x^3}{4}-x, & \text{якщо} x<-17\\ 3x+5, & \text{якщо} -17\le x0\\ x^2+4x+9, & \text{якщо} x>0\end{cases}$
21	$y=\begin{cases}x^2-7x+4, & \text{якщо} x<-19\\ 6x+3, & \text{якщо} -19\le x2\\ x^3+x^2-6, & \text{якщо} x>2\end{cases}$
22	$y=\begin{cases}\cos(x^2), & \text{якщо} x<-6\\ 4x^3-3x+2, & \text{якщо} -6leqx9\\ x^2+1, & \text{якщо} x>9\end{cases}$
23	$y=\begin{cases}2x^3+5x^2, & \text{якщо} x<-8\\ 7x-1, & \text{якщо} -8x7\\ x^4+x-3, & \text{якщо} x>7\end{cases}$
24	$y=\begin{cases}x^2+3, & \text{якщо} x<-18\\ 5x+4, & \text{якщо} -18x2\\ x^3-4x, & \text{якщо} x>2\end{cases}$
25	$y=\begin{cases}x^4-2x^2+1, & \text{якщо} x<-15\\ 3x+2, & \text{якщо} -15x6\\ 6x^2+9x-7, & \text{якщо} x>6\end{cases}$

26	$y=\begin{cases}4x^3-x^2+6, & \text{якщо} x<-3\\ x^2+x-5, & \text{якщо} -3x12\\ 7x+1, & \text{якщо} x>12\end{cases}$
27	$y=\begin{cases}\frac{x^3}{5}-x, & \text{якщо} x<-11\\ 3x^2+4x, & \text{якщо} -11x8\\ x+2, & \text{якщо} x>8\end{cases}$
28	$y=\begin{cases}\log_{10}(x^2+1), & \text{якщо} x<-7\\ 4x+3, & \text{якщо} -7x10\\ x^2-6x, & \text{якщо} x>10\end{cases}$
29	$y=\begin{cases}2x+5, & \text{якщо} x<-9\\ \frac{x^2}{2}-3x, & \text{якщо} -9x5\\ x^3+2x+1, & \text{якщо} x>5\end{cases}$
30	$y=\begin{cases}3x^2+7, & \text{якщо} x<-12\\ 5x^2+2x, & \text{якщо} -12x10\\ x+1, & \text{якщо} x>10\end{cases}$

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

Основною частиною протоколу є лістинг програми, який повинен містити повний, добре структурований код розробленої програми на Python з

належними коментарями. При цьому особлива увага має приділятися правильному використанню умовного оператора if-elif-else та функцій з модуля math. У розділі результатів виконання **необхідно навести щонайменше чотири приклади роботи програми: по одному прикладу для кожного з трьох проміжків та один приклад для граничного значення між проміжками, причому для кожного прикладу мають бути вказані введене значення *x*, отриманий результат та проміжок, до якого воно належить, а також додані скріншоти виконання програми.**

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 4

Тема: Реалізація математичних обчислень з використанням циклів та функцій у Python.

Мета: Навчитися створювати функції, використовувати цикли для обчислень та працювати з користувацьким введенням у Python.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Функції в Python

Функція у Python є блоком коду, який виконує певну задачу та може бути викликана багаторазово з різними параметрами. Функції дозволяють структурувати код, зробити його більш читабельним, зменшити повторення та забезпечити модульність програми. Основна перевага функцій полягає в тому, що один раз написаний код може використовуватися неодноразово з різними вхідними даними.

Синтаксис визначення функції включає ключове слово def, за яким слідує ім'я функції, параметри у круглих дужках та двокрапку. Тіло функції має бути відступлене від лівого краю та може містити оператор return для повернення результату виклику функції. Ім'я функції повинно бути описовим та відображати її призначення, а також слідувати правилам іменування змінних у Python.

Дефініція функції визначає її інтерфейс та поведінку. Параметри функції є змінними, які приймають значення, передані при виклику функції. Python підтримує різні типи параметрів: позиційні параметри, які передаються у певному порядку та є обов'язковими; іменовані параметри, які можна передавати у довільному порядку, вказуючи їх імена; параметри з значеннями за замовчуванням, які використовуються, якщо значення не передано при виклику функції.

Приклад створення простої функції для обчислення квадрата числа:

```
python
def square(x):
    """Функція для обчислення квадрата числа"""
    result = x ** 2
    return result

# Виклик функції
number = 5
squared = square(number)
print(f"Квадрат числа {number} дорівнює {squared}")
```

Приклад функції з кількома параметрами та значенням за замовчуванням:

```
def calculate_power(base, exponent=2):
    """Функція для обчислення степеня числа"""
    result = base ** exponent
```

```
    return result

# Різні способи виклику
print(calculate_power(3))              # Використання значення за замовчуванням
print(calculate_power(3, 4))    # Передача обох параметрів
print(calculate_power(base=2, exponent=3))  # Іменовані параметри
```

Оператор return завершує виконання функції та повертає значення до місця виклику. Якщо оператор return не вказано або вказано без значення, функція повертає None. Функція може мати кілька операторів return, але виконається тільки перший, до якого дійде програма.

Цикли в Python

Цикли є фундаментальними конструкціями програмування, які дозволяють виконувати блок коду багаторазово. Python надає два основні типи циклів: for та while, кожен з яких має свої особливості та сфери застосування.

Цикл for використовується для ітерації по послідовностях (рядки, списки, кортежі) або для виконання коду певну кількість разів. Він особливо корисний, коли кількість ітерацій відома наперед або потрібно перебрати всі елементи послідовності. Синтаксис циклу for включає ключове слово for, змінну-лічильник, ключове слово in, об'єкт для ітерації та двокрапку.

Функція range() часто використовується разом з циклом for для генерації послідовності чисел. Вона може приймати один параметр (кінцеве значення, починаючи з 0), два параметри (початкове та кінцеве значення) або три параметри (початкове значення, кінцеве значення та крок). Важливо пам'ятати, що кінцеве значення не включається до послідовності.

Приклади використання циклу for:

```
# Простий цикл від 0 до 4
for i in range(5):
    print(f"Ітерація {i}")

# Цикл з початковим та кінцевим значенням
for i in range(2, 8):
    print(f"Число: {i}")

# Цикл з кроком
for i in range(0, 10, 2):
    print(f"Парне число: {i}")

# Цикл для обчислення суми
sum_result = 0
for i in range(1, 6):
    sum_result += i
print(f"Сума чисел від 1 до 5: {sum_result}")
```

Цикл while виконується доти, доки задана умова залишається істинною. Він використовується, коли кількість ітерацій наперед невідома та залежить від виконання певної умови. Синтаксис включає ключове слово while, логічну умову та двокрапку. Важливо забезпечити, щоб умова в певний момент стала хибною, інакше цикл буде нескінченним.

Приклади використання циклу while:

```
# Простий лічильник
count = 0
while count < 5:
    print(f"Лічильник: {count}")
    count += 1

# Цикл з умовою користувача
number = 1
while number != 0:
    number = int(input("Введіть число (0 для виходу): "))
    if number != 0:
        print(f"Ви ввели: {number}")

# Цикл для знаходження факторіала
n = 5
factorial = 1
temp = n
while temp > 0:
    factorial *= temp
    temp -= 1
print(f"Факторіал {n} дорівнює {factorial}")
```

При роботі з циклами корисно знати про оператори break та continue. Оператор break завершує виконання циклу достроково, а continue пропускає решту коду в поточній ітерації та переходить до наступної ітерації.

При обчисленні сум або добутків у циклі важливо правильно ініціалізувати змінну-акумулятор перед початком циклу. Для сум початкове значення зазвичай дорівнює нулю, для добутків - одиниці. У кожній ітерації циклу до акумулятора додається (для сум) або множиться (для добутків) поточний член послідовності.

Цикли особливо корисні для обчислення математичних рядів, таких як ряди Тейлора, геометричні прогресії, арифметичні прогресії та інші послідовності. При реалізації таких обчислень важливо правильно організувати логіку циклу та забезпечити точність обчислень.

Форматування виводу в операторі print

Оператор print() у Python надає різноманітні можливості для форматування виводу результатів. Найсучасніший та зручний спосіб форматування - це f-рядки (f-strings), які дозволяють вставляти значення змінних безпосередньо у рядок. Синтаксис включає літеру f перед рядком та фігурні дужки для вставки змінних та виразів.

Для контролю кількості десяткових знаків у дійсних числах можна використовувати специфікатори формату після двокрапки у фігурних дужках. Специфікатор .nf виводить дійсне число з n знаками після коми, .ne - у науковому форматі, .n% - у відсотковому форматі.

Приклади форматування виводу:

```
python
# Основне форматування f-рядків
name = "Python"
version = 3.9
print(f"Мова програмування: {name}, версія: {version}")

# Форматування дійсних чисел
pi = 3.14159265359
print(f"Число π з 2 знаками: {pi:.2f}")
print(f"Число π з 4 знаками: {pi:.4f}")
print(f"Число π у науковому форматі: {pi:.2e}")

# Вирівнювання тексту
number = 42
print(f"Число з вирівнюванням по лівому краю: '{number:<10}'")
print(f"Число з вирівнюванням по правому краю: '{number:>10}'")
print(f"Число з центруванням: '{number:^10}'")

# Заповнення нулями
print(f"Число з нулями: {number:05d}")

# Відсотковий формат
fraction = 0.85
print(f"У відсотках: {fraction:.1%}")
```

Альтернативні методи форматування включають метод .format() та оператор %, хоча f-рядки є більш читабельними та ефективними:

```
# Метод .format()
name = "Python"
print("Мова програмування: {}".format(name))
print("Число π: {:.3f}".format(3.14159))

# Оператор %
print("Мова програмування: %s" % name)
print("Число π: %.3f" % 3.14159)
```

Для табличного виводу корисним є використання табуляції (\t) або форматування з фіксованою шириною полів:

```
# Табличний вивід
print("x\t\tf(x)")
print("-" * 15)
for x in [1, 2, 3, 4, 5]:
    result = x ** 2
    print(f"{x}\t\t{result}")

# Вивід з фіксованою шириною
```

```
print(f"{'x':>5} {'f(x)':>10}")
print("-" * 16)
for x in [1, 2, 3, 4, 5]:
    result = x ** 2
    print(f"{x:>5} {result:>10}")
```

Табуляція результатів

Табуляція функцій передбачає обчислення значень функції для різних значень аргументу з певним кроком. Це дозволяє проаналізувати поведінку функції на заданому інтервалі та побудувати таблицю значень для подальшого аналізу або візуалізації.

При табуляції важливо правильно вибрати початкове значення, кінцеве значення та крок зміни аргументу. Крок повинен бути достатньо малим для отримання детальної картини поведінки функції, але не настільки малим, щоб таблиця стала занадто великою та незручною для аналізу.

Результати табуляції зазвичай представляють у вигляді таблиці з колонками для значення аргументу та відповідного значення функції. Для покращення читабельності таблиці рекомендується використовувати заголовки колонок, розділювальні лінії та однакове форматування всіх числових значень.

Приклад коду

Нижче наведено приклад реалізації завдання для формули: Σ(sin(x) / n), де *n* змінюється від 1 до N.

```
import math

def calculate_sum(x, n):
    result = 0
    for i in range(1, n + 1):
        result += math.sin(x) / i
    return result

def main():
    x = float(input("Введіть значення x: "))
    n = int(input("Введіть значення n: "))

    print("\nТабуляція результатів:")
    print("x\t\tРезультат")
    print("----------------------")

    for i in range(5):  # Табулюємо для 5 значень x
        current_x = x + i * 0.5
        result = calculate_sum(current_x, n)
        print(f"{current_x:.2f}\t\t{result:.4f}")

if __name__ == "__main__":
    main()
```

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Створіть скрипт Python, який складається з основної частини програми та функції, яка реалізує задану математичну формулу (згідно з варіантом по журналу) з використанням операторів циклу для обчислення суми.

2. Функція повинна приймати два параметри:

x - значення змінної;

n - кількість ітерацій.

3. У основній частині програми:

- запитайте у користувача значення змінних *x* та *n*;
- викличте створену функцію для табулювання результатів у заданому діапазоні значень *x*;
- виведіть результати на екран.

Таблиця 4.1 – Варіанти індивідуальних завдань

Варіант	Завдання	Варіант	Завдання
1	$\sum_{n=1}^{N} \frac{\sin(x^n)}{n\sqrt{n}}$	16	$\sum_{n=1}^{N} \frac{\cos(nx)}{n^2+\log(n)}$
2	$\sum_{n=1}^{N} \frac{e^{nx}}{\tan(n)}$	17	$\sum_{n=1}^{N} \frac{\log(nx)}{n!}$
3	$\sum_{n=1}^{N} \frac{\sqrt{x^n+n}}{n\cdot\arctan(n)}$	18	$\sum_{n=1}^{N} \frac{\sin(nx)\cdot\cos(nx)}{n^2+e^n}$
4	$\sum_{n=1}^{N} \frac{\log(n+1)}{x^n+\sqrt{n}}$	19	$\sum_{n=1}^{N} \frac{n^x}{\sinh(n)+\cosh(n)}$
5	$\sum_{n=1}^{N} \frac{\arcsin(\frac{1}{n+1})}{x^n\cdot\log(n+2)}$	20	$\sum_{n=1}^{N} \frac{e^{\sin(nx)}}{\sqrt{n^3+\cos(x)}}$
6	$\sum_{n=1}^{N} \frac{\tan(nx)}{n^2+x^2}$	21	$\sum_{n=1}^{N} \frac{\arccos(\frac{1}{n+2})}{n\cdot e^x}$
7	$\sum_{n=1}^{N} \frac{\log(n+x)}{\sqrt{n^3+1}}$	22	$\sum_{n=1}^{N} \frac{x^n}{\sin(n)+\cos(n)}$
8	$\sum_{n=1}^{N} \frac{\sinh(nx)}{n^3+\log(x)}$	23	$\sum_{n=1}^{N} \frac{n!}{x^n+e^n}$
9	$\sum_{n=1}^{N} \frac{\log(n+x)}{n\cdot\sinh(x)}$	24	$\sum_{n=1}^{N} \frac{\cos(n^2x)}{\sqrt{n}+\log(n+1)}$
10	$\sum_{n=1}^{N} \frac{x^{n/2}}{\cosh(n)\cdot n^2}$	25	$\sum_{n=1}^{N} \frac{\arctan(nx)}{n\cdot\sin(x)}$

11	$\sum_{n=1}^{N} \frac{e^{-nx}}{\sqrt{n^4+1}}$	26	$\sum_{n=1}^{N} \frac{\sin(n^2)\cdot\cos(x^n)}{n^3+1}$
12	$\sum_{n=1}^{N} \frac{\arcsin(1/(n+1))}{n\cdot e^{x/n}}$	27	$\sum_{n=1}^{N} \frac{x^n}{\sqrt{n!}+\sinh(n)}$
13	$\sum_{n=1}^{N} \frac{\tan(nx)\cdot\cosh(x)}{n^2+\log(n+x)}$	28	$\sum_{n=1}^{N} \frac{\log(nx+1)}{n\cdot\arccos(1/(n+1))}$
14	$\sum_{n=1}^{N} \frac{e^{\cos(nx)}}{n^2\cdot\sqrt{\sin(x)+2}}$	29	$\sum_{n=1}^{N} \frac{\sinh(nx)\cdot\cosh(x)}{n^3+\tan(n)}$
15	$\sum_{n=1}^{N} \frac{\arctan(x/n)}{n\cdot\log(n+x+1)}$	30	$\sum_{n=1}^{N} \frac{n^x}{\sqrt{n^5+e^x}}$

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент реалізує програму, призначену для обчислення значення математичного виразу, що містить суму або іншу конструкцію з використанням циклів та функцій. Програма передбачає отримання вхідних даних від користувача, виконання обчислень за допомогою визначеної користувацької функції та виведення результатів у зручному форматі. Особливу увагу приділяється структуруванню коду, використанню циклів для ітеративних обчислень та організації введення-виведення.

У протоколі обов'язково мають бути наведені три ключові елементи: по-перше, завдання, в якому чітко записано індивідуальний математичний вираз згідно з варіантом студента; по-друге, лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють призначення функцій, циклів та основних блоків логіки; по-третє, результати виконання програми, представлені у вигляді скріншотів або текстового виводу консолі, що підтверджують коректну роботу коду для

заданих вхідних даних. Ці компоненти разом забезпечують повну відтворюваність та перевірку виконаного завдання.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 5

Тема: Робота з файлами в Python

Мета роботи: Отримати практичні навички роботи з файлами в Python, включаючи читання, запис та обробку текстових даних.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Робота з файлами є однією з фундаментальних можливостей мови програмування Python, що дозволяє програмам взаємодіяти з файловою системою операційної системи для читання, запису та модифікації даних, які зберігаються у постійній пам'яті. Python надає розробникам потужний та інтуїтивно зрозумілий інтерфейс для роботи з файлами різних типів, включаючи текстові та бінарні файли, завдяки вбудованим функціям та методам, які забезпечують ефективне управління файловими операціями. Основною перевагою Python у контексті файлових операцій є його здатність автоматично управляти ресурсами системи через механізм контекстних менеджерів, що гарантує правильне відкриття та закриття файлів навіть у випадку виникнення помилок під час виконання програми. Крім того, Python підтримує різноманітні кодування символів, що робить його ідеальним інструментом для обробки текстових файлів у будь-якій мові світу, включаючи українську, та забезпечує коректне відображення національних символів. Використання файлових операцій у Python дозволяє створювати програми, які можуть обробляти великі обсяги даних, зберігати результати обчислень для подальшого використання, налаштовувати параметри роботи програм через конфігураційні файли та забезпечувати персистентність даних між різними запусками програми.

Основні функції та методи для роботи з файлами в Python представлені в таблиці 5.1, яка систематизує найважливіші інструменти, необхідні для ефективної взаємодії з файловою системою.

Таблиця 5.1 - Основні функції та методи для роботи з файлами в Python

Функція/Метод	Призначення	Синтаксис
open()	Відкриття файлу для роботи	open(filename, mode, encoding)
close()	Закриття файлу	file_object.close()
read()	Читання всього вмісту файлу	file_object.read()
readline()	Читання одного рядка з файлу	file_object.readline()
readlines()	Читання всіх рядків у список	file_object.readlines()
write()	Запис рядка у файл	file_object.write(string)
writelines()	Запис списку рядків у файл	file_object.writelines(list)
seek()	Переміщення покажчика у файлі	file_object.seek(position)
tell()	Отримання поточної позиції покажчика	file_object.tell()
with	Контекстний менеджер для файлів	with open(filename) as file:

Функція open() та режими відкриття файлів

Функція open() є основним інструментом для відкриття файлів у Python і приймає декілька параметрів, серед яких найважливішими є ім'я файлу та режим відкриття. Перший параметр функції open() визначає шлях до файлу, який може бути як абсолютним, так і відносним відносно поточної робочої директорії програми. Другий параметр, режим відкриття, визначає, яким чином програма буде взаємодіяти з файлом та які операції будуть дозволені під час роботи з ним.

Режим 'r' (read) використовується для читання існуючих файлів і є режимом за замовчуванням, якщо режим не вказано явно. Цей режим дозволяє лише читати дані з файлу, але не дозволяє модифікувати його вміст. Якщо файл не існує, Python викине виняток FileNotFoundError.

```
# Відкриття файлу для читання
file = open('input.txt', 'r', encoding='utf-8')
content = file.read()
print(content)
file.close()
```

Режим 'w' (write) створює новий файл для запису або перезаписує існуючий файл, повністю видаляючи його попередній вміст. Використання цього режиму потребує особливої обережності, оскільки всі дані у файлі будуть безповоротно втрачені при його відкритті.

```
# Створення нового файлу для запису
file = open('output.txt', 'w', encoding='utf-8')
file.write('Привіт, світ!')
file.close()
```

Режим 'a' (append) дозволяє додавати нові дані до кінця існуючого файлу, зберігаючи при цьому весь попередній вміст. Якщо файл не існує, він буде створений автоматично. Цей режим особливо корисний для ведення журналів подій або накопичення результатів обчислень.

```
# Додавання даних до існуючого файлу
file = open('log.txt', 'a', encoding='utf-8')
file.write('\nНовий запис у журналі')
file.close()
```

Контекстний менеджер with

Конструкція with представляє найкращий спосіб роботи з файлами у Python, оскільки вона автоматично керує відкриттям та закриттям файлів, навіть якщо під час виконання коду виникають помилки або винятки. Контекстний менеджер гарантує, що файл буде коректно закритий після завершення блоку коду, що запобігає втраті даних та звільняє системні ресурси. Використання with також робить код більш читабельним та зменшує

ймовірність помилок, пов'язаних з некоректним управлінням файловими ресурсами.

```
# Використання контекстного менеджера with
with open('data.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
# Файл автоматично закривається після виходу з блоку with
```

Контекстний менеджер особливо корисний при роботі з великими файлами або при виконанні складних операцій, де висока ймовірність виникнення винятків. Навіть якщо у блоці with виникне помилка, файл все одно буде коректно закритий, що запобігає блокуванню файлових ресурсів операційної системи.

```
# Безпечна робота з файлами при можливих помилках
try:
    with  open('suspicious_file.txt',  'r',  encoding='utf-8')  as
file:
        data = file.read()
        result = int(data)  # Може викликати ValueError
        print(f"Результат: {result}")
except FileNotFoundError:
    print("Файл не знайдено")
except ValueError:
    print("Неможливо перетворити дані на число")
```

Методи читання даних з файлів

Метод read() зчитує весь вміст файлу як один рядок, що може бути зручно для невеликих файлів, але може спричинити проблеми з пам'яттю при роботі з великими файлами. Цей метод повертає рядок, який містить весь текст файлу, включаючи символи переносу рядків. Можна також передати параметр розміру, щоб прочитати лише певну кількість символів з файлу.

```
# Читання всього вмісту файлу
with open('book.txt', 'r', encoding='utf-8') as file:
    full_content = file.read()
    print(f"Довжина тексту: {len(full_content)} символів")

# Читання перших 100 символів
with open('book.txt', 'r', encoding='utf-8') as file:
    partial_content = file.read(100)
    print(f"Перші 100 символів: {partial_content}")
```

Метод readline() читає один рядок з файлу за раз, включаючи символ переносу рядка в кінці. Повторні виклики цього методу будуть читати наступні рядки файлу послідовно. Цей метод корисний, коли потрібно обробляти файл по рядках без завантаження всього вмісту в пам'ять одночасно.

```
# Читання файлу по рядках
with open('students.txt', 'r', encoding='utf-8') as file:
```

```
    line_number = 1
    while True:
        line = file.readline()
        if not line:  # Кінець файлу
            break
        print(f"Рядок {line_number}: {line.strip()}")
        line_number += 1
```

Метод readlines() зчитує всі рядки файлу та повертає їх у вигляді списку рядків, де кожен елемент списку відповідає одному рядку файлу. Кожен рядок у списку зберігає символ переносу рядка в кінці, тому часто потрібно використовувати метод strip() для його видалення.

```
# Читання всіх рядків у список
with open('words.txt', 'r', encoding='utf-8') as file:
    all_lines = file.readlines()
    clean_lines = [line.strip() for line in all_lines]
    print(f"Знайдено {len(clean_lines)} слів")
    for i, word in enumerate(clean_lines, 1):
        print(f"{i}. {word}")
```

Методи запису даних у файли

Метод write() записує рядок у файл і повертає кількість записаних символів. Важливо пам'ятати, що цей метод не додає автоматично символ переносу рядка, тому для створення нових рядків потрібно явно включати символ '\n' у рядок, який записується. Метод write() працює лише з рядковими даними, тому інші типи даних спочатку потрібно перетворити на рядки.

```
# Запис тексту у файл
with open('results.txt', 'w', encoding='utf-8') as file:
    file.write('Результати експерименту:\n')
    file.write('Температура: 25°C\n')
    file.write('Вологість: 60%\n')

    # Запис числових даних
    temperature = 25.5
    humidity = 62.3
    file.write(f'Точні значення: {temperature}°C, {humidity}%\n')
```

Метод writelines() приймає послідовність рядків (список або кортеж) і записує їх у файл один за одним. Цей метод не додає автоматично символи переносу рядків між елементами, тому їх потрібно включати у самі рядки або додавати окремо.

```
# Запис списку рядків у файл
data_lines = [
    'Студент 1: Іванов Іван\n',
    'Студент 2: Петренко Петро\n',
    'Студент 3: Сидорчук Марія\n'
]
```

```
with open('students_list.txt', 'w', encoding='utf-8') as file:
    file.writelines(data_lines)

# Альтернативний спосіб з додаванням переносів рядків
names = ['Іванов Іван', 'Петренко Петро', 'Сидорчук Марія']
with  open('students_formatted.txt',  'w',  encoding='utf-8')  as
file:
    file.writelines([name + '\n' for name in names])
```

Обробка винятків при роботі з файлами

Робота з файлами часто може призводити до різноманітних помилок, тому важливо передбачити обробку винятків для забезпечення стабільної роботи програми. Найпоширенішими винятками при роботі з файлами є FileNotFoundError, який виникає при спробі відкрити неіснуючий файл для читання, PermissionError, який з'являється при недостатніх правах доступу до файлу або директорії, та UnicodeDecodeError, який виникає при неправильному визначенні кодування файлу.

```
# Комплексна обробка винятків при роботі з файлами
def safe_file_processing(filename):
    try:
        with open(filename, 'r', encoding='utf-8') as file:
            content = file.read()
            word_count = len(content.split())
            return f"Файл містить {word_count} слів"

    except FileNotFoundError:
        return f"Помилка: файл '{filename}' не знайдено"

    except PermissionError:
        return  f"Помилка:  недостатньо  прав  для  читання  файлу
'{filename}'"

    except UnicodeDecodeError:
        return f"Помилка: неможливо декодувати файл '{filename}' з
кодуванням UTF-8"

    except Exception as e:
        return f"Несподівана помилка: {str(e)}"

# Використання функції
result = safe_file_processing('test.txt')
print(result)
```

Для обробки помилок кодування можна використовувати параметр errors функції open(), який дозволяє вказати стратегію обробки некоректних символів. Значення 'ignore' ігнорує некоректні символи, 'replace' замінює їх символом заміни, а 'strict' (за замовчуванням) викликає виняток.

```
# Робота з файлами різних кодувань
def read_file_with_fallback(filename):
```

```
    encodings = ['utf-8', 'cp1251', 'latin-1']

    for encoding in encodings:
        try:
            with open(filename, 'r', encoding=encoding) as file:
                content = file.read()
                print(f"Файл успішно прочитано з кодуванням
{encoding}")
                return content
        except UnicodeDecodeError:
            print(f"Не вдалося прочитати з кодуванням {encoding}")
            continue

    # Останній варіант з ігноруванням помилок
    with open(filename, 'r', encoding='utf-8', errors='ignore') as
file:
        content = file.read()
        print("Файл прочитано з ігноруванням некоректних
символів")
        return content
```

Навігація у файлах за допомогою seek() та tell()

Методи seek() та tell() дозволяють контролювати позицію покажчика читання/запису у файлі, що особливо корисно при роботі з великими файлами або при необхідності читання певних частин файлу у довільному порядку. Метод tell() повертає поточну позицію покажчика у байтах від початку файлу, а метод seek() переміщує покажчик у вказану позицію.

```
# Демонстрація роботи з позиціонуванням у файлі
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write('Перший рядок\nДругий рядок\nТретій рядок\n')

with open('example.txt', 'r', encoding='utf-8') as file:
    # Читаємо початок файлу
    first_part = file.read(10)
    print(f"Перші 10 символів: '{first_part}'")

    # Визначаємо поточну позицію
    current_position = file.tell()
    print(f"Поточна позиція: {current_position}")

    # Переміщуємося на початок файлу
    file.seek(0)
    full_content = file.readline()
    print(f"Перший рядок: '{full_content.strip()}'")

    # Переміщуємося у кінець файлу
    file.seek(0, 2)  # 2 означає кінець файлу
    end_position = file.tell()
    print(f"Розмір файлу: {end_position} байтів")
```

Рисунок 5.1 демонструє концептуальну схему роботи покажчика позиції у файлі при використанні методів seek() та tell(). На схемі показано файл як послідовність символів з нумерацією позицій, покажчик поточної позиції, який може переміщуватися за допомогою seek(), та операції читання, які починаються з поточної позиції покажчика. Метод tell() повертає числове значення поточної позиції, яка вимірюється у байтах від початку файлу, причому позиція 0 відповідає самому першому символу файлу.

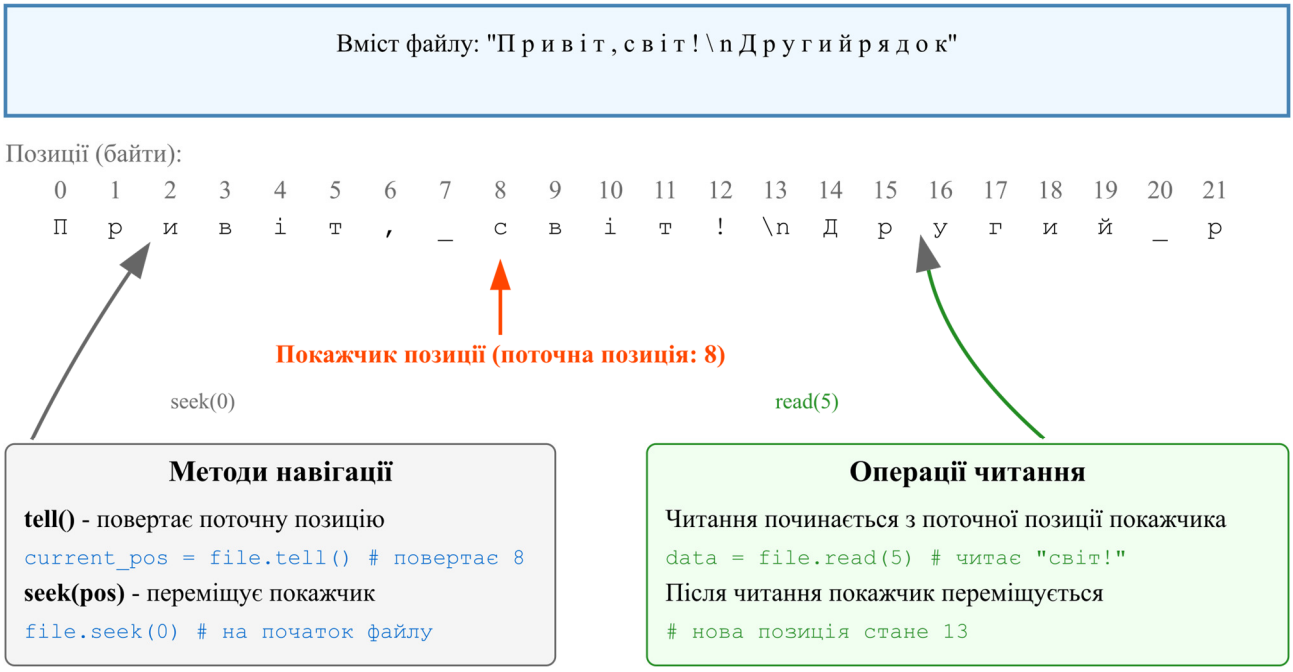


Рисунок 5.1 - Схема роботи покажчика позиції у файлі

Практичний приклад обробки текстового файлу

Розглянемо практичний приклад програми, яка демонструє основні принципи роботи з файлами для вирішення типової задачі аналізу текстової інформації. Програма читає текстовий файл, підраховує кількість абзаців та визначає середню довжину слова у тексті, після чого записує результати у вихідний файл.

```
def process_text_file(input_filename, output_filename):
    """
    Програма для обробки текстового файлу:
    підраховує кількість абзаців та знаходить середню довжину слова
    """

    try:
        # Читання вхідного файлу
        with open(input_filename, 'r', encoding='utf-8') as file:
            content = file.read()

        # Підрахунок абзаців
        paragraphs = content.split('\n\n')
        paragraphs_count = len([p for p in paragraphs if p.strip()])

        # Підрахунок середньої довжини слова
```

```
        import string
        clean_text        =        content.translate(str.maketrans('',        '',
string.punctuation))
        words = clean_text.split()
        words = [word for word in words if word]  # Видалення порожніх слів

        if words:
            total_length = sum(len(word) for word in words)
            average_word_length = total_length / len(words)
        else:
            average_word_length = 0

        # Запис результатів у вихідний файл
        with open(output_filename, 'w', encoding='utf-8') as file:
            file.write("РЕЗУЛЬТАТИ АНАЛІЗУ\n")
            file.write("=" * 20 + "\n")
            file.write(f"Кількість абзаців: {paragraphs_count}\n")
            file.write(f"Середня    довжина    слова:    {average_word_length:.2f}
символів\n")

        print(f"Обробка     завершена.     Результати     збережено     у     файл
'{output_filename}'")

        # Виведення результатів на екран
        print(f"Кількість абзаців: {paragraphs_count}")
        print(f"Середня довжина слова: {average_word_length:.2f} символів")

    except FileNotFoundError:
        print(f"Помилка: Файл '{input_filename}' не знайдено.")
    except Exception as e:
        print(f"Виникла помилка: {e}")

# Головна частина програми
if __name__ == "__main__":
    input_file = "input.txt"
    output_file = "output.txt"

    # Виконання обробки файлу
    process_text_file(input_file, output_file)
```

Цей приклад демонструє комплексне використання основних методів роботи з файлами у Python, включаючи читання текстових даних за допомогою методу read(), обробку та аналіз вмісту файлу з використанням рядкових операцій, створення вихідного файлу з результатами аналізу та правильну обробку винятків. Програма показує практичне застосування контекстного менеджера with для безпечної роботи з файлами, використання вбудованого модуля string для очищення тексту від розділових знаків та форматування результатів у зручному для читання вигляді.

2 КЛЮЧОВІ ПИТАННЯ

1. Які основні режими відкриття файлів існують у Python та яке їх призначення?
2. Як правильно відкрити та закрити файл у Python для уникнення втрати даних?

3. Які методи використовуються для читання даних з файлу та чим вони відрізняються?

4. Як здійснюється запис даних у файл та які особливості мають різні режими запису?

5. Яке призначення конструкції with при роботі з файлами та які переваги вона надає?

6. Як обробляти винятки при роботі з файлами та які найпоширеніші типи помилок можуть виникнути?

7. Як визначити кодування файлу та чому це важливо при роботі з текстовими даними?

8. Які методи дозволяють читати файл по рядках та як їх використовувати ефективно?

9. Як додати нові дані до існуючого файлу без втрати попередньої інформації?

10. Які особливості має робота з великими файлами та як оптимізувати використання пам'яті?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Розробити програму мовою Python для обробки текстового файлу відповідно до індивідуального варіанту завдання. Програма повинна здійснювати читання вхідних даних з текстового файлу, виконувати необхідні обчислювальні операції над текстовою інформацією та записувати результати обробки у вихідний текстовий файл. При реалізації програми обов'язково використовувати конструкцію контекстного менеджера `with` для забезпечення коректного відкриття та автоматичного закриття файлів, що гарантує належне управління ресурсами системи та запобігає можливим втратам даних.

Вхідний файл повинен містити текстову інформацію обсягом не менше 10-15 рядків для забезпечення належної демонстрації можливостей розробленої програми. Студент має самостійно створити вхідний файл з відповідним вмістом або використати тестові дані, надані викладачем. Вихідний файл повинен містити результати обробки вхідних даних у зручному для читання форматі, який відповідає специфікації обраного варіанту завдання. Особлива увага повинна приділятися правильному форматуванню вихідних даних та їх повноті відповідно до поставленої задачі.

Програма повинна бути реалізована з урахуванням можливих виняткових ситуацій, зокрема випадків відсутності вхідного файлу, проблем з доступом до файлової системи або некоректного формату вхідних даних. Для цього необхідно передбачити відповідну обробку винятків за допомогою конструкцій try-except, які забезпечать стабільну роботу програми у нестандартних ситуаціях. Кожен студент отримує індивідуальний номер варіанту згідно з таблицею варіантів, яка містить 30 різноманітних завдань з обробки текстової інформації, що охоплює широкий спектр операцій з аналізу, фільтрації та трансформації текстових даних.

Таблиця 5.2 – Варіанти індивідуальних завдань

№	Завдання
1	Порахувати кількість слів у файлі.
2	Порахувати кількість рядків у файлі.
3	Знайти найдовше слово у файлі.
4	Знайти найкоротше слово у файлі.
5	Порахувати кількість голосних літер у файлі.
6	Порахувати кількість приголосних літер у файлі.
7	Вивести всі слова у верхньому регістрі.
8	Вивести всі числа, що зустрічаються у файлі.
9	Замінити всі голосні літери на символ *.
10	Вивести всі унікальні слова файлу.
11	Визначити, чи є текст файлу паліндромом (без урахування пробілів).
12	Вивести слова, що починаються з великої літери.
13	Вивести слова, довжина яких більша за 5 символів.
14	Вивести слова, що містять цифри.
15	Порахувати частоту появи кожного слова у файлі.
16	Створити новий файл, де кожне слово написане навпаки.
17	Вивести всі речення, що містять слово "Python".
18	Замінити всі цифри у файлі на символ #.
19	Вивести всі речення, довжина яких більше 10 слів.
20	Порахувати кількість речень у файлі.
21	Вивести слова, що складаються лише з великих літер.
22	Видалити всі слова, що повторюються.
23	Вивести всі речення у зворотному порядку.
24	Визначити, чи всі речення файлу починаються з великої літери.
25	Знайти найдовше речення у файлі.
26	Перетворити всі слова у нижній регістр.
27	Видалити всі пробіли з файлу.
28	Замінити всі розділові знаки на символ `
29	Порахувати кількість символів у файлі (без пробілів).
30	Вивести всі слова, що мають парну довжину.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент реалізує програму, призначену для обробки текстового файлу згідно з індивідуальним завданням. Програма передбачає читання вхідних даних з файлу, виконання заданих операцій над текстовою інформацією (наприклад, аналіз, фільтрація, перетворення) та запис результатів у новий вихідний файл. Особливу увагу приділяється коректному відкриттю та закриттю файлів за допомогою контекстного менеджера with, обробці можливих винятків, а також читабельності та структурованості коду.

У протоколі обов'язково мають бути наведені три ключові елементи:

– завдання, у якому чітко записано індивідуальний варіант обробки текстових даних;

– лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють логіку читання, обробки та запису файлів;

– результати виконання програми, представлені у вигляді фрагментів вхідного та вихідного файлів (текстовий вивід або скріншоти), що підтверджують коректну роботу програми.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов'язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 6

Тема: Об'єктно-орієнтоване програмування в Python

Мета роботи: Отримати практичні навички створення класів, об'єктів та використання основних принципів об'єктно-орієнтованого програмування в мові Python.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Об'єктно-орієнтоване програмування (ООП) є парадигмою програмування, яка базується на концепції об'єктів, що містять дані (атрибути) та код (методи). У Python клас являє собою шаблон або креслення для створення об'єктів, визначаючи їхню структуру та поведінку. Об'єкт є екземпляром класу - конкретною реалізацією, що створюється на основі цього класу. Наприклад, якщо клас "Автомобіль" описує загальні характеристики та функції автомобіля, то конкретний об'єкт може представляти окремий автомобіль з унікальними значеннями марки, моделі та року випуску. Зв'язок між класом та об'єктом подібний до зв'язку між кресленням будинку та самим побудованим будинком.

Створення класу в Python здійснюється за допомогою ключового слова class, після якого вказується назва класу з великої літери. Метод __init__ є спеціальним методом-конструктором, який автоматично викликається при створенні нового об'єкта класу та використовується для ініціалізації атрибутів об'єкта початковими значеннями. Перший параметр методу __init__ завжди має назву self і представляє посилання на поточний екземпляр класу. Через цей параметр здійснюється доступ до атрибутів та методів конкретного об'єкта, що дозволяє кожному екземпляру класу мати власний набір значень атрибутів.

```
class Student:
    def __init__(self, name, age, course):
        self.name = name
        self.age = age
        self.course = course
        self.grades = []
```

Атрибути класу можуть бути двох типів: атрибути екземпляра та атрибути класу. Атрибути екземпляра визначаються в методі __init__ та є унікальними для кожного об'єкта, тоді як атрибути класу є спільними для всіх екземплярів класу. Методи класу є функціями, визначеними всередині класу, які описують дії, що можуть виконувати об'єкти цього класу. Доступ до атрибутів та методів здійснюється через крапкову нотацію: object.attribute або object.method(). Методи завжди отримують self як перший параметр, що дозволяє їм працювати з атрибутами конкретного екземпляра.

У Python існують різні рівні доступу до атрибутів та методів, які реалізуються через умовні позначення. Публічні атрибути та методи доступні з

будь-якого місця програми та не мають спеціальних префіксів. Захищені (protected) елементи позначаються одним підкресленням на початку назви (_attribute) і за домовленістю не повинні використовуватися поза класом та його нащадками. Приватні (private) атрибути та методи починаються з двох підкреслень (__attribute) і автоматично перейменовуються інтерпретатором Python, що ускладнює доступ до них ззовні класу.

Інкапсуляція є одним з основних принципів ООП, який передбачає приховування внутрішньої реалізації об'єкта та надання контрольованого доступу до його стану через публічні методи. В Python інкапсуляція реалізується через використання приватних атрибутів та методів-аксесорів (getter) і мутаторів (setter). Властивості (properties) дозволяють створювати методи, які можна використовувати як атрибути, забезпечуючи при цьому контроль над процесом читання та запису значень. Декоратор @property перетворює метод на властивість для читання, а @attribute.setter створює відповідний метод для запису.

```
class Student:
    def __init__(self, name, age, course):
        self.name = name
        self.__age = age  # приватний атрибут
        self.course = course
        self.__grades = []  # приватний список оцінок

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, value):
        if 16 <= value <= 100:
            self.__age = value
        else:
            raise ValueError("Вік студента має бути від 16 до 100
років")

    def add_grade(self, grade):
        if 0 <= grade <= 100:
            self.__grades.append(grade)
        else:
            raise ValueError("Оцінка має бути від 0 до 100")

    def get_average_grade(self):
        if self.__grades:
            return sum(self.__grades) / len(self.__grades)
        return 0

    def get_grades_count(self):
        return len(self.__grades)

    def __str__(self):
```

```
        return f"Студент {self.name}, {self.__age} років,
{self.course} курс"
```

Створення об'єкта класу Student та робота з його методами демонструє практичне застосування принципів ООП. Для створення нового студента використовується конструктор класу, передаючи необхідні параметри: ім'я, вік та курс. Після створення об'єкта можна викликати його методи для додавання оцінок, обчислення середнього балу та отримання інформації про студента. Важливо зауважити, що спроба безпосереднього доступу до приватних атрибутів (__age або __grades) поза класом призведе до помилки або отримання модифікованої назви атрибута через механізм name mangling.

```
# Створення об'єкта студента
student1 = Student("Іван Петренко", 20, 3)

# Додавання оцінок
student1.add_grade(85)
student1.add_grade(92)
student1.add_grade(78)

# Отримання інформації про студента
print(student1)  # Виведе: Студент Іван Петренко, 20 років, 3 курс
print(f"Середній бал: {student1.get_average_grade():.2f}")     #
85.00
print(f"Кількість оцінок: {student1.get_grades_count()}")  # 3

# Зміна віку через властивість
student1.age = 21
print(f"Новий вік: {student1.age}")  # 21

# Спроба встановити некоректний вік призведе до винятку
try:
    student1.age = 15
except ValueError as e:
    print(f"Помилка: {e}")
```

Наслідування дозволяє створювати нові класи на основі існуючих, успадковуючи їхні атрибути та методи. Батьківський (базовий) клас визначає загальну функціональність, яку можуть використовувати дочірні (похідні) класи. У Python наслідування реалізується шляхом вказання батьківського класу в дужках після назви дочірнього класу. Функція super() використовується для звернення до методів батьківського класу, що особливо корисно при перевизначенні методів. Дочірні класи можуть додавати нові атрибути та методи або змінювати поведінку успадкованих методів, зберігаючи при цьому всю функціональність батьківського класу.

Перевизначення методів у дочірньому класі дозволяє змінити поведінку, успадковану від батьківського класу, зберігаючи ту ж назву методу. Це досягається шляхом оголошення методу з тією ж назвою в дочірньому класі.

При цьому можна повністю замінити функціональність батьківського методу або розширити її, викликавши спочатку батьківський метод через super(), а потім додавши додаткову логіку. Такий підхід забезпечує гнучкість у проектуванні класів та дозволяє адаптувати загальну поведінку під специфічні потреби дочірніх класів.

Поліморфізм є принципом ООП, який дозволяє об'єктам різних класів мати методи з однаковими назвами, але різною реалізацією. У Python поліморфізм реалізується природним чином завдяки динамічній типізації - один і той же код може працювати з об'єктами різних типів, якщо вони мають необхідні методи. Це дозволяє писати більш гнучкий та універсальний код, який може обробляти об'єкти різних класів уніфікованим способом. Наприклад, список може містити об'єкти різних класів, і для кожного з них можна викликати метод з однаковою назвою, отримуючи при цьому поведінку, специфічну для кожного типу об'єкта.

Python підтримує множинне наслідування, що дозволяє класу успадковувати атрибути та методи від декількох батьківських класів одночасно. Це реалізується шляхом перерахування всіх батьківських класів через кому в дужках після назви дочірнього класу. Однак множинне наслідування може призводити до складних ситуацій, таких як проблема діаманта, коли декілька батьківських класів мають спільного предка. Python вирішує такі конфлікти за допомогою алгоритму MRO (Method Resolution Order), який визначає порядок пошуку методів у ієрархії класів.

Спеціальні методи, також відомі як "магічні методи" або "dunder methods", є методами з подвійними підкресленнями на початку та в кінці назви, які дозволяють класам взаємодіяти з вбудованими функціями та операторами Python. Найпоширенішими є __init__ для ініціалізації, __str__ для рядкового представлення об'єкта, __len__ для визначення довжини, __add__ для операції додавання, __eq__ для порівняння на рівність. Ці методи дозволяють об'єктам користувацьких класів поводитися як вбудовані типи даних, забезпечуючи природний та інтуїтивний інтерфейс для роботи з ними. Наприклад, визначивши метод __add__, можна використовувати оператор + для додавання об'єктів класу.

Таблиця 6.1 - Основні спеціальні методи Python для класів

Метод	**Призначення**	**Приклад використання**
__init__(self, ...)	Конструктор класу	Ініціалізація об'єкта
__str__(self)	Рядкове представлення для користувачів	print(object)
__repr__(self)	Формальне рядкове представлення	Для розробників
__len__(self)	Довжина об'єкта	len(object)
__add__(self, other)	Додавання об'єктів	object1 + object2
__eq__(self, other)	Порівняння на рівність	object1 == object2
__lt__(self, other)	Порівняння "менше ніж"	object1 < object2
__getitem__(self, key)	Доступ за індексом	object[key]

Правильне проектування класів передбачає дотримання принципів інкапсуляції, відокремлення обов'язків та створення зрозумілого інтерфейсу.

Кожен клас повинен мати чітко визначену відповідальність та не виконувати занадто багато різнорідних функцій. Назви класів зазвичай є іменниками в однині з великої літери, назви методів - дієсловами в нижньому регістрі з підкресленнями для розділення слів. Важливо забезпечити, щоб клас мав логічно пов'язані атрибути та методи, а також продумати, які елементи повинні бути публічними, а які - приватними.

Рисунок 6.1 демонструє класичну ієрархію наслідування з використанням композиції для створення складної системи. У верхній частині діаграми розташований базовий клас "Person", який містить загальні атрибути name та age, що є спільними для всіх осіб у системі. Цей клас служить фундаментом для створення більш спеціалізованих типів об'єктів та включає конструктор init() для ініціалізації базових властивостей. Використання абстрактного базового класу дозволяє уникнути дублювання коду та забезпечує консистентність у структурі даних для всіх типів осіб. Приватні атрибути (позначені знаком "-") вказують на інкапсуляцію даних, що обмежує прямий доступ до них з зовнішнього коду.

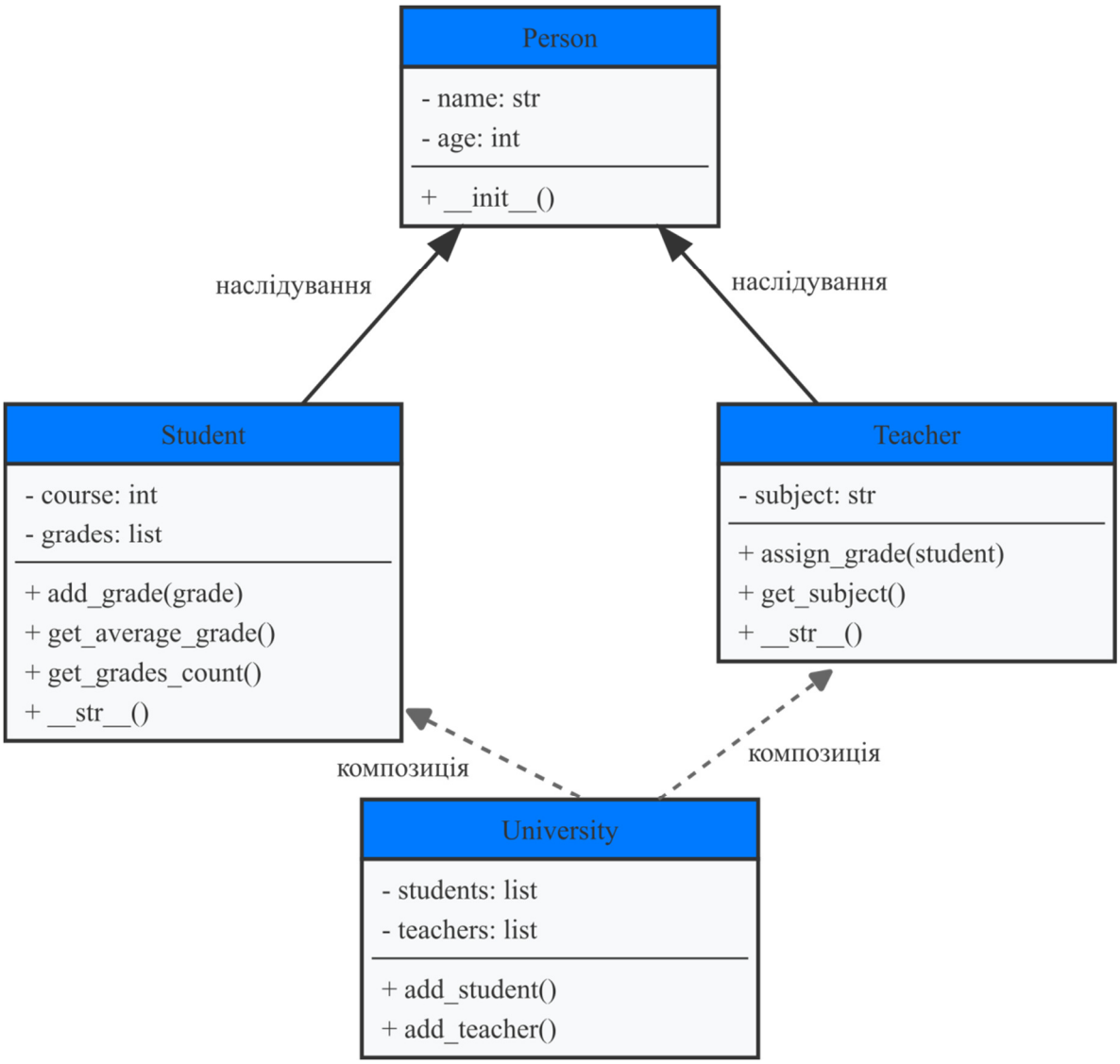


Рисунок 6.1 - Діаграма взаємозв'язків класів у системі управління студентами

Дочірні класи "Student" та "Teacher" успадковують всі властивості батьківського класу "Person" та розширюють його функціональність специфічними для їх ролі атрибутами та методами. Клас "Student" додає атрибут course для зберігання номера курсу та приватний список grades для оцінок студента. Методи цього класу включають add_grade() для додавання нових оцінок з валідацією, get_average_grade() для обчислення середнього балу, get_grades_count() для отримання кількості оцінок та перевизначений метод **str**() для зручного представлення інформації про студента. Клас "Teacher" містить атрибут subject для предмета, який викладає вчитель, та методи assign_grade() для виставлення оцінок студентам, get_subject() для отримання назви предмета та власну реалізацію методу **str**(). Стрілки наслідування на діаграмі показують, що обидва класи є спеціалізаціями базового класу "Person".

Клас "University" в нижній частині діаграми демонструє принцип композиції, агрегуючи колекції об'єктів Student та Teacher через відповідні списки students та teachers. Цей клас не наслідує від "Person", оскільки університет не є особою, а представляє організацію, що управляє людськими ресурсами. Методи add_student() та add_teacher() забезпечують функціональність для додавання нових учасників освітнього процесу до відповідних колекцій. Пунктирні стрілки композиції вказують на те, що University має (has-a) відношення зі Student та Teacher, на відміну від відношення "є" (is-a), характерного для наслідування. Така архітектура дозволяє університету управляти множинами студентів та викладачів, забезпечуючи при цьому слабку зв'язаність між компонентами системи.

Представлена діаграма ілюструює ключові переваги об'єктно-орієнтованого підходу: можливість створення ієрархій класів для моделювання реальних відношень, повторне використання коду через наслідування та гнучке управління складними структурами даних через композицію. Поліморфізм проявляється в тому, що методи з однаковими назвами (наприклад, **str**()) можуть мати різну реалізацію в кожному класі, забезпечуючи специфічну поведінку для кожного типу об'єкта. Інкапсуляція забезпечується через приватні атрибути та контрольовані методи доступу, що гарантує цілісність даних та правильність операцій. Така структура легко масштабується та модифікується, дозволяючи додавати нові типи осіб або розширювати функціональність існуючих класів без впливу на інші компоненти системи.

Тестування об'єктно-орієнтованого коду включає перевірку правильності роботи конструкторів, методів класу, взаємодії між об'єктами та коректності реалізації наслідування. Важливо тестувати як нормальні сценарії використання, так і граничні випадки, включаючи некоректні вхідні дані. При створенні об'єктів слід перевіряти правильність ініціалізації всіх атрибутів, а при тестуванні методів - їх вплив на стан об'єкта та повернені значення. Особливу увагу треба приділити тестуванню приватних методів через публічний інтерфейс та перевірці правильності роботи властивостей з getter та setter методами.

2 КЛЮЧОВІ ПИТАННЯ

1. Що таке клас та об'єкт у Python та як вони пов'язані між собою?
2. Як створити клас у Python та яке призначення методу __init__?
3. Як визначити атрибути та методи класу та як з ними працювати?
4. Які рівні доступу до атрибутів та методів існують у Python?
5. Як реалізується інкапсуляція у мові Python?
6. Що таке наслідування та як його використовувати для створення ієрархій класів?
7. Як перевизначити методи батьківського класу у дочірньому класі?
8. Що таке поліморфізм та як він реалізується в Python?
9. Як працює множинне наслідування та які в нього особливості?
10. Яке призначення спеціальних методів (magic methods) у класах Python?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Розробити програму на Python, яка демонструє використання об'єктно-орієнтованого підходу для вирішення задачі згідно з варіантом завдання. Програма повинна включати створення класів з атрибутами та методами, використання конструктора для ініціалізації об'єктів, реалізацію інкапсуляції та, за необхідності, наслідування. Програма повинна створювати об'єкти класів та демонструвати їх взаємодію між собою.

Таблиця 6.2 – Варіанти індивідуальних завдань

№	Завдання
1	Створити клас "Студент" з атрибутами ім'я, вік, курс. Реалізувати методи для встановлення оцінок та обчислення середнього балу.
2	Розробити клас "Автомобіль" з атрибутами марка, модель, рік випуску. Додати методи для запуску двигуна та визначення віку автомобіля.
3	Створити клас "Бібліотека" з колекцією книг. Реалізувати методи для додавання, видалення та пошуку книг.
4	Розробити клас "Банківський рахунок" з методами поповнення, зняття коштів та перевірки балансу.
5	Створити клас "Прямокутник" з атрибутами довжина та ширина. Додати методи для обчислення площі та периметра.
6	Розробити клас "Калькулятор" з методами для виконання базових арифметичних операцій.
7	Створити клас "Телефонна книга" з методами додавання, видалення та пошуку контактів.
8	Розробити клас "Фігура" та дочірні класи "Коло", "Трикутник", "Квадрат" з методами обчислення площі.
9	Створити клас "Магазин" з атрибутами назва, адреса, список товарів. Додати методи управління товарами.
10	Розробити клас "Час" з методами додавання, віднімання часу та переведення у різні формати.
11	Створити клас "Працівник" з атрибутами ім'я, посада, зарплата. Додати методи для розрахунку премії та податків.
12	Розробити клас "Вектор" з методами додавання, віднімання векторів та обчислення довжини.
13	Створити клас "Курс валют" з методами конвертації між різними валютами.
14	Розробити клас "Календар" з методами для роботи з датами та подіями.

15	Створити клас "Ігровий персонаж" з атрибутами здоров'я, рівень, досвід. Додати методи для розвитку персонажа.
16	Розробити клас "Матриця" з методами додавання, множення матриць та транспонування.
17	Створити клас "Температура" з методами конвертації між шкалами Цельсія, Фаренгейта, Кельвіна.
18	Розробити клас "Склад" з методами управління запасами товарів.
19	Створити клас "Літак" з атрибутами модель, швидкість, висота. Додати методи для зльоту, посадки, зміни висоти.
20	Розробити клас "Графік функції" з методами побудови графіків різних математичних функцій.
21	Створити клас "Сортувальник" з методами реалізації різних алгоритмів сортування.
22	Розробити клас "Комплексне число" з методами арифметичних операцій над комплексними числами.
23	Створити клас "Система резервування" для готелю або авіакомпанії.
24	Розробити клас "Статистика" з методами обчислення середнього, медіани, дисперсії набору даних.
25	Створити клас "Генератор паролів" з методами створення паролів різної складності.
26	Розробити клас "Система голосування" з методами підрахунку голосів та визначення переможця.
27	Створити клас "Музична бібліотека" з методами організації та пошуку музичних композицій.
28	Розробити клас "Система тестування" з методами створення тестів та підрахунку результатів.
29	Створити клас "Погода" з методами аналізу метеоданих та прогнозування.
30	Розробити клас "Фінансовий планер" з методами планування доходів та витрат.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов’язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент розробляє програму, що реалізує об’єктно-орієнтований підхід до розв’язання індивідуального завдання. Програма передбачає створення одного або кількох класів з визначеними атрибутами та методами, використання конструктора для ініціалізації об’єктів, застосування принципів інкапсуляції (зокрема приватних атрибутів та властивостей) та, за потреби, наслідування. Особливу увагу приділяється логічній структурі коду, читабельності та дотриманню основних принципів ООП.

У протоколі обов’язково мають бути наведені три ключові елементи:

– завдання, у якому чітко сформульовано індивідуальний варіант роботи згідно з номером студента;

– лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють призначення класів, методів, атрибутів та механізмів взаємодії об’єктів;

– результати виконання програми, представлені у вигляді скріншотів або текстового виводу консолі, що підтверджують коректну роботу коду для різних сценаріїв використання.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 7

Тема: Робота з табличними даними за допомогою бібліотеки Pandas

Мета роботи: навчитися працювати Excel файлами у Pandas DataFrame та опанувати базові операції фільтрації даних

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Основні відомості про бібліотеку Pandas та її об'єкти

Бібліотека Pandas є однією з найважливіших для аналізу та обробки даних у Python. Вона надає зручні інструменти для роботи з табличними даними через об'єкт DataFrame, який дозволяє ефективно виконувати операції фільтрації, сортування, агрегації та інші маніпуляції з даними. Основна перевага Pandas полягає в тому, що вона надає зручний інтерфейс для роботи з даними, які можуть бути представлені у вигляді таблиць, що робить її незамінним інструментом при аналізі великих обсягів даних.

Для початківців важливо зрозуміти, що Pandas працює з даними у вигляді таблиць, подібних до Excel або таблиць SQL. Це дозволяє легко виконувати знайомі операції, такі як фільтрація рядків, вибірка стовпців, сортування даних. Pandas особливо корисна при роботі з великими наборами даних, де ручна обробка була б надто трудомісткою. Бібліотека також забезпечує ефективну роботу з відсутніми даними, автоматичне визначення типів даних та інтеграцію з іншими бібліотеками Python для наукових обчислень.

DataFrame - це двовимірна структура даних, що складається з рядків і стовпців. DataFrame можна уявити як таблицю, де кожен стовпець може мати різний тип даних (рядки, числа, дати тощо). Кожен рядок і стовпець має власну назву, що дозволяє легко звертатися до конкретних елементів даних. DataFrame є основним об'єктом для роботи з табличними даними у Pandas.

Series - це одновимірний масив даних з мітками (індексами). Series можна розглядати як один стовпець DataFrame. Він може містити дані будь-якого типу (цілі числа, рядки, числа з плаваючою точкою тощо). Series часто використовується при виконанні операцій над окремими стовпцями даних або при створенні нових обчислених полів.

Для новачків важливо зрозуміти, що DataFrame і Series тісно пов'язані між собою. Кожен стовпець DataFrame є об'єктом Series, а DataFrame можна уявити як колекцію Series, об'єднаних разом. Ця структура дозволяє легко виконувати операції як над окремими стовпцями, так і над усією таблицею одночасно.

Таблиця 7.1 – Атрибути DataFrame

Атрибут	Опис	Приклад
df.index	Індекси рядків DataFrame	df.index повертає індекси
df.columns	Назви стовпців DataFrame	df.columns повертає список стовпців
df.values	Значення DataFrame у вигляді масиву NumPy	df.values повертає масив
df.dtypes	Типи даних кожного стовпця	df.dtypes показує типи
df.shape	Розмірність DataFrame (рядки, стовпці)	df.shape повертає (100, 5)

Атрибути DataFrame є важливими інструментами для розуміння структури даних. Наприклад, df.shape дозволяє швидко дізнатися розмір таблиці, що корисно при роботі з великими наборами даних. Атрибут df.dtypes допомагає зрозуміти, які типи даних містяться в кожному стовпці, що важливо для подальшої обробки та аналізу. Для початківців рекомендується регулярно перевіряти ці атрибути для кращого розуміння даних, з якими вони працюють.

Робота з Excel файлами

Для завантаження даних з Excel файлів використовується функція pd.read_excel(). Вона дозволяє зчитувати дані з Excel файлів з різними параметрами налаштування. Ця функція є однією з найбільш корисних при роботі з реальними бізнес-даними, оскільки багато організацій зберігають інформацію саме у форматі Excel.

Таблиця 7.2 – Аргументи функції pd.read_excel()

Параметр	Опис	Приклад
sheet_name	Назва або індекс листа	sheet_name='Sheet1'
usecols	Які стовпці читати	usecols=['A:D']
skiprows	Скільки рядків пропустити	skiprows=1
na_values	Які значення вважати відсутніми	na_values=['NA', 'n/a']
dtype	Словник типів даних для стовпців	dtype={'price': 'float64'}
header	Номер рядка з заголовками	header=0

При роботі з Excel файлами важливо звертати увагу на структуру файлу. Іноді файли містять зайві рядки з заголовками або примітками, які потрібно пропустити за допомогою параметра skiprows. Також важливо правильно вказати типи даних для стовпців, особливо при роботі з числовими даними, щоб уникнути проблем з подальшими обчисленнями.

Приклад завантаження даних:

```
import pandas as pd

# Завантаження даних з Excel файлу
df = pd.read_excel('products.xlsx')

# Завантаження з додатковими параметрами
df = pd.read_excel(
    'products.xlsx',
    sheet_name='Sheet1',
    usecols=['A:E'],
```

```
    skiprows=1,
    na_values=['NA', 'n/a']
)
```

Фільтрація даних

Фільтрація даних у Pandas виконується за допомогою логічних умов. Ви можете фільтрувати дані за окремими стовпцями або комбінаціями умов. Це одна з найважливіших операцій при аналізі даних, оскільки дозволяє виділяти потрібну інформацію з великих наборів даних.

Таблиця 7.3 – Фільтрація даних у Pandas

Операція	Опис	Приклад
==	Рівність	df[df['category'] == 'Phone']
&	Логічне "і"	df[(df['price'] >= 500) & (df['category'] == 'Phone')]
`	`	Логічне "або"
isin()	Перевірка на входження в список	df[df['brand'].isin(['Apple', 'Samsung'])]
str.contains()	Пошук підрядка	df[df['name'].str.contains('iPhone')]

При фільтрації даних важливо пам'ятати про правильне використання дужок при комбінуванні умов. Кожна умова повинна бути взята в дужки, а для комбінування умов використовуються оператори & (і) та | (або). Це особливо важливо для початківців, оскільки неправильне використання операторів може призвести до неочікуваних результатів.

Приклад фільтрації даних:

```
# Фільтрація за категорією та ціною
filtered_df = df[
    (df['category_id'] == 'Телефон') &
    (df['price'] >= 200) &
    (df['price'] <= 500)
]

# Фільтрація за брендом
brand_filtered = df[df['brand_id'] == 'Apple']
```

Сортування даних

Сортування даних виконується за допомогою методу sort_values(). Можна сортувати за одним або кількома стовпцями. Сортування є важливим етапом при підготовці даних для аналізу або представлення результатів.

Таблиця 7.4 – Сортування даних у Pandas

Параметр	Опис	Приклад
by	Стовпець або список стовпців для сортування	by='price'
ascending	Порядок сортування (True - за зростанням)	ascending=False
na_position	Позиція для відсутніх значень	na_position='last'

Сортування дозволяє організувати дані у зручному для аналізу порядку. Наприклад, сортування за ціною дозволяє швидко знайти найдешевші або найдорожчі товари. При сортуванні за кількома стовпцями важливо звертати увагу на порядок стовпців у списку - перший стовпець має найвищий пріоритет при сортуванні.

Приклад сортування:

```
# Сортування за ціною за зростанням
sorted_df = df.sort_values('price')

# Сортування за кількома стовпцями
multi_sorted = df.sort_values(['category_id', 'price'],
ascending=[True, False])

# Сортування з обробкою відсутніх значень
sorted_with_na = df.sort_values('price', na_position='last')
```

Додавання та модифікація стовпців

У Pandas можна легко додавати нові стовпці або модифікувати існуючі. Це дозволяє створювати нові обчислені поля на основі існуючих даних, що є важливим етапом при аналізі даних.

Таблиця 7.5 – Додавання стовпців у Pandas

Операція	**Опис**	**Приклад**
Просте додавання	Створення нового стовпця	df['price_uah'] = df['price'] * 39.5
Умовне додавання	Використання np.where	df['price_category'] = np.where(df['price'] > 1000, 'High', 'Low')
Функції для рядків	Застосування методів str	df['name_upper'] = df['name'].str.upper()
Комбінування	Об'єднання стовпців	df['full_name'] = df['brand'] + ' ' + df['model']

Додавання нових стовпців дозволяє розширити інформативність даних. Наприклад, можна додати стовпець з цінами в іншій валюті, категорії товарів або інші обчислені показники. Для початківців важливо зрозуміти, що нові стовпці додаються автоматично, і не потрібно попередньо визначати їх структуру.

Приклад додавання стовпця:

```
# Додавання стовпця з цінами в гривнях (курс 39.5)
df['price_uah'] = df['price'] * 39.5

# Умовне додавання категорій цін
```

```
import numpy as np
df['price_category'] = np.where(df['price'] > 1000, 'Висока',
'Середня')
```

Статистичні операції

Pandas надає зручні методи для виконання статистичних операцій. Ці операції дозволяють швидко отримати загальну інформацію про дані, що особливо корисно при первинному аналізі набору даних.

Таблиця 7.6 – Статистичні операції в Pandas

Метод	Опис	Приклад
describe()	Основні статистичні показники	df.describe()
groupby()	Групування даних	df.groupby('category').agg({'price': 'mean'})
value_counts()	Підрахунок унікальних значень	df['category'].value_counts()

Статистичні операції дозволяють швидко отримати уявлення про розподіл даних. Наприклад, метод describe() показує основні статистичні характеристики числових стовпців, що допомагає зрозуміти діапазон значень, середнє значення та інші важливі параметри.

Приклад статистичних операцій:

```
# Основні статистичні показники
statistics = df.describe()

# Групування за категоріями з агрегацією
category_stats = df.groupby('category_id').agg({
    'price': ['mean', 'min', 'max'],
    'brand_id': 'count'
})

# Підрахунок товарів по категоріях
category_counts = df['category_id'].value_counts()
```

Збереження даних в Excel

Для збереження результатів у Excel файл використовується метод to_excel(). Це дозволяє зберігати оброблені дані для подальшого використання або передачі іншим користувачам.

Таблиця 7.7 – Аргументи методу to_excel()

Параметр	Опис	Приклад
sheet_name	Назва листа	sheet_name='Результати'
index	Чи зберігати індекси	index=False
header	Чи зберігати заголовки	header=True
float_format	Формат чисел з плаваючою точкою	float_format='%.2f'
na_rep	Чим замінювати відсутні значення	na_rep='-'
freeze_panes	Які рядки/стовпці закріпити	freeze_panes=(1, 0)

При збереженні даних важливо враховувати форматування та структуру файлу. Для зручності подальшого використання рекомендується вказувати зрозумілі назви листів та форматувати числові дані у відповідному форматі.

Приклад збереження:

```
# Збереження результатів
filtered_df.to_excel('результат.xlsx', index=False)

# Збереження з додатковими параметрами
filtered_df.to_excel(
    'результат.xlsx',
    sheet_name='Товари',
    index=False,
    float_format='%.2f'
)
```

Корисні команди для аналізу даних

Ці команди є незамінними при роботі з новими наборами даних. Вони дозволяють швидко ознайомитися зі структурою даних, перевірити наявність проблем та створити резервні копії для безпечної роботи.

Таблиця 7.8 – Команди для аналізу даних

Команда	Опис	Приклад
df.info()	Загальна інформація про DataFrame	df.info()
df.head()	Перші 5 рядків	df.head()
df.tail()	Останні 5 рядків	df.tail()
df.isnull().sum()	Перевірка відсутніх значень	df.isnull().sum()
df.copy()	Створення копії DataFrame	df_copy = df.copy()

Приклад аналізу даних:

```
# Отримання інформації про структуру даних
print(df.info())

# Перегляд перших рядків
print(df.head())

# Перевірка відсутніх значень
print(df.isnull().sum())
```

Приклад виконання завдання

Розглянемо приклад виконання завдання для варіанту 1 (Категорія: Телефон, Діапазон цін: 200-500, Додаткова умова: Сортування за ціною за зростанням). Цей приклад демонструє повний цикл роботи з даними від завантаження до збереження результатів.

```
import pandas as pd
import numpy as np

# Завантаження даних
df = pd.read_excel('products.xlsx')
```

```
# Фільтрація за категорією та ціною
filtered_df = df[
    (df['category_id'] == 'Телефон') &
    (df['price'] >= 200) &
    (df['price'] <= 500)
]

# Сортування за ціною за зростанням
sorted_df = filtered_df.sort_values('price', ascending=True)

# Додавання стовпця з цінами в гривнях
sorted_df['price_uah'] = sorted_df['price'] * 39.5

# Збереження результатів
sorted_df.to_excel('результат.xlsx', index=False)

print("Фільтрація завершена. Знайдено {} товарів.".format(len(sorted_df)))
```

Цей приклад показує, як поєднуються різні операції Pandas для вирішення конкретної задачі. Для початківців важливо зрозуміти, що кожна операція виконується послідовно, і результат кожної операції стає вхідними даними для наступної. Такий підхід дозволяє легко модифікувати код для інших варіантів завдання, змінюючи параметри фільтрації та сортування відповідно до вимог конкретного варіанту.

При роботі з реальними даними важливо завжди перевіряти проміжні результати, щоб переконатися в правильності виконання операцій. Для цього можна використовувати команди head(), tail() або просто виводити DataFrame на екран для візуального контролю результатів.

2 КЛЮЧОВІ ПИТАННЯ

1. Які основні об'єкти бібліотеки Pandas використовуються для роботи з табличними даними та які їх основні характеристики?
2. Які параметри функції `pd.read_excel()` найчастіше використовуються при завантаженні даних з Excel файлів та яке їх призначення?
3. Як виконується фільтрація даних у DataFrame за допомогою логічних умов та які оператори використовуються для комбінування умов?
4. Які методи сортування даних доступні у Pandas та як можна сортувати дані за кількома стовпцями одночасно?
5. Як додаються нові стовпці до DataFrame та які способи створення обчислених полів існують?
6. Які статистичні методи надає Pandas для аналізу даних та як виконується групування даних з агрегацією?
7. Які параметри функції `to_excel()` дозволяють налаштувати збереження даних у Excel файл та яке їх практичне застосування?

8. Які методи використовуються для аналізу структури даних та перевірки наявності відсутніх значень у DataFrame?

9. Як виконується умовне додавання даних до стовпців та які функції бібліотеки NumPy використовуються для цього?

10. Які етапи включає повний цикл обробки даних від завантаження до збереження результатів та яке призначення кожного етапу?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Завантаження даних з Excel-файлу. Для виконання завдання надається Excel-файл products.xlsx, який містить табличні дані про товари. Структура файлу включає такі стовпці:

- id — порядковий номер товару (ціле число);
- name — назва товару (рядок);
- description — короткий опис товару (рядок);
- price — ціна товару в доларах США (число з плаваючою комою);
- category_id — категорія товару (рядок, наприклад, «Телефон», «Ноутбук» тощо);
- brand_id — назва виробника (рядок, наприклад, «Apple», «Samsung»);
- features — розширений опис характеристик товару (рядок).

На початку роботи необхідно завантажити таблицю даних з файлу products.xlsx за допомогою функції pd.read_excel(). Переконайтеся, що файл розташований у тій самій директорії, що й ваш Python-скрипт. Під час завантаження вкажіть назву листа (за замовчуванням — перший), а також при потребі налаштуйте параметри, такі як пропуск рядків заголовка або обробка відсутніх значень. Після завантаження рекомендується вивести перші кілька рядків DataFrame за допомогою методу .head() для перевірки коректності імпорту.

3.2 Фільтрація даних за категорією та ціновим діапазоном. Виконайте фільтрацію записів відповідно до вашого варіанту, вихідні дані для якого наведено в табл. 7.9**.** Необхідно обрати лише ті рядки, у яких значення в стовпці *category_id* відповідає заданій категорії (наприклад, «Телефон»), а ціна (*price*) знаходиться в межах визначеного для варіанту діапазону (наприклад, від 200 до 500 доларів США). Для реалізації фільтрації використовуйте логічні умови з операторами порівняння, поєднуючи їх за допомогою оператора `&` та обов’язково застосовуючи дужки для кожної окремої умови.

3.3 Виконання додаткової умови (сортування або фільтрація за брендом). Після основної фільтрації застосуйте додаткову операцію, передбачену вашим варіантом. Якщо варіант передбачає сортування, скористайтеся методом .sort_values(), вказавши стовпець для сортування та напрямок (за зростанням або спаданням). Якщо ж потрібно відфільтрувати записи за конкретним брендом (наприклад, brand_id == 'Apple'), застосуйте

додаткову умову аналогічно до попереднього кроку. Це дозволить отримати остаточний набір даних, що повністю відповідає вимогам завдання.

3.4 Додавання нового стовпця з цінами в гривнях. Створіть новий стовпець price_uah у відфільтрованому DataFrame, який міститиме ціни, конвертовані з доларів США в українські гривні. Для цього помножте значення стовпця price на курс 39.5 (або інший актуальний курс, якщо вказано інструкцією). Операція виконується векторизовано — без циклів — завдяки вбудованій підтримці арифметичних операцій над стовпцями у Pandas.

3.5 Збереження результатів у новий Excel-файл. Збережіть остаточну таблицю у новий файл з назвою результат.xlsx за допомогою методу .to_excel(). При цьому вкажіть параметр index=False, щоб не зберігати автоматично згенеровані індекси рядків, і встановіть float_format='%.2f', щоб числа з плаваючою комою відображалися з двома знаками після коми. Переконайтеся, що файл успішно створено, і відкрийте його в Excel для візуальної перевірки структури та коректності даних.

3.6 Аналіз та документування результатів. У звіті наведіть фрагмент вихідних даних (перші 5 рядків з файлу products.xlsx), а також фрагмент результуючої таблиці (перші 5 відфільтрованих записів). Додайте коментарі щодо того, як саме були виконані фільтрація, сортування та конвертація валют. Також вкажіть кількість записів, що залишилися після фільтрації, і поясніть, чи відповідають результати очікуванням. Це допоможе продемонструвати розуміння процесу обробки табличних даних за допомогою Pandas.

Таблиця 7.9 – Варіанти індивідуальних завдань

Варіант	Категорія	Діапазон цін (USD)	Додаткова умова
1	Телефон	200 - 500	Сортування за ціною за зростанням
2	Ноутбук	800 - 1500	Фільтр по brand_id = 'Apple'
3	Роутер	50 - 150	Сортування за назвою
4	Планшет	300 - 700	Фільтр по brand_id = 'Samsung'
5	Монітор	200 - 400	Сортування за ціною за спаданням
6	Телефон	500 - 1000	Фільтр по brand_id = 'Xiaomi'
7	Ноутбук	1500 - 2500	Сортування за назвою
8	Роутер	150 - 300	Фільтр по brand_id = 'TP-Link'
9	Планшет	700 - 1200	Сортування за ціною за зростанням
10	Монітор	400 - 800	Фільтр по brand_id = 'LG'
11	Телефон	300 - 800	Сортування за brand_id
12	Ноутбук	1000 - 2000	Фільтр по brand_id = 'Lenovo'
13	Роутер	100 - 200	Сортування за ціною за спаданням
14	Планшет	400 - 900	Фільтр по brand_id = 'Huawei'
15	Монітор	300 - 600	Сортування за назвою
16	Телефон	400 - 900	Фільтр по brand_id = 'OnePlus'
17	Ноутбук	1200 - 2200	Сортування за ціною за зростанням
18	Роутер	75 - 250	Фільтр по brand_id = 'D-Link'
19	Планшет	500 - 1000	Сортування за brand_id

20	Монітор	350 - 750	Фільтр по brand_id = 'Dell'
21	Телефон	600 - 1200	Сортування за ціною за спаданням
22	Ноутбук	900 - 1800	Фільтр по brand_id = 'Asus'
23	Роутер	125 - 275	Сортування за назвою
24	Планшет	600 - 1100	Фільтр по brand_id = 'Lenovo'
25	Монітор	250 - 550	Сортування за ціною за зростанням
26	Телефон	700 - 1300	Фільтр по brand_id = 'Sony'
27	Ноутбук	1300 - 2300	Сортування за brand_id
28	Роутер	200 - 350	Фільтр по brand_id = 'Mikrotik'
29	Планшет	800 - 1300	Сортування за ціною за спаданням
30	Монітор	450 - 850	Фільтр по brand_id = 'Samsung'

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент реалізує програму, призначену для обробки табличних даних з Excel-файлу за допомогою бібліотеки Pandas. Програма передбачає завантаження даних, їх фільтрацію за заданими критеріями (категорія, ціновий діапазон, бренд тощо), застосування додаткових операцій (наприклад, сортування або створення нових стовпців) та збереження результатів у новий Excel-файл. Особливу увагу приділяється коректному використанню методів pandas для читання, фільтрації, модифікації та запису даних.

У протоколі обов'язково мають бути наведені три ключові елементи:

– завдання, у якому чітко вказано індивідуальний варіант (категорія товарів, ціновий діапазон, додаткова умова);

– лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють кожен етап обробки даних — від завантаження файлу до збереження результатів;

– результати виконання програми, представлені у вигляді фрагментів вхідних і вихідних даних (текстовий вивід або скріншоти Excel-файлів), що підтверджують коректну роботу коду.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у

процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 8

Тема: Основи роботи з матрицями та розв'язання СЛАР в NumPy

Мета роботи: Навчитися створювати та перетворювати квадратні матриці, виконувати базові матричні операції та розв'язувати системи лінійних алгебраїчних рівнянь (СЛАР) за допомогою бібліотеки NumPy.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Загальна інформація про модуль NumPy та його основні функції

NumPy (Numerical Python) була створена у 2005 році Тревісом Олифантом як еволюція попередніх бібліотек Numeric та Numarray. Головною метою було створення єдиної, потужної та ефективної бібліотеки для наукових обчислень у Python. NumPy стала фундаментом для екосистеми наукових обчислень Python, включаючи такі популярні бібліотеки як SciPy, pandas, scikit-learn та matplotlib. Сьогодні NumPy є стандартом де-факто для роботи з багатовимірними масивами та матрицями в Python.

Матриця в математиці та програмуванні є прямокутною таблицею чисел, організованих у рядки та стовпці. Розмірність матриці характеризується парою чисел (m × n), де m — кількість рядків, n — кількість стовпців. За формою матриці поділяються на квадратні ($m = n$), прямокутні ($m \neq n$), рядкові ($m = 1$) та стовпцеві ($n = 1$). За змістом розрізняють нульові матриці (всі елементи дорівнюють нулю), одиничні матриці (одиниці на головній діагоналі, нулі в інших позиціях), діагональні матриці (ненульові елементи тільки на діагоналі), симетричні матриці ($A_{ij} = A_{ji}$), верхньотрикутні та нижньотрикутні матриці.

Створення матриць у NumPy здійснюється за допомогою функції numpy.array(), яка перетворює вкладені списки Python у ефективні багатовимірні масиви з однорідними типами даних. NumPy автоматично визначає оптимальний тип даних або дозволяє його явно вказати через параметр dtype.

Таблиця 8.1 — Основні функції створення матриць у NumPy

Функція	Опис	Приклад використання	Коментар
np.array()	Створення масиву з списку або кортежу	A = np.array([[1, 2], [3, 4]])	Основний спосіб створення матриць
np.zeros()	Створення матриці з нулів	Z = np.zeros((3, 3))	Корисно для ініціалізації
np.ones()	Створення матриці з одиниць	O = np.ones((2, 4))	Часто використовується як базова матриця
np.eye()	Створення одиничної матриці	I = np.eye(3)	Тільки для квадратних матриць
np.diag()	Створення діагональної матриці	D = np.diag([1, 2, 3])	Можна також витягти діагональ

np.random.rand()	Матриця випадкових чисел [0,1)	R = np.random.rand(2, 3)	Рівномірний розподіл
np.random.randn()	Матриця нормально розподілених чисел	N = np.random.randn(3, 3)	Стандартний нормальний розподіл
np.arange()	Створення послідовності чисел	A = np.arange(12).reshape(3, 4)	Потребує reshape для матриці

Таблиця 8.2 — Основні операції з матрицями в NumPy

Операція	Функція/Оператор	Приклад	Коментар
Додавання	+ або np.add()	C = A + B	Поелементне додавання
Віднімання	- або np.subtract()	C = A - B	Поелементне віднімання
Множення поелементне	* або np.multiply()	C = A * B	Множення Адамара
Матричне множення	@ або np.dot()	C = A @ B	Математичне матричне множення
Ділення поелементне	/ або np.divide()	C = A / B	Поелементне ділення
Транспонування	.T або np.transpose()	At = A.T	Заміна рядків на стовпці
Визначник	np.linalg.det()	det_A = np.linalg.det(A)	Тільки для квадратних матриць
Обернена матриця	np.linalg.inv()	A_inv = np.linalg.inv(A)	Існує тільки для невироджених матриць
Ранг матриці	np.linalg.matrix_rank()	rank_A = np.linalg.matrix_rank(A)	Кількість лінійно незалежних рядків
Власні значення	np.linalg.eig()	w, v = np.linalg.eig(A)	Повертає значення та вектори

Визначник квадратної матриці A порядку n обчислюється за формулою:

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)},$$

де S_n — множина всіх перестановок чисел від 1 до n, $\operatorname{sgn}(\sigma)$ — знак перестановки. Геометрично визначник показує, у скільки разів змінюється об’єм при лінійному перетворенні, заданому матрицею. Обернена матриця A^{-1} існує тільки для невироджених квадратних матриць (коли $\det(A) \neq 0$) і задовольняє умову: $A \cdot A^{-1} = A^{-1} \cdot A = I$ де I — одинична матриця відповідного порядку.

Системи лінійних алгебраїчних рівнянь та їх розв'язання в NumPy

Система лінійних алгебраїчних рівнянь (СЛАР) є фундаментальним об'єктом вивчення лінійної алгебри та має широке застосування в інженерії,

фізиці, економіці та інших областях. Загальний вигляд СЛАР з n невідомими та m рівняннями:

Системи лінійних алгебраїчних рівнянь та їх розв’язання в NumPy Система лінійних алгебраїчних рівнянь (СЛАР) є фундаментальним об’єктом вивчення лінійної алгебри та має широке застосування в інженерії, фізиці, економіці та інших областях. Загальний вигляд СЛАР з n невідомими та m рівняннями:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

У матричному записі ця система має вигляд $Ax = b$, де A — матриця коефіцієнтів розміром $m \times n$, x — вектор невідомих розміром $n \times 1$, b — вектор вільних членів розміром $m \times 1$. NumPy надає декілька методів розв’язання СЛАР залежно від властивостей системи. Для квадратних невироджених систем (коли $m = n$ і $\det(A) \neq 0$) існує єдиний розв’язок, який теоретично можна знайти за формулою $x = A^{-1}b$. Однак NumPy використовує більш ефективні та чисельно стабільні алгоритми.

Таблиця 8.3 – Методи розв'язання СЛАР у NumPy

Метод	Функція	Застосування	Коментар
Пряме розв'язання	np.linalg.solve(A, b)	Квадратні невироджені системи	Найефективніший метод
Метод найменших квадратів	np.linalg.lstsq(A, b)	Перевизначені/недовизначені системи	Мінімізує функцію похибки
Псевдообернена матриця	np.linalg.pinv(A) @ b	Сингулярні системи	Використовує SVD-розклад

Функція np.linalg.solve() є найбільш розповсюдженим та ефективним методом розв’язання квадратних систем. Вона використовує LU-розклад з частковим вибором головного елемента, що забезпечує чисельну стабільність та високу швидкість обчислень. Алгоритм розкладає матрицю A на добуток нижньотрикутної матриці L та верхньотрикутної матриці U: $A = LU$.

Для систем, де кількість рівнянь не дорівнює кількості невідомих, використовується метод найменших квадратів, який мінімізує евклідову норму похибки $\| Ax - b \|^2$. Псевдообернена матриця застосовується у випадках, коли матриця A є сингулярною або близькою до сингулярної.

Приклад розв'язання СЛАР у Python з NumPy:

```
import numpy as np

# Задаємо систему рівнянь:
# 2x + 3y + z = 1
# x + 4y + 5z = 2
```

```
# 3x + y + 2z = 3

# Матриця коефіцієнтів A
A = np.array([[2, 3, 1],
              [1, 4, 5],
              [3, 1, 2]])

# Вектор вільних членів b
b = np.array([1, 2, 3])

# Перевіряємо, чи система має єдиний розв'язок
det_A = np.linalg.det(A)
print(f"Визначник матриці A: {det_A:.6f}")

if abs(det_A) > 1e-10:  # Матриця невироджена
    # Розв'язуємо систему
    x = np.linalg.solve(A, b)
    print(f"Розв'язок системи: x = {x}")

    # Перевірка правильності розв'язку
    check = A @ x  # Обчислюємо Ax
    print(f"Перевірка A*x = {check}")
    print(f"Вектор b =     {b}")
    print(f"Різниця = {np.abs(check - b)}")

    # Перевіряємо точність з допуском
    if np.allclose(check, b):
        print("Розв'язок правильний!")
    else:
        print("Помилка в розв'язку!")

else:
    print("Матриця вироджена, система не має єдиного розв'язку")

# Альтернативний метод через обернену матрицю (менш ефективний)
try:
    A_inv = np.linalg.inv(A)
    x_alt = A_inv @ b
    print(f"Розв'язок через обернену матрицю: {x_alt}")
except np.linalg.LinAlgError:
    print("Не вдалося обчислити обернену матрицю")
```

Цей приклад демонструє повний цикл роботи з СЛАР: створення матриці коефіцієнтів та вектора вільних членів, перевірку на виродженість, розв'язання системи, верифікацію результату та альтернативний метод через обернену матриці. Важливо відзначити використання функції np.allclose() для порівняння результатів з урахуванням похибок округлення, що є критично важливим при роботі з числами з плаваючою комою.

2 КЛЮЧОВІ ПИТАННЯ

1. Що таке бібліотека NumPy і яке її значення для наукових обчислень у Python?
2. Як створюється двовимірний масив (матриця) у NumPy за допомогою функції np.array()? Які ще існують способи створення матриць?
3. У чому різниця між поелементним множенням (*) і матричним множенням (@) у NumPy?
4. Що таке визначник квадратної матриці і як він пов'язаний з існуванням оберненої матриці?
5. Як обчислити визначник і знайти обернену матрицю за допомогою функцій numpy.linalg.det() та numpy.linalg.inv()?
6. Що таке транспонування матриці і як його виконати у NumPy?
7. У якому вигляді записується система лінійних алгебраїчних рівнянь (СЛАР) у матричній формі?
8. Яка функція у NumPy використовується для прямого розв'язання квадратних невироджених систем лінійних рівнянь і чому вона є найефективнішою?
9. Як перевірити правильність знайденого розв'язку СЛАР у Python, враховуючи похибки округлення чисел з плаваючою комою?
10. Чому не рекомендується використовувати обернену матрицю (A_inv @ b) для розв'язання СЛАР замість numpy.linalg.solve()?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Робота з квадратною матрицею

Створіть квадратну матрицю A розмірністю 3x3 згідно з варіантом. Для цього використовуйте функцію numpy.array(). Виведіть створену матрицю. Обчисліть та виведіть визначник матриці A за допомогою функції numpy.linalg.det(). Перевірте, чи є матриця A оборотною (визначник не дорівнює нулю). Якщо матриця оборотна, знайдіть її обернену матрицю A^{-1} за допомогою функції numpy.linalg.inv() та виведіть результат. Обчисліть добуток вихідної матриці A на її обернену матрицю A^{-1} за допомогою numpy.dot() або оператора @. Переконайтеся, що результатом є одинична матриця (з урахуванням похибки обчислень). Виконайте транспонування матриці A за допомогою .T або numpy.transpose() та виведіть транспоновану матрицю.

Таблиця 8.4 – Варіанти індивідуальних завдань для створення матриць

№	Матриця A (3X3)	№	Матриця A (3X3)
1	[[1, 2, 3], [0, 1, 4], [5, 6, 0]]	16	[[1, 4, 2], [3, 1, 1], [2, 2, 3]]
2	[[2, -1, 0], [3, 4, 1], [1, 2, 3]]	17	[[2, 0, 1], [1, 3, 2], [3, 1, 0]]
3	[[4, 2, 1], [1, 3, 2], [2, 1, 4]]	18	[[4, 2, 3], [1, 5, 1], [3, 1, 2]]
4	[[0, 1, 2], [3, 0, 1], [1, 2, 0]]	19	[[1, 1, 2], [2, 3, 1], [3, 1, 1]]
5	[[1, 0, 1], [2, 1, 0], [0, 2, 1]]	20	[[0, 3, 1], [2, 0, 4], [1, 2, 0]]
6	[[3, 1, 2], [1, 2, 3], [2, 3, 1]]	21	[[3, 1, 1], [2, 4, 2], [1, 1, 3]]
7	[[1, 1, 0], [0, 1, 1], [1, 0, 1]]	22	[[2, 2, 1], [1, 3, 2], [3, 1, 1]]

8	[[2, 3, 1], [4, 0, 2], [1, 1, 3]]	23	[[1, 0, 3], [2, 1, 0], [0, 3, 1]]
9	[[5, 1, 2], [3, 4, 1], [1, 1, 2]]	24	[[5, 2, 1], [3, 3, 2], [1, 1, 4]]
10	[[1, 3, 2], [2, 1, 3], [3, 2, 1]]	25	[[1, 2, 3], [3, 1, 2], [2, 3, 1]]
11	[[0, 2, 1], [1, 0, 3], [2, 1, 0]]	26	[[0, 1, 3], [4, 0, 1], [1, 3, 0]]
12	[[4, 1, 1], [1, 3, 2], [2, 1, 3]]	27	[[4, 1, 2], [1, 2, 1], [2, 3, 4]]
13	[[2, 1, 3], [3, 2, 1], [1, 3, 2]]	28	[[3, 3, 1], [2, 1, 3], [1, 2, 2]]
14	[[1, 2, 0], [0, 1, 2], [2, 0, 1]]	29	[[2, 1, 0], [0, 2, 1], [1, 0, 2]]
15	[[3, 2, 1], [1, 4, 3], [2, 1, 1]]	30	[[6, 1, 2], [2, 5, 3], [1, 1, 1]]

3.2 Розв’язання системи лінійних алгебраїчних рівнянь (СЛАР)

Розв’яжіть систему лінійних алгебраїчних рівнянь (СЛАР) згідно з варіантом, використовуючи матричний метод. Систему рівнянь можна записати у вигляді матричного рівняння Ax = b, де A — це матриця коефіцієнтів при невідомих (яку ви використовували в першій частині), x — вектор невідомих (x_1, x_2, x_3), a b — вектор вільних членів. Створіть вектор вільних членів b згідно з варіантом, використовуючи numpy.array(). Переконайтеся, що матриця A є невиродженою (її визначник не дорівнює нулю), що було перевірено в першій частині. Розв’яжіть систему за допомогою функції numpy.linalg.solve(A, b). Виведіть отриманий вектор розв’язків x. Для перевірки підставте знайдений розв’язок назад у вихідну систему рівнянь (обчисліть добуток A@x) і порівняйте результат з вектором b.

Таблиця 8.5 – Варіанти індивідуальних завдань для вирішення СЛАР

№ варіанту	Вектор вільних членів B	№ варіанту	Вектор вільних членів B
1	[9, 4, 11]	16	[7, 5, 9]
2	[1, 10, 9]	17	[1, 7, 3]
3	[7, 6, 8]	18	[9, 6, 7]
4	[5, 4, 3]	19	[4, 7, 5]
5	[1, 4, 2]	20	[4, 8, 3]
6	[6, 6, 6]	21	[6, 8, 4]
7	[2, 1, 2]	22	[5, 6, 4]
8	[6, 8, 7]	23	[3, 4, 3]
9	[8, 8, 4]	24	[8, 10, 6]
10	[5, 5, 5]	25	[6, 6, 6]
11	[3, 7, 4]	26	[4, 5, 4]
12	[6, 5, 7]	27	[7, 4, 9]
13	[6, 6, 6]	28	[7, 7, 7]
14	[2, 2, 2]	29	[2, 1, 3]
15	[6, 10, 5]	30	[9, 10, 3]

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов’язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У цьому розділі студент реалізує дві частини роботи. У першій частині він створює матрицю A згідно з варіантом, обчислює її визначник, перевіряє на оборотність, знаходить обернену матрицю (якщо вона існує), виконує перевірку добутком на обернену матрицю та транспонує вихідну матрицю, виводячи всі результати. У другій частині він створює вектор вільних членів b, розв’язує СЛАР за допомогою np.linalg.solve(), виводить вектор розв’язків x і обов’язково виконує перевірку правильності розв’язку шляхом обчислення A@x і порівняння з b, бажано з використанням функції np.allclose() для коректної роботи з похибками округлення.

У протоколі обов’язково мають бути наведені три ключові елементи:

– завдання, у якому чітко вказано індивідуальний варіант (матриця A та вектор b);

– лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють кожен етап обчислень — від створення матриць до перевірки розв’язку;

– результати виконання програми, представлені у вигляді текстового виводу консолі (або скріншотів), що підтверджують коректну роботу коду та точність отриманих результатів.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 9

Тема: Візуалізація даних за допомогою бібліотеки Matplotlib

Мета роботи: Набути практичних навичок побудови та оформлення графіків у Python за допомогою бібліотеки Matplotlib.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Matplotlib є однією з найпопулярніших бібліотек Python для створення статичних, анімованих та інтерактивних візуалізацій даних. Бібліотека була створена Джоном Хантером у 2003 році з метою надання Python-спільноті інструменту для наукової візуалізації, який за функціональністю міг би конкурувати з комерційними пакетами на кшталт MATLAB.

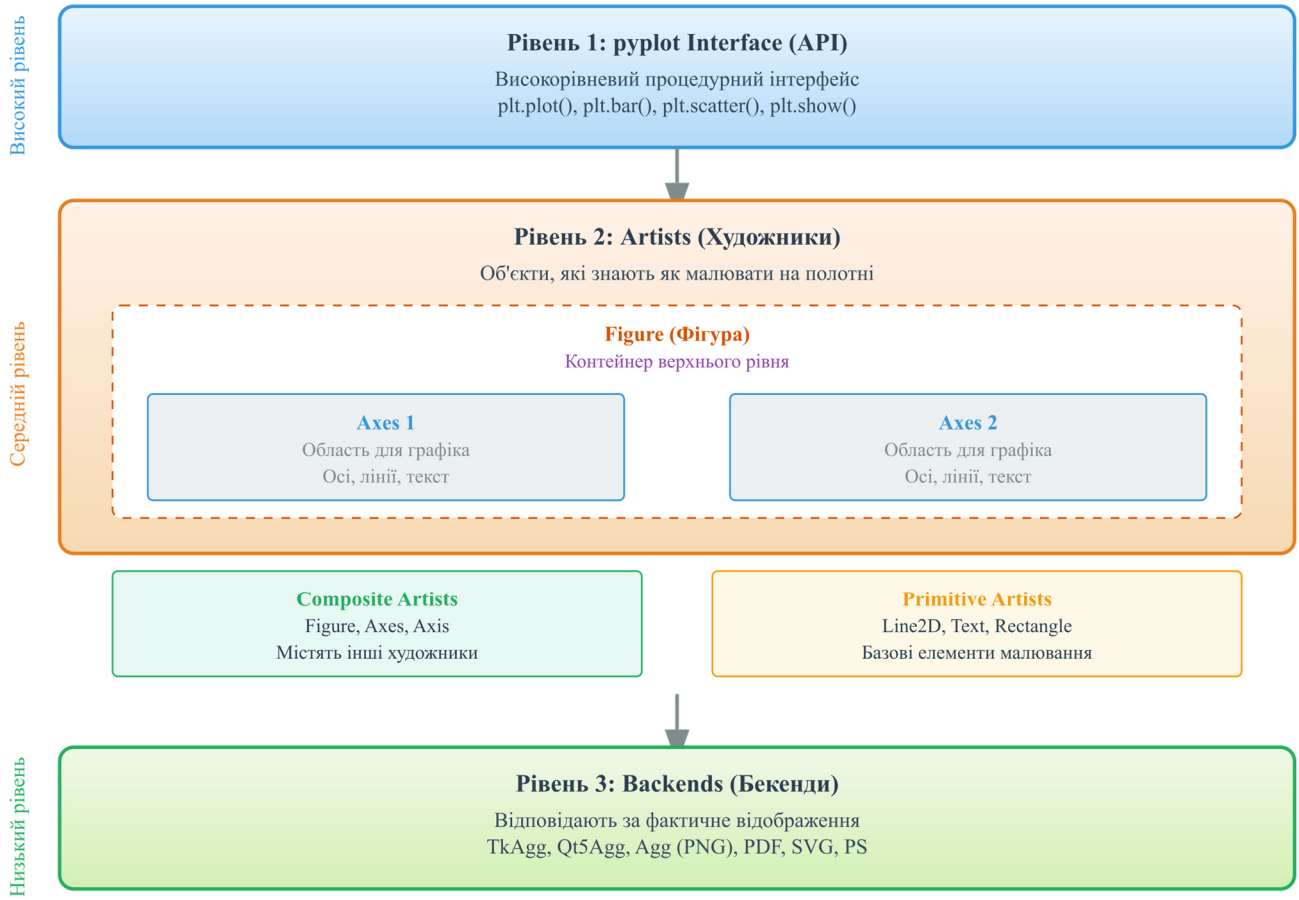
Початковою мотивацією для створення matplotlib було забезпечення дослідників та аналітиків потужним інструментом для графічного представлення результатів наукових досліджень. За більш ніж двадцять років розвитку бібліотека еволюціонувала від простого інструменту для побудови графіків до комплексної екосистеми візуалізації, яка підтримує широкий спектр форматів виводу та інтеграцію з іншими науковими бібліотеками Python.

Сучасна matplotlib підтримує понад 20 типів графіків, від базових лінійних діаграм до складних 3D-візуалізацій. Бібліотека активно розвивається завдяки відкритій спільноті розробників та широко використовується в наукових дослідженнях, аналізі даних, машинному навчанні та веб-розробці. Інтеграція з numpy, pandas та іншими ключовими бібліотеками Python робить matplotlib стандартом де-факто для візуалізації даних у Python-екосистемі.

Архітектура matplotlib побудована на трьох основних рівнях абстракції, що забезпечують гнучкість та потужність бібліотеки. Найнижчий рівень представлений бекендами (backends), які відповідають за фактичне відображення графіків на різних пристроях виводу, включаючи екран, файли та веб-інтерфейси.

Середній рівень складають художники (artists) – об'єкти, які знають, як використовувати бекенди для малювання на полотні. Всі елементи графіка у matplotlib є художниками, від простих ліній до складних анотацій. Художники поділяються на два типи: примітивні (primitive artists), які малюють основні елементи як лінії та текст, та композитні (composite artists), які містять інші художники та координують їх роботу.

Найвищий рівень представлений інтерфейсом pyplot, який надає зручний доступ до функціональності matplotlib через простий процедурний API. Цей інтерфейс автоматично керує створенням фігур (Figure) та осей (Axes), дозволяючи швидко створювати графіки без глибокого розуміння внутрішньої архітектури. Фігура є контейнером верхнього рівня, який може містити один або кілька наборів осей, кожен з яких представляє окрему область для побудови графіків.



Таблиця 9.1 – Основні функції для створення графіків

Тип графіка	Функція	Опис	Основні параметри
Лінійний графік	plt.plot()	Побудова ліній та точок	x, y, color, linestyle, marker, linewidth
Стовпчаста діаграма	plt.bar(), plt.barh()	Вертикальні та горизонтальні стовпці	x, height, width, color, alpha
Точкова діаграма	plt.scatter()	Точки з різними розмірами та кольорами	x, y, s, c, marker, alpha
Гістограма	plt.hist()	Розподіл частот	x, bins, color, alpha, density
Кругова діаграма	plt.pie()	Секторна діаграма	x, labels, colors, autopct, explode
Коробкова діаграма	plt.boxplot()	Статистичний розподіл даних	x, vert, patch_artist, notch
Теплова карта	plt.imshow()	Матриця значень як зображення	X, cmap, aspect, interpolation
Контурна діаграма	plt.contour(), plt.contourf()	Лінії та заповнені області рівних значень	X, Y, Z, levels, colors, cmap

Таблиця 9.2 – Функції оформлення графіків

Категорія	Функція	Призначення	Приклад використання
Заголовки	plt.title()	Заголовок графіка	plt.title('Назва графіка', fontsize=14)
Підписи осей	plt.xlabel(), plt.ylabel()	Підписи осей координат	plt.xlabel('Вісь X'), plt.ylabel('Вісь Y')
Сітка	plt.grid()	Координатна сітка	plt.grid(True, alpha=0.3)
Легенда	plt.legend()	Пояснення до ліній/даних	plt.legend(['Лінія 1', 'Лінія 2'])
Межі осей	plt.xlim(), plt.ylim()	Діапазон відображення	plt.xlim(0, 10), plt.ylim(-5, 5)
Підграфіки	plt.subplot(), plt.subplots()	Кілька графіків в одній фігурі	plt.subplot(2, 1, 1)
Анотації	plt.annotate(), plt.text()	Текстові позначки	plt.text(x, y, 'текст')
Кольорова шкала	plt.colorbar()	Шкала кольорів для теплових карт	plt.colorbar(label='Значення')

Таблиця 9.3 – Додаткові типи графіків

Тип графіка	Функція	Застосування
Крокова діаграма	plt.step()	Ступінчасті зміни значень
Діаграма помилок	plt.errorbar()	Дані з показниками похибки
Заповнення області	plt.fill(), plt.fill_between()	Виділення областей під кривими
Стрімплот	plt.streamplot()	Векторні поля та потоки
Полярна діаграма	plt.polar()	Дані в полярних координатах
3D графіки	plt.plot3D(), plt.scatter3D()	Тривимірна візуалізація

Приклади створення графіків

Лінійний графік. Для створення лінійного графіка функції $y=x^2$ на інтервалі [-10,10] з кроком 0.5 використовується наступний підхід. Спочатку генеруються значення x за допомогою numpy.arange(), потім обчислюються відповідні значення y. Функція plt.plot() будує лінію з синім кольором, суцільним стилем та круглими маркерами.

```
import numpy as np
import matplotlib.pyplot as plt

# Створення даних
x = np.arange(-10, 10.1, 0.5)
y = x**2

# Побудова графіка
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='blue', linestyle='-', marker='o',
markersize=4)

# Оформлення графіка
plt.title('Графік параболи', fontsize=14)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
```

```
plt.grid(True, alpha=0.3)

# Відображення графіка
plt.tight_layout()
plt.show()
```

Стовпчаста діаграма. Стовпчаста діаграма створюється для категоріальних даних за допомогою функції plt.bar(). Категорії розміщуються на осі x, а їх значення визначають висоту стовпців. Колір, ширина стовпців та інші параметри налаштовуються через відповідні аргументи функції.

```
import matplotlib.pyplot as plt

# Створення даних
groups = ['А', 'Б', 'В']
values = [23, 19, 25]

# Побудова графіка
plt.figure(figsize=(8, 6))
plt.bar(groups, values, color='green', alpha=0.7)

# Оформлення графіка
plt.xlabel('Групи', fontsize=12)
plt.ylabel('Кількість', fontsize=12)
plt.title('Розподіл по групах', fontsize=14)

# Додавання значень на стовпці
for i, v in enumerate(values):
    plt.text(i, v + 0.5, str(v), ha='center', va='bottom')

# Відображення графіка
plt.tight_layout()
plt.show()
```

Кругова діаграма. Кругова діаграма ефективно показує співвідношення частин до цілого. Функція plt.pie() автоматично обчислює кути секторів на основі переданих значень. Параметр autopct дозволяє відображати відсотки, а explode виділяє окремі сектори.

```
import matplotlib.pyplot as plt

# Створення даних
labels = ['Їжа', 'Транспорт', 'Розваги', 'Інше']
sizes = [40, 20, 25, 15]
colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen']
explode = (0.1, 0, 0, 0)  # виділити перший сектор

# Побудова графіка
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
        explode=explode, shadow=True, startangle=90)

# Оформлення графіка
```

```
plt.title('Розподіл витрат', fontsize=14)
plt.axis('equal')  # рівні пропорції для круглої форми

# Відображення графіка
plt.tight_layout()
plt.show()
```

Гістограма. Гістограма відображає розподіл частот у наборі даних. Функція plt.hist() автоматично розбиває дані на інтервали (bins) та підраховує кількість значень у кожному інтервалі. Параметр bins визначає кількість стовпців гістограми.

```
import numpy as np
import matplotlib.pyplot as plt

# Створення даних – нормальний розподіл
np.random.seed(42)  # для відтворюваності результатів
data = np.random.normal(0, 1, 100)

# Побудова графіка
plt.figure(figsize=(8, 6))
plt.hist(data,       bins=15,       color='gray',       alpha=0.7,
edgecolor='black')

# Оформлення графіка
plt.xlabel('Значення', fontsize=12)
plt.ylabel('Частота', fontsize=12)
plt.title('Гістограма нормального розподілу', fontsize=14)
plt.grid(True, alpha=0.3)

# Додавання статистичної інформації
plt.axvline(np.mean(data), color='red', linestyle='--',
           label=f'Середнє: {np.mean(data):.2f}')
plt.legend()

# Відображення графіка
plt.tight_layout()
plt.show()
```

Точкова діаграма. Точкова діаграма відображає залежність між двома змінними за допомогою функції plt.scatter(). Кожна точка представляє одне спостереження з координатами (x, y). Різні маркери, кольори та розміри точок дозволяють кодувати додаткову інформацію.

```
import numpy as np
import matplotlib.pyplot as plt

# Створення випадкових даних
np.random.seed(42)
x = np.random.uniform(0, 10, 50)
y = np.random.uniform(0, 10, 50)

# Побудова графіка
```

```
plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='red', marker='x', s=50, alpha=0.7)

# Оформлення графіка
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.title('Точкова діаграма', fontsize=14)
plt.grid(True, alpha=0.3)

# Встановлення меж осей
plt.xlim(0, 10)
plt.ylim(0, 10)

# Відображення графіка
plt.tight_layout()
plt.show()
```

Горизонтальна стовпчаста діаграма. Горизонтальна стовпчаста діаграма ефективна для відображення категоріальних даних з довгими назвами. Функція plt.barh() створює горизонтальні стовпці, де довжина відповідає значенню.

```
import matplotlib.pyplot as plt

# Створення даних
days = ['Понеділок', 'Вівторок', 'Середа']
hours = [8, 6, 7]

# Побудова графіка
plt.figure(figsize=(8, 6))
plt.barh(days, hours, color='orange', alpha=0.7)

# Оформлення графіка
plt.xlabel('Години', fontsize=12)
plt.ylabel('День', fontsize=12)
plt.title('Робочі години по днях', fontsize=14)

# Додавання значень на стовпці
for i, v in enumerate(hours):
    plt.text(v + 0.1, i, str(v), va='center')

# Відображення графіка
plt.tight_layout()
plt.show()
```

Підграфіки (Subplots). Підграфіки дозволяють розміщувати кілька графіків в одній фігурі для порівняння різних наборів даних. Функція plt.subplot() створює сітку підграфіків, де кожен може мати власні налаштування та дані.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Створення даних
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Створення фігури з підграфіками
plt.figure(figsize=(10, 8))

# Перший підграфік
plt.subplot(2, 1, 1)
plt.plot(x, y1, 'b-', linewidth=2, label='sin(x)')
plt.title('Синус', fontsize=14)
plt.ylabel('y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend()

# Другий підграфік
plt.subplot(2, 1, 2)
plt.plot(x, y2, 'g-', linewidth=2, label='cos(x)')
plt.title('Косинус', fontsize=14)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend()

# Відображення графіків
plt.tight_layout()
plt.show()
```

2 КЛЮЧОВІ ПИТАННЯ

1. Яка основна мета бібліотеки matplotlib у Python, і чому вона є стандартом де-факто для візуалізації даних у наукових дослідженнях?
2. Які три рівні архітектури matplotlib ви знаєте, і яку функцію виконує кожен із них?
3. Що таке Figure і Axes у matplotlib, і як вони пов’язані між собою?
4. Які основні функції matplotlib.pyplot використовуються для побудови різних типів графіків (лінійний, стовпчаста, кругова, гістограма, точкова діаграма)? Наведіть приклади.
5. Як за допомогою numpy створити масив значень для побудови графіка функції, і чому це важливо?
6. Які функції використовуються для оформлення графіка (заголовок, підписи осей, сітка, легенда), і які параметри дозволяють налаштувати їх зовнішній вигляд?
7. Як створити кілька графіків у межах одного вікна за допомогою plt.subplot() або plt.subplots()? У яких випадках це доцільно?
8. Як відобразити відсотки на круговій діаграмі, виділити окремий сектор і додати легенду? Які параметри для цього використовуються?

9. Як правильно зберегти побудований графік у файл, і чому важливо використовувати plt.tight_layout() перед збереженням?

10. Чому в коді важливо використовувати коментарі, і що має містити аналітичний висновок після побудови графіка?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Розробіть програму на мові Python для візуалізації даних, використовуючи бібліотеку matplotlib відповідно до вашого індивідуального варіанта з таблиці варіантів. Програма повинна реалізувати повний цикл створення візуалізації від генерації даних до фінального оформлення графіка з дотриманням усіх технічних вимог та параметрів, зазначених у вашому варіанті. Завдання виконуйте по наступним крокам:

1. Підготуйте вихідні дані для візуалізації згідно з вашим варіантом. Якщо задана математична функція, використовуйте бібліотеку numpy для обчислення її значень на вказаному інтервалі з заданим кроком дискретизації. Для категоріальних чи статистичних даних створіть відповідні структури даних або скористайтеся генераторами випадкових чисел. Переконайтеся, що дані повністю відповідають специфікації варіанта та мають правильний формат для подальшої візуалізації.

2. Створіть графік заданого типу, використовуючи відповідну функцію matplotlib. Налаштуйте всі візуальні параметри згідно з варіантом: колір елементів, стиль та товщину ліній, тип та розмір маркерів. Особливу увагу приділіть точному дотриманню стилістичних вимог варіанта, оскільки вони є обов'язковими для виконання. Для складних графіків з кількома наборами даних забезпечте правильне відображення кожного елемента з унікальними візуальними характеристиками.

3. Оформіть графік з усіма необхідними елементами презентації даних. Додайте інформативний заголовок, що чітко відображає суть візуалізованих даних. Підпишіть осі координат з указанням одиниць вимірювання або назв категорій. За необхідності увімкніть координатну сітку для покращення читабельності графіка. Якщо на графіку відображено кілька наборів даних, розмістіть легенду у зручному місці. Усі текстові елементи мають бути зрозумілими та професійно оформленими.

4. Організуйте код програми логічними блоками з детальними коментарями, що пояснюють призначення кожного етапу створення візуалізації. Коментарі повинні розкривати особливості використаних функцій, параметрів налаштування та логіку обробки даних. Відобразіть готовий графік на екрані за допомогою plt.show() та за потреби збережіть його у файл використовуючи plt.savefig(). Завершіть роботу аналітичним висновком з описом характерних особливостей візуалізованих даних або поведінки досліджуваної функції.

Таблиця 9.4 – Варіанти індивідуальних завдань

№	Тип графіка	Дані / функція	Діапазон / крок	Колір	Стиль лінії / маркери	Додаткові вимоги
1	Лінійний графік	$y=x^2$	$x\in[-10,10]$, крок 0.5	синій	суцільна, маркери 'o'	Заголовок: "Графік параболи", сітка
2	Стовпчаста діаграма	Групи: А — 23, Б — 19, В — 25	—	зелений	—	Підпис осі X: "Групи", Y: "Кількість"
3	Кругова діаграма	Їжа — 40%, транспорт — 20%, розваги — 25%, інше — 15%	—	різні кольори	—	Відобразити відсотки, виноску для "Їжа"
4	Гістограма	100 чисел з нормального розподілу	$\mu=0,\sigma=1$	сірий	—	15 стовпців, сітка, заголовок
5	Точкова діаграма	x,y— випадкові (50 точок)	$x,y\in[0,10]$	червоний	маркери 'x'	Підписи осей, сітка
6	Лінійний графік	$y=\cos(2x)$	$[0,2\pi]$, крок 0.1	фіолетовий	суцільна	Сітка, підписи осей
7	Два графіки	$y=x, y=x^2$	[0,5], крок 0.2	синій, червоний	суцільні	Легенда, заголовок, сітка
8	Стовпчаста діаграма	Температура: [18, 20, 22, 19, 21]	Дні тижня	оранжевий	—	Осі: "День", "Температура (°C)"
9	Лінійний графік	$y=x$	$[-5,-0.1]\cup[0.1,5]$, крок 0.1	чорний	пунктирна	Два окремі графіки, сітка
10	Кругова діаграма	Відмінники — 10, добре — 15, задовільно — 5, незад. — 2	—	різні	—	Легенда, заголовок: "Успішність"
11	Лінійний графік	$y=e^{-x}\cdot\sin(x)$	$[0,4\pi]$, крок 0.1	зелений	суцільна, маркери '.'	Сітка, підписи осей
12	Гістограма	Вік 30 людей	[18,60]	блакитний	—	10 стовпців, заголовок, сітка
13	Точкова діаграма	Залежність ваги від висоти (40 пар)	висота: 150–200 см, вага: 50–100 кг	фіолетовий	маркери 'o'	Осі: "Висота", "Вага"
14	Стовпчаста діаграма	Дохід: [120, 135, 130, 150, 145, 160]	Місяці: січ—черв	коричневий	—	Осі: "Місяць", "Дохід (тис. грн)"
15	Лінійний графік	$y=\tan(x)$	$[-\pi,\pi]$, крок 0.01	червоний	суцільна	Виключити точки розриву, сітка
16	Два графіки	$y=\sin(x)$, $y=\cos(x)$	$[0,2\pi]$, крок 0.1	синій, зелений	суцільні	Легенда, заголовок,

						сітка
17	Кругова діаграма	Сон — 8, робота — 8, відпочинок — 4, інше — 4	—	різні	—	Відсотки, заголовок: "Розподіл часу"
18	Гістограма	Оцінки студентів (100 значень)	[1,12]	сірий	—	12 стовпців, сітка, підпис Y: "Частота"
19	Лінійний графік	$y=x^3-3x$	[−3,3], крок 0.1	оранжевий	штрихпунктирна	Сітка, підписи осей
20	Точкова діаграма	x=[1,2,3,4,5],y=[2,4,1,5,3]	—	чорний	маркери 's' (квадрати)	Заголовок, сітка
21	Стовпчаста діаграма	Книги: [7,5,9,6]	Імена: 4 друзів	зелений	—	Осі: "Ім'я", "Кількість"
22	Лінійний графік	$y=\sqrt{x}$	[0,10], крок 0.2	фіолетовий	суцільна	Сітка, підпис Y: "$y=\sqrt{x}$"
23	Кругова діаграма	Apple — 30%, Samsung — 35%, Xiaomi — 20%, інші — 15%	—	різні	—	Легенда, відсотки
24	Гістограма	Рівномірний розподіл	[0,10], 100 точок	блакитний	—	20 стовпців, заголовок, сітка
25	Лінійний графік	y=log(x)	[0.1,10], крок 0.1	чорний	суцільна	Підпис Y: "log(x)", сітка
26	Точкова діаграма	x,y— випадкові (60 точок)	[0,10]	за кольоровою мапою	маркери 'o'	Колір залежить від значення x
27	Стовпчаста діаграма	Значення: [-5, 3, -2, 7, -1]	Категорії: A–E	різні	—	Від’ємні стовпці вниз, підписи осей
28	Лінійний графік	y=|x|	[−5,5], крок 0.2	червоний	суцільна	Сітка, заголовок: "Модуль x"
29	Subplots	y=sin(x), y=cos(x)	[0,2π], крок 0.1	синій, зелений	суцільні	Два графіки у стовпці, заголовки
30	Горизонтальна діаграма	Години: [8,6,7]	Дні: Пн, Вт, Ср	оранжевий	—	Осі: "Години", "День", горизонтальні стовпці

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов’язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент розробляє програму, призначену для візуалізації даних у вигляді графіка згідно з індивідуальним варіантом завдання. Програма передбачає генерацію або завантаження вхідних даних, побудову графіка заданого типу (лінійний, стовпчастий, круговий тощо), його оформлення (заголовок, підписи осей, легенда, сітка) та відображення або збереження результату. Особливу увагу приділяється коректному використанню функцій бібліотеки matplotlib, читабельності коду та дотриманню вимог щодо візуального оформлення графіка.

У протоколі обов’язково мають бути наведені три ключові елементи:

– по-перше, завдання, у якому чітко вказано індивідуальний варіант (тип графіка, дані або функція, параметри візуалізації);

– по-друге, лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють етапи генерації даних, побудови та оформлення графіка;

– по-третє, результати виконання програми, представлені у вигляді скріншота побудованого графіка або збереженого зображення, що підтверджує відповідність візуалізації вимогам завдання.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 10

Тема: Мережеве програмування з використанням сокетів у Python

Мета роботи: Навчитися створювати TCP-сервери та клієнти за допомогою бібліотеки socket у Python, обмінюватися даними з використанням простого текстового протоколу, а також тестувати взаємодію за допомогою PuTTY у Raw-режимі.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Мережеве програмування з використанням сокетів є фундаментальною основою для створення розподілених додатків та систем клієнт-сервер. Бібліотека socket у Python є частиною стандартної бібліотеки мови та надає програмісту можливість працювати безпосередньо з мережевими з'єднаннями на рівні транспортного шару моделі OSI, забезпечуючи повний контроль над процесом передачі даних між процесами.

Сокет представляє собою програмний інтерфейс для мережевого з'єднання між двома процесами, які можуть виконуватися як на одному комп'ютері, так і на різних машинах у мережі. Концепція сокетів була розроблена в операційній системі BSD Unix у 1980-х роках та з тих пір стала стандартом для мережевого програмування. В Python бібліотека socket забезпечує об'єктно-орієнтований інтерфейс для роботи із системними викликами сокетів.

Бібліотека socket підтримує різні типи сокетів та протоколів. Найбільш поширеними є TCP-сокети, які використовують протокол надійної передачі даних з контролем потоку та автоматичним виправленням помилок, та UDP-сокети для швидкої передачі даних без гарантій доставки. TCP-сокети забезпечують надійну, упорядковану доставку даних з автоматичним контролем помилок та повторною передачею втрачених пакетів, що робить їх ідеальними для застосувань, де важлива цілісність даних.

Робота з сокетами в Python базується на використанні констант сімейства адрес та типів сокетів. Константа socket.AF_INET вказує на використання протоколу IPv4, тоді як socket.AF_INET6 призначена для IPv6. Тип сокета визначається константами socket.SOCK_STREAM для TCP-з'єднань та socket.SOCK_DGRAM для UDP-протоколу.

Процес створення TCP-сервера включає декілька послідовних етапів, кожен з яких має своє призначення. Спочатку створюється об'єкт сокета за допомогою функції socket.socket() з параметрами, що вказують на використання протоколу IPv4 та TCP. Наступним кроком сокет прив'язується до конкретної адреси та порту методом bind(), що резервує цей порт для використання сервером. Після прив'язки сокет переводиться в режим очікування підключень за допомогою методу listen(), який вказує максимальну кількість клієнтів у черзі підключення. Сервер приймає вхідні з'єднання методом accept(), який

блокує виконання програми до надходження запиту на підключення та повертає новий сокет для роботи з конкретним клієнтом разом з його адресою.

Створення TCP-клієнта є простішим процесом порівняно з сервером. Клієнт також починає зі створення сокета за допомогою socket.socket(), але замість прив'язки до порту використовує метод connect() для встановлення з'єднання з віддаленим сервером. Після успішного з'єднання клієнт може відправляти дані методом send() та отримувати відповіді за допомогою recv().

Важливим аспектом роботи з сокетами є правильне управління ресурсами та обробка помилок. Сокети споживають системні ресурси, тому їх необхідно коректно закривати після завершення роботи. Мережеві з'єднання можуть перериватися з різних причин, включаючи проблеми з мережею, відключення клієнта або помилки в програмі, тому код повинен містити відповідну обробку виключень.

Таблиця 10.1 – Основні функції модуля Socket

Функція	Опис	Приклад використання
socket.socket(family, type)	Створює новий сокет з вказаним сімейством адрес та типом	s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
bind(address)	Прив'язує сокет до локальної адреси та порту	s.bind(('localhost', 8080))
listen(backlog)	Переводить сокет в режим очікування підключень	s.listen(5)
accept()	Приймає вхідне з'єднання, повертає новий сокет та адресу клієнта	client_socket, addr = s.accept()
connect(address)	Встановлює з'єднання з віддаленим сервером (для клієнта)	s.connect(('localhost', 8080))
send(data)	Відправляє дані через сокет, повертає кількість відправлених байтів	s.send(b'Hello World')
recv(bufsize)	Отримує дані з сокета, максимум bufsize байтів	data = s.recv(1024)
close()	Закриває сокет та звільняє ресурси	s.close()
setsockopt(level, option, value)	Встановлює опції сокета	s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

Робота з сокетами вимагає ретельної обробки помилок, оскільки мережеві з'єднання можуть перериватися з різних причин. Важливо правильно закривати сокети після завершення роботи для запобігання витоку ресурсів системи.

Приклад echo-сервера зі зміною регістра символів

У цьому лабораторному завданні розглядається приклад реалізації простого TCP echo-сервера мовою Python із використанням модуля socket. Сервер очікує підключення клієнта, приймає текстові повідомлення та надсилає відповідь, змінюючи регістр символів отриманого рядка: рядки в нижньому регістрі перетворюються у верхній, у верхньому — в нижній, а у змішаному регістрі – інвертуються. Приклад демонструє базові принципи мережевого

програмування, зокрема створення сокета, прив’язку до адреси та порту, роботу в режимі очікування підключень, обмін даними між сервером і клієнтом, а також коректне завершення з’єднання. На основі наведеного коду студентам необхідно проаналізувати логіку роботи сервера та використати її під час виконання практичного завдання.

```
import socket

def main():
    # Створення TCP сокета
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

    # Дозвіл повторного використання адреси
    server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)

    # Прив'язка до локальної адреси та порту
    host = 'localhost'
    port = 12345
    server_socket.bind((host, port))

    # Перехід в режим очікування підключень
    server_socket.listen(1)
    print(f"Сервер запущено на {host}:{port}")
    print("Очікування підключення...")

    try:
        while True:
            # Прийняття нового клієнта
            client_socket, client_address = server_socket.accept()
            print(f"Підключення від {client_address}")

            try:
                while True:
                    # Отримання даних від клієнта
                    data = client_socket.recv(1024)

                    if not data:
                        break

                    # Декодування отриманих даних
                    message = data.decode('utf-8').strip()
                    print(f"Отримано: {message}")

                    # Зміна регістра символів
                    if message.islower():
                        response = message.upper()
                    elif message.isupper():
                        response = message.lower()
                    else:
                        response = message.swapcase()
```

```
                    # Відправка відповіді клієнту
                    client_socket.send((response                    +
'\n').encode('utf-8'))
                    print(f"Відправлено: {response}")

            except Exception as e:
                print(f"Помилка: {e}")
            finally:
                client_socket.close()
                print("З'єднання з клієнтом закрито")

    except KeyboardInterrupt:
        print("\nЗавершення роботи сервера...")
    finally:
        server_socket.close()

if __name__ == "__main__":
    main()
```

2 КЛЮЧОВІ ПИТАННЯ

1. Яке призначення бібліотеки socket у Python
2. Яка різниця між TCP та UDP сокетами
3. З яких етапів складається процес створення TCP сервера
4. Які основні дії необхідні для створення TCP клієнта
5. Яку роль відіграють методи bind listen та accept при роботі сервера
6. Яке призначення методів connect send та recv при роботі клієнта
7. Чому важливо використовувати обробку винятків та правильно закривати сокети
8. Як налаштувати PuTTY для підключення до розробленого TCP сервера у Raw режимі
9. Як реалізувати специфічну логіку обробки отриманих від клієнта даних згідно з варіантом завдання
10. Як можна модифікувати сервер щоб він міг обслуговувати кількох клієнтів одночасно

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Розробити на мові Python TCP-сервер, який працює на вказаному порті, очікує підключення клієнта через PuTTY у режимі Raw, приймає від клієнта текстове повідомлення, обробляє отримані дані згідно з умовою варіанту (), надсилає відповідь клієнту та залишається активним для обслуговування наступних клієнтів.

Сервер має використовувати виключно стандартну бібліотеку socket без використання високорівневих фреймворків типу Flask або asyncio. Код повинен коректно обробляти закриття з'єднання клієнтом та містити обробку помилок. Програма має бути читабельною з відповідними коментарями та забезпечувати

можливість одночасного обслуговування декількох клієнтів за допомогою потоків або процесів.

Основні кроки, необхідні для виконання завдання наведено в таблиці нижче:

– проаналізувати варіант завдання та визначити логіку обробки даних, яку має реалізовувати сервер;

– створити основну структуру TCP-сервера з використанням модуля socket, включаючи створення сокета, прив'язку до порту та очікування підключень;

– реалізувати функцію обробки клієнтських запитів відповідно до умов варіанту завдання з додаванням обробки помилок для коректної роботи з несправними з'єднаннями та некоректними даними від клієнта;

– запустити розроблений сервер на локальному комп'ютері та переконатися, що він успішно прослуховує вказаний порт;

– підключитися до сервера за допомогою PuTTY у Raw-режимі, налаштувавши відповідні параметри з'єднання;

– протестувати роботу сервера, відправивши різні типи повідомлень та перевіривши правильність отриманих відповідей згідно з логікою варіанту;

– оформити звіт з роботи, включаючи код програми, скріншоти тестування та опис реалізованої функціональності.

Для створення підключення через PuTTY виконайте наступні дії:

– запустіть програму PuTTY на комп'ютері, де має працювати клієнтська частина;

– у головному вікні PuTTY у полі "Host Name (or IP address)" введіть localhost якщо сервер працює на тому ж комп'ютері, або IP-адресу віддаленого сервера;

– у полі "Port" вкажіть номер порту, на якому працює ваш TCP-сервер (наприклад, 12345 або згідно з варіантом завдання);

– у розділі "Connection type" оберіть варіант "Raw", що забезпечить передачу даних без додаткових протоколів верхнього рівня;

– натисніть кнопку "Open" для встановлення з'єднання з сервером;

– у відкритому вікні терміналу введіть тестове повідомлення відповідно до логіки вашого варіанту та натисніть клавішу Enter для відправки даних на сервер.

Таблиця 10.2 – Варіанти індивідуальних завдань

№	Опис завдання
1	Сервер приймає число та повертає його квадрат. При отриманні нечислового значення повертає повідомлення про помилку.
2	Сервер приймає рядок та повертає його у зворотному порядку. Якщо рядок порожній, повертає повідомлення "Порожній рядок".
3	Сервер приймає два числа через пробіл та повертає їх суму. При некоректному вводі повертає повідомлення про помилку формату.
4	Сервер приймає рядок та рахує кількість голосних літер (a, e, i, o, u, а, е, і, о, у). Повертає результат підрахунку.

5	Сервер приймає число та перевіряє, чи є воно простим. Повертає "Просте" або "Складне" відповідно.
6	Сервер приймає рядок та повертає його, замінивши всі цифри на символ '*'.
7	Сервер приймає температуру в Цельсіях та повертає її значення у Фаренгейтах з формулою F = C * 9/5 + 32.
8	Сервер приймає список чисел через кому та повертає їх середнє арифметичне з точністю до двох знаків після коми.
9	Сервер приймає рядок та повертає кількість слів у ньому. Слова розділяються пробілами.
10	Сервер приймає число та повертає усі його дільники, розділені комами.
11	Сервер приймає рядок та повертає його, перетворивши перші літери кожного слова на великі.
12	Сервер приймає число та повертає його факторіал. Для від'ємних чисел повертає повідомлення про помилку.
13	Сервер приймає два рядки через символ '
14	Сервер приймає число та повертає його у двійковому представленні.
15	Сервер приймає рядок та рахує кількість входжень кожної літери, повертаючи результат у вигляді "літера:кількість".
16	Сервер приймає список чисел через пробіл та повертає максимальне та мінімальне значення через дефіс.
17	Сервер приймає рядок та перевіряє, чи є він паліндромом. Повертає "Так" або "Ні".
18	Сервер приймає число секунд та конвертує їх у формат "години:хвилини:секунди".
19	Сервер приймає рядок та повертає його, видаливши всі пробіли та розділові знаки.
20	Сервер приймає два числа через пробіл та повертає їх найбільший спільний дільник (НСД).
21	Сервер приймає рядок та повертає його, кодуючи за допомогою шифру Цезаря зі зсувом 3.
22	Сервер приймає список чисел через кому та повертає їх відсортованими за зростанням.
23	Сервер приймає рядок та рахує кількість великих та малих літер окремо, повертаючи результат через дефіс.
24	Сервер приймає число та повертає суму його цифр.
25	Сервер приймає два рядки через символ '#' та повертає їх конкатенацію з пробілом між ними.
26	Сервер приймає число та перевіряє, чи є воно числом Фібоначчі. Повертає "Так" або "Ні".
27	Сервер приймає рядок та повертає його, замінивши всі малі літери на великі та навпаки.
28	Сервер приймає список слів через кому та повертає найдовше слово.
29	Сервер приймає число та повертає його римське представлення (для чисел від 1 до 3999).
30	Сервер приймає математичний вираз виду "число операція число" та повертає результат обчислення.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов’язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім’я та по батькові студента, назва академічної

групи, а також прізвище, ім’я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент розробляє TCP-сервер на мові Python за допомогою стандартного модуля socket. Програма реалізує обробку текстових повідомлень, отриманих від клієнта через мережеве з’єднання, виконує індивідуальну логіку обробки даних згідно з варіантом завдання та надсилає відповідь клієнту. Особливу увагу приділяється коректному створенню та налаштуванню сокета, обробці помилок, безпечному закриттю з’єднання, а також можливості обслуговування декількох клієнтів (за потреби – через потоки).

У протоколі обов’язково мають бути наведені три ключові елементи:

– завдання, у якому чітко описано індивідуальний варіант (логіка обробки вхідних даних, формат відповіді, очікувана поведінка сервера);

– лістинг програми, що містить повний, добре структурований код TCP-сервера на мові Python з необхідними коментарями, які пояснюють етапи створення сокета, прийняття з’єднання, обробки даних та відправки відповіді;

– результати виконання програми, представлені у вигляді скріншотів консолі сервера (з виводом отриманих і надісланих повідомлень) та вікна клієнтського інструменту (наприклад, PuTTY у Raw-режимі), що підтверджують коректну роботу сервера при різних сценаріях введення.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 11

Тема: Розробка та тестування HTTP веб-сервера на мові Python з використанням модуля socket

Мета роботи: Отримати практичні навички створення простого HTTP веб-сервера на Python з використанням сокетів, обробки запитів та формування відповідей.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

HTTP протокол та його основи

HTTP (HyperText Transfer Protocol) є фундаментальним протоколом передачі даних у всесвітній мережі Інтернет, який визначає правила взаємодії між клієнтами та серверами в архітектурі всесвітньої павутини. Цей протокол працює за принципом запит-відповідь, де клієнт завжди ініціює комунікацію, надсилаючи HTTP запит до сервера з вказівкою бажаного ресурсу, методу обробки та додаткових параметрів. Сервер отримує запит, аналізує його зміст, здійснює необхідні операції з ресурсами та формує відповідну HTTP відповідь, яка містить статус виконання операції та запитувані дані. HTTP характеризується відсутністю стану між запитами, що означає неможливість сервера автоматично зберігати інформацію про попередні взаємодії з конкретним клієнтом. Кожен запит обробляється як незалежна операція, що спрощує логіку серверних додатків та підвищує їх надійність і масштабованість. Протокол функціонує поверх надійного транспортного протоколу TCP, який забезпечує гарантовану доставку даних та їх правильну послідовність.

Структура HTTP повідомлень має суворо визначений формат, який забезпечує стандартизовану взаємодію між різними програмними системами незалежно від їх внутрішньої реалізації. HTTP запит складається з трьох основних компонентів: стартового рядка, набору заголовків та опціонального тіла повідомлення, кожен з яких виконує специфічну роль у передачі інформації. Стартовий рядок містить метод запиту (GET, POST, PUT, DELETE тощо), повний URL запитуваного ресурсу та версію HTTP протоколу, всі компоненти розділені одинарними пробілами. Заголовки надають розширену метаінформацію про запит, включаючи тип очікуваного контенту, розмір повідомлення, інформацію про клієнтське програмне забезпечення, параметри автентифікації та налаштування кешування. Тіло повідомлення використовується для передачі корисних даних у випадках, коли це необхідно, наприклад при відправці форм, завантаженні файлів або передачі JSON даних у API запитах.

HTTP відповідь має аналогічну структуру з тією відмінністю, що замість стартового рядка запиту використовується статусний рядок, який інформує клієнта про результат обробки його запиту. Статусний рядок містить версію

HTTP протоколу, тризначний числовий код статусу та коротку текстову фразу, що описує значення цього коду зрозумілою для людини мовою. Коди статусу класифікуються на п'ять основних категорій (табл. 1): інформаційні коди серії 100 використовуються для проміжних повідомлень про стан обробки, коди серії 200 сигналізують про успішне виконання запиту, серія 300 вказує на необхідність перенаправлення або додаткових дій, коди 400 означають помилки на стороні клієнта, а серія 500 повідомляє про внутрішні помилки сервера. Заголовки відповіді містять метаінформацію про повернуті дані, включаючи тип контенту, розмір, параметри кешування та додаткові інструкції для клієнта щодо обробки отриманої інформації.

Таблиця 11.1 - Класифікація HTTP статус кодів

Діапазон	Категорія	Опис
100-199	Інформаційні	Проміжні відповіді про стан обробки запиту
200-299	Успішні	Запит був успішно отриманий та оброблений
300-399	Перенаправлення	Потрібні додаткові дії для завершення запиту
400-499	Помилки клієнта	Запит містить синтаксичні помилки або не може бути оброблений
500-599	Помилки сервера	Сервер не зміг обробити коректний запит

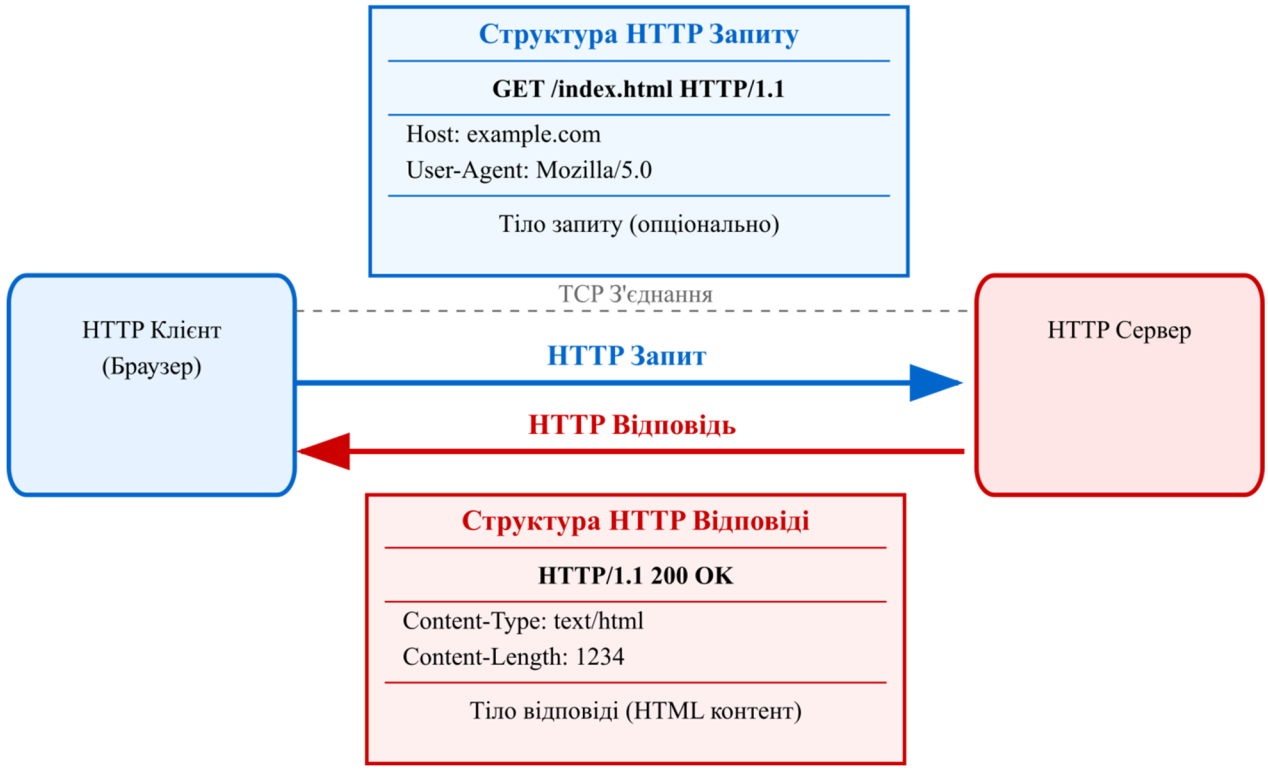


Рисунок 11.1 – Процедура HTTP обміну між клієнтом та сервером з структурою пакетів

Діаграма демонструє повний цикл HTTP комунікації, починаючи з лівого боку, де зображено блок "HTTP Клієнт (Браузер)" у вигляді прямокутника з округленими кутами. З цього блоку виходить горизонтальна стрілка вправо з підписом "HTTP Запит", яка веде до правого блоку "HTTP Сервер", також оформленого як прямокутник з округленими кутами. Під стрілкою запиту

розміщено детальну структуру HTTP запиту у вигляді вертикального блоку, що містить три секції: верхня секція показує стартовий рядок "GET /index.html HTTP/1.1", середня секція містить заголовки "Host: example.com" та "User-Agent: Mozilla/5.0", нижня секція підписана як "Тіло запиту (опціонально)". З серверного блоку виходить зворотна стрілка вниз і вліво з підписом "HTTP Відповідь", під якою розташована структура відповіді: верхня частина містить статусний рядок "HTTP/1.1 200 OK", середня частина показує заголовки "Content-Type: text/html" та "Content-Length: 1234", нижня частина підписана як "Тіло відповіді (HTML контент)". Між клієнтом та сервером також показано пунктирну лінію з підписом "TCP З'єднання", що ілюструє транспортний рівень комунікації.

Методи HTTP запитів визначають тип операції, яку клієнт бажає виконати з вказаним ресурсом на сервері. Метод GET є найпоширенішим та використовується для отримання інформації без внесення змін до стану сервера, при цьому всі параметри передаються через URL у вигляді query string. Метод POST призначений для відправки даних на сервер з метою створення нових ресурсів або обробки інформації, при цьому дані передаються у тілі запиту і не відображаються в URL. Методи PUT та DELETE використовуються для модифікації існуючих ресурсів та їх видалення відповідно, хоча не всі сервери підтримують ці операції. Метод HEAD аналогічний до GET, але повертає тільки заголовки без тіла відповіді, що корисно для перевірки існування ресурсу або отримання метаінформації. Кожен метод має свою семантику та очікувану поведінку, що дозволяє створювати консистентні та передбачувані веб-інтерфейси.

Архітектура веб-сервера та обробка запитів згідно наданого Python скрипта

Архітектура досліджуваного Python веб-сервера базується на простій синхронній моделі обробки запитів, яка реалізована через послідовний цикл прийняття з'єднань та їх обробки за допомогою стандартного модуля socket. Основний цикл сервера працює у нескінченному режимі, очікуючи вхідні TCP з'єднання на порту 8080 та обробляючи кожен запит по черзі до його повного завершення. Ця архітектура, хоча і має обмеження щодо продуктивності через неможливість одночасної обробки декількох запитів, є достатньою для навчальних цілей та демонстрації основних принципів роботи HTTP серверів. Сервер використовує файлову систему як сховище статичного контенту, де всі веб-ресурси розміщуються у спеціальній директорії 'www/', що забезпечує логічне розділення між кодом сервера та контентом сайту. Налаштування сокета включає встановлення опції SO_REUSEADDR, яка дозволяє швидко перезапускати сервер без очікування повного закриття попередніх з'єднань операційною системою. Прослуховування здійснюється з параметром backlog равным 5, що означає можливість накопичення до п'яти очікуючих з'єднань у системній черзі до моменту їх прийняття серверним додатком.

Таблиця 11.2 – Основні файли каталогу www

№	Назва файлу	Тип файлу	Призначення	Обов'язковий	MIME-тип
1	index.html	HTML	Головна сторінка сайту, яка завантажується за замовчуванням при запиті кореневого URL ("/")	Так	text/html
2	404.html	HTML	Сторінка помилки "Не знайдено", яка відображається при запиті неіснуючих файлів	Так	text/html
3	favicon.ico	ICO	Іконка сайту, що відображається у вкладці браузера та закладках	Так	image/x-icon
4	about.html	HTML	Сторінка "Про нас" або інформаційна сторінка	Ні	text/html
5	contact.html	HTML	Сторінка контактів	Ні	text/html
6	services.html	HTML	Сторінка послуг	Ні	text/html
7	gallery.html	HTML	Сторінка галереї	Ні	text/html
8	logo.jpg	JPEG	Логотип або зображення для сайту	Ні	image/jpeg
9	banner.jpg	JPEG	Банер або заголовне зображення	Ні	image/jpeg
10	photo1.jpg	JPEG	Фотографія для галереї	Ні	image/jpeg

Веб-сервер автоматично обслуговує три основні категорії файлів. HTML файли з розширеннями .html та .htm обробляються функцією send_HTTP_text(), яка відправляє текстовий контент з правильним MIME-типом text/html. При запиті кореневого URL "/" сервер автоматично перенаправляє на файл index.html, який виступає головною сторінкою сайту.

JPEG зображення з розширенням .jpg обслуговуються через функцію send_HTTP_image(), що використовує MIME-тип image/jpeg та передає файли у бінарному режимі. Це дозволяє коректно відображати графічний контент у веб-браузерах. Особливо важливим є правильне встановлення типу контенту для забезпечення сумісності з різними браузерами.

Файл favicon.ico має спеціальну обробку через окрему функцію send_HTTP_icon() з MIME-типом image/x-icon. Цей файл автоматично запитується браузерами для відображення іконки сайту у вкладках та закладках. Обов'язковою умовою роботи сервера є наявність файлу 404.html, який відображається при запиті неіснуючих ресурсів з HTTP статусом 404.

Структура каталогу www повинна містити мінімальний набір файлів: index.html як головну сторінку, 404.html для обробки помилок та favicon.ico для іконки сайту. Додаткові HTML сторінки та JPEG зображення можуть додаватися відповідно до потреб проекту, але всі вони будуть автоматично обслуговуватися сервером згідно з встановленими правилами маршрутизації.

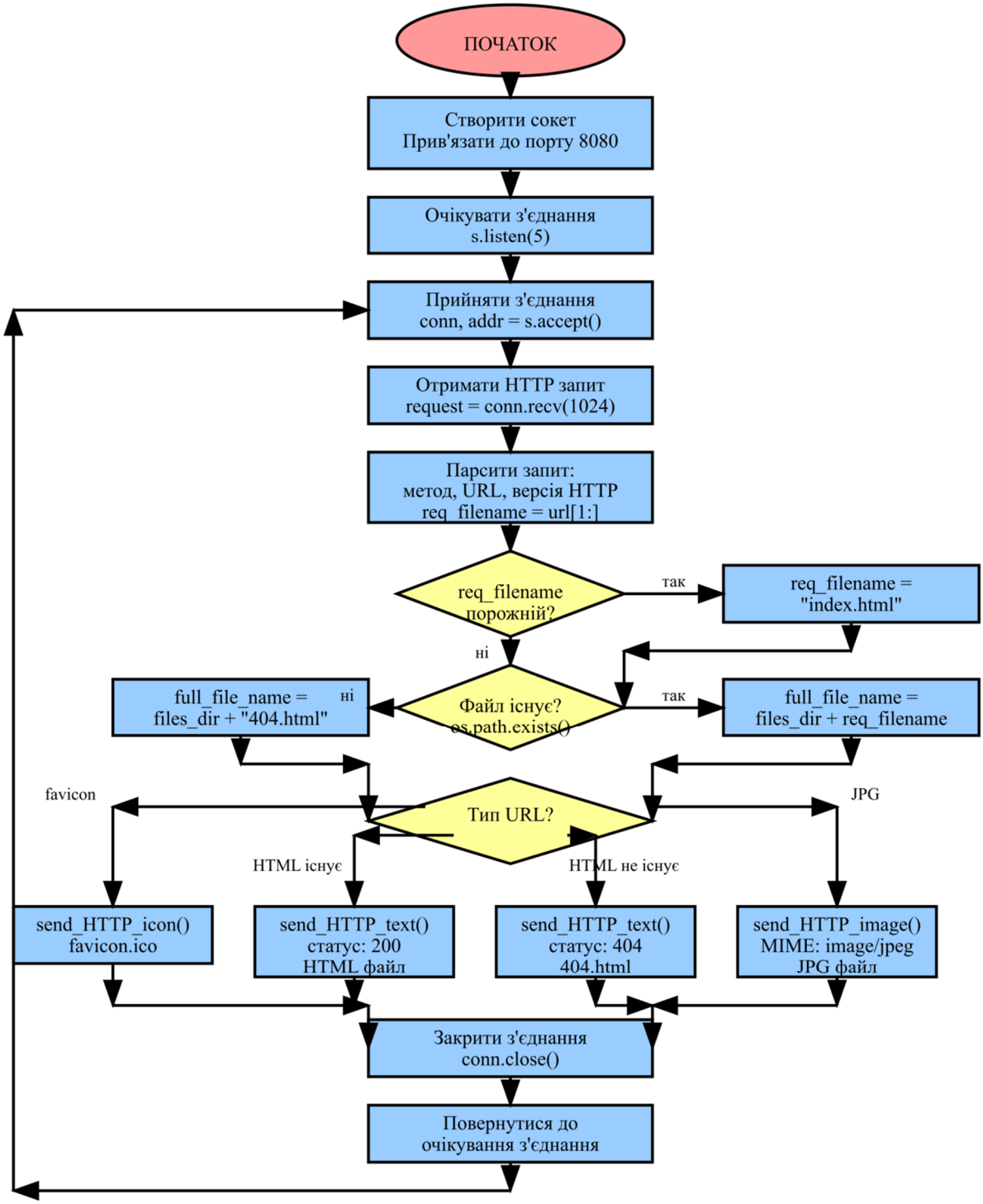


Рисунок 11.2 – Блок-схема алгоритму веб-сервера

Процес обробки кожного клієнтського запиту починається з прийняття TCP з'єднання методом accept(), який блокує виконання програми до появи нового клієнта. Після встановлення з'єднання сервер читає HTTP запит за допомогою методу recv() з буфером розміром 1024 байти, що є достатнім для більшості стандартних HTTP заголовків. Отримані дані передаються функції req_parser(), яка здійснює комплексний аналіз структури запиту та визначає подальші дії сервера. Парсинг починається з декодування байтового потоку у UTF-8 рядок та його розбиття по символах переносу рядка для виділення

окремих компонентів HTTP повідомлення. Стартовий рядок аналізується для витягування HTTP методу, URL ресурсу та версії протоколу, при цьому сервер виводить запитуваний URL у консоль для діагностичних цілей. Обробка URL включає видалення початкового слешу та встановлення значення за замовчуванням 'index.html' для запитів до кореневої директорії сайту.

Система визначення типу запитуваного ресурсу базується на аналізі розширення файлу та спеціальній обробці окремих випадків, що дозволяє серверу коректно обслуговувати різні типи контенту. Першочергово здійснюється перевірка існування файлу за допомогою функції os.path.exists(), і у випадку відсутності запитуваного ресурсу сервер автоматично перенаправляє обробку на файл '404.html'. Спеціальна логіка реалізована для обробки запитів favicon.ico, які автоматично генеруються більшістю браузерів та потребують повернення іконки сайту з правильним MIME типом 'image/x-icon'. HTML файли ідентифікуються за наявністю підрядка 'htm' у назві файлу та обробляються функцією send_HTTP_text() з відповідним статус кодом 200 для існуючих файлів або 404 для відсутніх. JPEG зображення розпізнаються за присутністю підрядка 'jpg' у назві файлу та передаються через функцію send_HTTP_image() з MIME типом 'image/jpeg'. Після завершення обробки кожного запиту з'єднання закривається методом close(), і сервер виводить повідомлення про закриття з'єднання для моніторингу активності.

Функція send_HTTP_text() реалізує передачу текстового контенту з формуванням відповідних HTTP заголовків та обробкою різних статус кодів. Процес починається з створення статусного рядка HTTP відповіді, який включає версію протоколу HTTP/1.1, переданий статус код та стандартну фразу 'OK'. Статусний рядок виводиться у консоль для діагностики та передається клієнту у закодованому вигляді UTF-8. Наступним кроком є встановлення заголовка 'Content-Type: text/html; charset=utf-8', який інформує браузер про тип контенту та кодування символів для правильного відображення. Обов'язковий порожній рядок, що відділяє заголовки від тіла повідомлення, передається як подвійна послідовність символів переносу рядка '\r\n\r\n'. Передача файлового контенту здійснюється построково через читання файлу у текстовому режимі та послідовну відправку кожного рядка клієнту у закодованому UTF-8 форматі. Цей підхід дозволяє ефективно обробляти файли різного розміру без завантаження всього контенту в пам'ять одночасно, хоча для великих файлів може бути доцільним використання буферизованого читання.

Функція send_HTTP_image() призначена для обробки бінарних файлів зображень з урахуванням специфіки їх структури та вимог до передачі. Формування HTTP відповіді починається аналогічно до текстових файлів зі створення статусного рядка '200 OK' та його виведення у консоль і передачі клієнту. Ключовою відмінністю є встановлення відповідного MIME типу через параметр content_type, який передається у функцію та дозволяє коректно обробляти різні формати зображень. Після відправки всіх необхідних заголовків файл відкривається у бінарному режимі 'rb', що критично важливо для збереження цілісності двійкових даних зображення. Весь вміст файлу читається в пам'ять за одну операцію методом read() та передається клієнту як

єдиний блок даних без додаткового кодування або перетворень. Такий підхід забезпечує точну передачу бінарної структури файлу, але може створювати навантаження на пам'ять сервера при обробці великих зображень.

Спеціалізована функція send_HTTP_icon() реалізує оптимізовану обробку favicon запитів з жорстко заданими параметрами для максимальної сумісності з різними браузерами. Функція формує стандартну HTTP відповідь зі статусом '200 OK' та встановлює специфічний MIME тип 'image/x-icon', який є загальноприйнятим стандартом для іконок веб-сайтів. Шлях до файлу іконки жорстко закодований як 'www/favicon.ico', що відповідає конвенції розміщення favicon у кореневій директорії сайту. Обробка файлу здійснюється у бінарному режимі аналогічно до інших зображень, забезпечуючи точну передачу структури ICO файлу браузеру. Ця функція демонструє принцип спеціалізації в архітектурі веб-сервера, де певні типи запитів отримують оптимізовану обробку для покращення продуктивності та надійності.

Діагностичні можливості сервера включають детальне логування у консоль всіх етапів обробки запитів, що дозволяє в реальному часі відстежувати активність та виявляти потенційні проблеми. Кожне нове з'єднання супроводжується виведенням IP адреси клієнта, що корисно для моніторингу джерел трафіку та виявлення підозрілої активності. Повний текст HTTP запиту виводиться у консоль між маркерами початку та кінця, надаючи можливість аналізувати структуру запитів та діагностувати проблеми парсингу. Інформація про запитуваний URL, визначений файл та шлях до нього виводиться окремими повідомленнями, дозволяючи простежити логіку обробки кожного запиту. Повідомлення про закриття з'єднання маркує завершення обробки запиту та готовність сервера до прийняття наступного клієнта, забезпечуючи візуальний контроль над циклом роботи сервера.

2 КЛЮЧОВІ ПИТАННЯ

1. Який принцип роботи HTTP протоколу та яка різниця між запитом та відповіддю?
2. Як створити та налаштувати TCP сокет у Python для прослуховування підключень?
3. Яка структура HTTP запиту та як його розпарсити у Python?
4. Що таке MIME типи та чому важливо встановлювати правильний Content-Type?
5. Як правильно формувати HTTP відповідь з заголовками та контентом?
6. Яка різниця між обробкою текстових та бінарних файлів у веб-сервері?
7. Що таке favicon та чому він потребує спеціальної обробки?
8. Як правильно закривати з'єднання після обробки запиту?
9. Які основні кроки тестування веб-сервера та на що звертати увагу?
10. Як реалізувати механізм обробки помилок (наприклад, 404 Not Found) у веб-сервері та чому важливо повертати коректні коди стану HTTP?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Студентам надається готовий скрипт Python веб-сервера, який реалізує базову функціональність HTTP сервера з підтримкою статичних файлів. Основне завдання полягає у створенні власного веб-контенту, розміщенні його у відповідній структурі каталогів та всебічному тестуванні роботи сервера з документуванням результатів.

Першим етапом роботи є створення каталогу www у тій же директорії, де знаходиться скрипт веб-сервера. Цей каталог буде служити кореневою директорією для всіх статичних файлів, які сервер буде обслуговувати. Структура каталогу повинна бути логічно організованою та містити різні типи файлів для всебічного тестування функціональності сервера.

У каталозі www необхідно створити файл index.html, який буде головною сторінкою веб-сайту. Ця сторінка повинна містити повноцінний HTML код з використанням різних тегів, посилань на зображення та стилізації. Рекомендується включити заголовки, абзаци тексту, списки, таблиці та інші HTML елементи для демонстрації можливостей сервера по обслуговуванню HTML контенту.

Окрім головної сторінки, потрібно створити файл 404.html, який буде відображатися у випадку запиту неіснуючих ресурсів. Ця сторінка повинна містити інформативне повідомлення про помилку та можливо посилання для повернення на головну сторінку. Дизайн сторінки помилки може відрізнятися від основної сторінки, але повинен підтримувати загальний стиль сайту.

До структури сайту необхідно додати кілька зображень у форматі JPEG для тестування обробки бінарних файлів. Зображення можуть бути різного розміру та змісту, але повинні бути оптимізованими для веб-використання. Рекомендується включити ці зображення у HTML сторінки за допомогою тегу img для перевірки їх правильного відображення у браузері.

Особливу увагу слід приділити файлу favicon.ico, який повинен бути розміщений у кореневому каталозі www. Цей файл буде автоматично запитуватися браузерами та потребує спеціальної обробки сервером. Іконка повинна бути у правильному форматі ICO та мати стандартні розміри для коректного відображення у браузерах.

Після підготовки всіх файлів необхідно запустити веб-сервер та провести всебічне тестування його роботи. Тестування повинно включати запити до існуючих ресурсів, спроби доступу до неіснуючих файлів, перевірку завантаження зображень та аналіз роботи з різними браузерами. Особливу увагу слід приділити аналізу виводу сервера у консолі та поведінці браузера при різних типах запитів.

Протягом тестування необхідно зробити скріншоти текстового виводу у консолі, де запущено сервер, щоб продемонструвати процес обробки HTTP запитів. Скріншоти повинні показувати інформацію про підключення клієнтів, розпарсені URL запитів, визначені файли та статус їх обробки. Також потрібно зафіксувати моменти успішної обробки запитів та ситуації з помилками.

Паралельно необхідно робити скріншоти браузера під час тестування різних сценаріїв використання. Скріншоти повинні демонструвати завантаження головної сторінки, відображення зображень, роботу favicon, та сторінку помилки 404. Рекомендується також показати інструменти розробника браузера з інформацією про HTTP заголовки та статус коди відповідей.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент отримує готовий скрипт простого HTTP-сервера, реалізованого на основі модуля socket. Завдання полягає не в написанні сервера «з нуля», а в аналізі його роботи, створенні відповідної структури веб-контенту та всебічному тестуванні функціональності сервера. Сервер обробляє запити до HTML-сторінок, JPEG-зображень та спеціального файлу favicon.ico, автоматично формує HTTP-відповіді з правильними заголовками Content-Type і підтримує базову обробку помилок (сторінка 404.html).

У протоколі обов'язково мають бути наведені три ключові елементи:

– завдання, у якому чітко описано індивідуальний варіант — зокрема, вимоги до структури каталогу www/ та змісту веб-сторінок;

– лістинг програми, що містить повний код наданого сервера (webserver.py) з коментарями, які пояснюють логіку обробки HTTP-запитів, парсинг URL, визначення типу контенту та формування відповідей;

– результати виконання програми, представлені у вигляді скріншотів: консольного виводу сервера під час обробки різних запитів (до існуючих і неіснуючих ресурсів, до зображень, до favicon.ico) та відображення цих ресурсів у веб-браузері (включаючи головну сторінку, зображення та сторінку помилки 404).

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності

у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 12

Тема: Робота з JSON і API для обробки даних з веб сервісів

Мета роботи: Навчитися виконувати HTTP-запити до публічних API, отримувати та обробляти дані у форматі JSON, виконувати аналіз та візуалізацію отриманої інформації.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Сучасна розробка програмного забезпечення неможлива без інтеграції з зовнішніми сервісами та обміну даними між різними системами. У епоху цифрової трансформації та хмарних технологій розробники постійно стикаються з необхідністю отримання актуальної інформації від віддалених серверів, будь то погодні дані, курси валют, соціальні мережі чи корпоративні бази даних. Ефективна взаємодія з веб-сервісами вимагає розуміння принципів роботи з API та форматами обміну даними, серед яких JSON займає провідне місце завдяки своїй універсальності та простоті. Python, як одна з найпопулярніших мов програмування, надає потужні інструменти для роботи з мережевими запитами та обробки JSON-даних через бібліотеки requests та вбудований модуль json.

API (Application Programming Interface) є набором правил та протоколів, що дозволяють різним програмним додаткам обмінюватися даними та функціональністю. У веб-розробці API забезпечують доступ до ресурсів сервера через HTTP-протокол, дозволяючи клієнтським додаткам отримувати, створювати, оновлювати та видаляти дані на віддалених серверах. Сучасні веб-API зазвичай працюють за принципом REST (Representational State Transfer), який визначає стандартні методи взаємодії з ресурсами. Основною перевагою використання API є можливість інтеграції зовнішніх сервісів без необхідності створення власної інфраструктури для зберігання та обробки даних. Публічні API надають доступ до величезної кількості інформації: погодні дані, курси валют, соціальні мережі, картографічні сервіси та багато іншого.

JSON (JavaScript Object Notation) став стандартним форматом обміну даними у веб-API завдяки своїй простоті, читабельності та підтримці більшістю сучасних мов програмування. Цей легкий текстовий формат базується на структурі "ключ-значення" та підтримує основні типи даних: рядки, числа, булеві значення, масиви та об'єкти. JSON є більш компактним та швидким у обробці порівняно з XML, що робить його ідеальним вибором для мобільних додатків та веб-сервісів. Структура JSON інтуїтивно зрозуміла розробникам та легко перетворюється у нативні структури даних різних мов програмування. При роботі з API важливо розуміти, що JSON-дані передаються у вигляді рядків та потребують десеріалізації для використання у програмі.

Бібліотека requests є найпопулярнішим інструментом для виконання HTTP-запитів у Python та значно спрощує взаємодію з веб-API. Ця бібліотека надає зрозумілий та елегантний інтерфейс для роботи з HTTP-протоколом,

автоматично обробляючи багато низькорівневих деталей. Основна філософія requests полягає у тому, щоб зробити HTTP-запити простими для людини, приховуючи складність протоколу за інтуїтивним API. Бібліотека автоматично обробляє кодування URL, параметри запитів, cookies, SSL-сертифікати та багато інших аспектів HTTP-взаємодії. Для встановлення бібліотеки використовується команда pip install requests, після чого вона стає доступною для імпорту у Python-скриптах.

Таблиця 12.1 демонструє основні методи бібліотеки requests та їх практичне застосування при роботі з веб-API.

Таблиця 12.1 - Основні методи бібліотеки requests

Метод	Призначення	Синтаксис	Приклад використання
get()	Отримання даних	requests.get(url, params=None)	requests.get('https://api.github.com/users/octocat')
post()	Створення ресурсу	requests.post(url, data=None, json=None)	requests.post('https://api.example.com/users', json={'name': 'John'})
put()	Оновлення ресурсу	requests.put(url, data=None, json=None)	requests.put('https://api.example.com/users/1', json={'name': 'Jane'})
delete()	Видалення ресурсу	requests.delete(url)	requests.delete('https://api.example.com/users/1')
head()	Отримання заголовків	requests.head(url)	requests.head('https://api.example.com/status')

Робота з бібліотекою requests починається з імпорту модуля та виконання HTTP-запиту. Найпростішим є GET-запит, який використовується для отримання даних від сервера без внесення змін. Функція requests.get() повертає об'єкт Response, який містить всю інформацію про відповідь сервера: статус-код, заголовки, тіло відповіді та методи для роботи з даними. Важливо завжди перевіряти статус-код відповіді перед обробкою даних, оскільки сервер може повернути помилку навіть у випадку успішного з'єднання. Статус-коди 200-299 вказують на успішне виконання запиту, 400-499 - на помилки клієнта, а 500-599 - на помилки сервера.

```
import requests

# Простий GET-запит
response                                                    =
requests.get('https://jsonplaceholder.typicode.com/posts/1')

# Перевірка статус-коду
if response.status_code == 200:
    print("Запит успішний")
    print(f"Тип контенту: {response.headers['content-type']}")
    print(f"Розмір відповіді: {len(response.text)} символів")
else:
    print(f"Помилка: {response.status_code}")
```

Модуль json є частиною стандартної бібліотеки Python та надає функціональність для серіалізації та десеріалізації JSON-даних. Серіалізація - це процес перетворення Python-об'єктів у JSON-рядки, а десеріалізація - зворотний процес перетворення JSON-рядків у Python-об'єкти. Модуль автоматично обробляє відповідність типів даних між Python та JSON: словники перетворюються в об'єкти JSON, списки - в масиви, рядки залишаються рядками, числа - числами, а булеві значення - булевими. При роботі з файлами модуль надає спеціальні функції для прямої роботи з файловими об'єктами без необхідності попереднього читання або запису рядків. Важливо пам'ятати, що не всі Python-об'єкти можуть бути серіалізовані в JSON - підтримуються лише базові типи даних.

Таблиця 12.2 наведена нижче показує основні функції модуля json та їх застосування для різних сценаріїв роботи з JSON-даними.

Таблиця 12.2 - Основні функції модуля json

Функція	Призначення	Синтаксис	Приклад використання
loads()	Рядок → Python-об'єкт	json.loads(json_string)	data = json.loads('{"name": "John", "age": 30}')
dumps()	Python-об'єкт → рядок	json.dumps(obj, indent=None)	json_str = json.dumps({"name": "John"}, indent=2)
load()	Файл → Python-об'єкт	json.load(file_object)	with open('data.json') as f: data = json.load(f)
dump()	Python-об'єкт → файл	json.dump(obj, file_object, indent=None)	with open('out.json', 'w') as f: json.dump(data, f)

Робота з JSON-даними у Python є інтуїтивною завдяки природній відповідності між структурами JSON та Python. JSON-об'єкти перетворюються у словники Python, масиви - у списки, а примітивні типи залишаються незмінними. При десеріалізації JSON-рядка функція json.loads() автоматично визначає структуру даних та створює відповідні Python-об'єкти. Для створення читабельного JSON-формату використовується параметр indent, який додає відступи та переноси рядків. Параметр ensure_ascii=False дозволяє зберігати символи Unicode у їх оригінальному вигляді замість escape-послідовностей.

```
import json

# Десеріалізація JSON-рядка
json_data = '{"users": [{"id": 1, "name": "Alice"}, {"id": 2, "name": "Bob"}]}'
python_data = json.loads(json_data)
print(f"Тип даних: {type(python_data)}")
print(f"Кількість користувачів: {len(python_data['users'])}")

# Серіалізація з форматуванням
formatted_json = json.dumps(python_data, indent=4, ensure_ascii=False)
print("Відформатований JSON:")
```

```
print(formatted_json)
```

Розглянемо практичний приклад, що демонструє основні принципи роботи з API та JSON при отриманні та обробці даних про користувачів з публічного API JSONPlaceholder. Це завдання типове для багатьох реальних проектів, де потрібно отримати дані від зовнішнього сервісу, обробити їх згідно з певними критеріями та зберегти результат для подальшого використання. JSONPlaceholder є тестовим API, який надає фіктивні дані про користувачів, пости та коментарі, що робить його ідеальним для навчання та тестування. У цьому прикладі ми отримаємо список користувачів, відфільтруємо тих, хто має певний домен електронної пошти, та збережемо результат у JSON-файл.

Мета завдання полягає у створенні програми, яка автоматично отримує дані про користувачів, аналізує їх електронні адреси та зберігає інформацію про користувачів з корпоративними доменами для подальшого аналізу. Такий підхід часто використовується у бізнес-аналітиці, маркетингу та управлінні клієнтськими базами даних. Програма має бути стійкою до мережевих помилок та некоректних даних, забезпечуючи надійну роботу в реальних умовах.

```
import requests
import json

def fetch_users_and_filter():
    """
    Отримує список користувачів з API та фільтрує їх за доменом пошти
    """
    # URL для отримання списку користувачів
    url = "https://jsonplaceholder.typicode.com/users"

    try:
        # Виконання GET-запиту до API
        print("Відправка запиту до API...")
        response = requests.get(url)

        # Перевірка статус-коду відповіді
        if response.status_code == 200:
            print("Дані успішно отримані")

            # Перетворення JSON-відповіді у Python-об'єкт
            users = response.json()
            print(f"Отримано {len(users)} користувачів")

            # Фільтрація користувачів з доменом .biz
            filtered_users = []
            for user in users:
                email = user.get('email', '')
                if email.endswith('.biz'):
                    # Створення спрощеного об'єкта користувача
                    filtered_user = {
                        'id': user.get('id'),
```

```
                        'name': user.get('name'),
                        'email': email,
                        'company':              user.get('company',
{}).get('name', 'N/A')
                    }
                    filtered_users.append(filtered_user)

            print(f"Знайдено {len(filtered_users)} користувачів з
доменом .biz")

            # Збереження відфільтрованих даних у JSON-файл
            with open('filtered_users.json', 'w', encoding='utf-
8') as file:
                json.dump(filtered_users,      file,      indent=4,
ensure_ascii=False)

            print("Дані збережено у файл filtered_users.json")

            # Виведення прикладу збережених даних
            if filtered_users:
                print("\nПриклад збережених даних:")
                print(json.dumps(filtered_users[0],       indent=2,
ensure_ascii=False))

        else:
            print(f"Помилка HTTP: {response.status_code}")

    except requests.exceptions.RequestException as e:
        print(f"Помилка мережевого з'єднання: {e}")
    except json.JSONDecodeError as e:
        print(f"Помилка обробки JSON: {e}")
    except Exception as e:
        print(f"Загальна помилка: {e}")

# Виклик функції
if __name__ == "__main__":
    fetch_users_and_filter()
```

Програма демонструє повний цикл роботи з API: від виконання HTTP-запиту до збереження оброблених даних. Функція requests.get() виконує запит до JSONPlaceholder API та отримує список користувачів у форматі JSON. Метод response.json() автоматично десеріалізує JSON-відповідь у Python-список словників, кожен з яких представляє окремого користувача. Фільтрація виконується за допомогою циклу for та методу str.endswith(), який перевіряє закінчення email-адреси. Використання методу dict.get() забезпечує безпечний доступ до полів словника навіть у випадку їх відсутності. Функція json.dump() зберігає відфільтровані дані у файл з читабельним форматуванням завдяки параметру indent=4. Обробка виключень забезпечує стійкість програми до мережевих помилок та некоректних даних, що є критично важливим при роботі з зовнішніми API.

2 КЛЮЧОВІ ПИТАННЯ

1. Що таке API і для чого воно використовується у програмуванні?
2. Який формат даних найчастіше використовується при обміні даними через веб-API?
3. Які основні методи HTTP використовуються при роботі з API?
4. Як у Python виконати GET-запит до веб-API?
5. Як перетворити JSON-рядок на Python-об’єкт?
6. Як перевірити, чи успішно виконано HTTP-запит?
7. Що таке API-ключ і чому він іноді потрібний?
8. Як зберегти отримані дані у JSON-файл?
9. Як запобігти помилці KeyError при зверненні до словника?
10. Як використовувати .env-файли для зберігання секретних даних?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Встановіть бібліотеку requests, якщо вона ще не встановлена, за допомогою команди pip install requests. Переконайтеся, що бібліотека успішно імпортується в Python-середовищі.
2. Оберіть один із публічних API, які не вимагають реєстрації, наприклад JSONPlaceholder або REST Countries. Ознайомтеся з його документацією та визначте URL для отримання списку об’єктів (наприклад, список користувачів або країн).
3. Напишіть функцію fetch_data(url), яка відправляє GET-запит до вказаного URL за допомогою requests.get(), перевіряє статус-код відповіді та повертає дані у вигляді Python-об’єкта, якщо запит успішний. У разі помилки запиту або відсутності з’єднання функція має виводити зрозуміле повідомлення та повертати None.
4. Отримайте дані з обраного API та виведіть на екран структуру перших двох-трьох елементів, щоб проаналізувати їхні ключі та значення.
5. Виконайте фільтрацію даних за будь-яким самостійно обраним критерієм — наприклад, відберіть країни з населенням понад 50 мільйонів, користувачів із доменом електронної пошти example.com або пости з кількістю коментарів понад 5. Результат збережіть у окремий список.
6. Збережіть відфільтровані дані у файл filtered_data.json у форматі JSON із читабельним форматуванням (відступи 4 пробіли), використовуючи контекстний менеджер with та функцію json.dump(). Переконайтеся, що файл створено та містить коректні дані.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New

Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент розробляє програму на мові Python, яка взаємодіє з публічним веб-API за допомогою бібліотеки requests, отримує дані у форматі JSON, виконує їх фільтрацію згідно з індивідуальним критерієм та зберігає результат у файл. Програма передбачає виконання HTTP-запиту, обробку відповіді сервера, аналіз структури даних, застосування логіки фільтрації та коректне серіалізування результатів у форматі JSON. Особливу увагу приділяється обробці помилок, валідації відповіді сервера, читабельності коду та дотриманню стандартів роботи з мережевими даними.

У протоколі обов'язково мають бути наведені три ключові елементи:

– завдання, у якому чітко описано індивідуальний варіант — зокрема, вибраний API, критерій фільтрації та очікуваний формат результату;

– лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють етапи виконання запиту, обробки JSON, фільтрації даних та збереження результатів;

– результати виконання програми, представлені у вигляді фрагментів консольного виводу (успішне отримання даних, кількість відфільтрованих записів) та вмісту згенерованого JSON-файлу, що підтверджують коректну роботу програми.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов'язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 13

Тема: Основи створення графічного інтерфейсу з використанням Tkinter

Мета роботи: ознайомитися з основами бібліотеки tkinter та навчитися створювати графічний інтерфейс користувача з елементами введення даних, кнопками, мітками та меню для реалізації простих обчислень.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Бібліотека Tkinter (Tool Kit Interface) є стандартною бібліотекою для створення графічних інтерфейсів користувача в мові програмування Python, яка входить до складу стандартної поставки Python починаючи з версії 1.4. Вона базується на наборі інструментів Tk, розробленому Джоном Остерхаутом у 1988 році для мови програмування Tcl. У 1991 році Гвідо ван Россум, творець Python, адаптував Tk для використання з Python, створивши модуль Tkinter. Протягом багатьох років бібліотека зазнавала численних удосконалень та модифікацій, зокрема в Python 3.x вона була перейменована на tkinter (з маленької літери). Незважаючи на появу більш сучасних альтернатив, таких як PyQt, Kivy або wxPython, Tkinter залишається найпопулярнішим вибором для створення простих та середньої складності графічних додатків завдяки своїй простоті, доступності та інтеграції з Python.

Основою будь-якого графічного інтерфейсу, створеного за допомогою Tkinter, є головне вікно (root window), яке створюється викликом функції Tk(). Це вікно служить контейнером для всіх інших елементів інтерфейсу, які називаються віджетами (widgets). До найважливіших віджетів належать Label для відображення тексту або зображень, Entry для введення текстових даних, Button для створення кнопок, Frame для групування інших віджетів, Menu для створення меню, Text для роботи з багаторядковим текстом, Listbox для відображення списків елементів, Checkbutton та Radiobutton для вибору опцій. Кожен віджет має власний набір властивостей та методів, які визначають його зовнішній вигляд та поведінку. Взаємодія між віджетами та програмним кодом здійснюється через систему подій та callback-функцій, що дозволяє реагувати на дії користувача, такі як натискання кнопок або введення тексту.

Рисунок 13.1 показує структурну схему взаємодії основних компонентів Tkinter-додатку. На схемі зображено головне вікно (Root Window) у центрі, від якого відходять зв'язки до основних груп компонентів: віджетів (Widgets) зліва, менеджерів геометрії (Geometry Managers) зверху, системи подій (Event System) справа та змінних Tkinter (Tkinter Variables) знизу. Кожна група містить відповідні елементи: віджети включають Label, Entry, Button та Menu; менеджери геометрії представлені pack(), grid() та place(); система подій містить callback-функції та bind(); змінні Tkinter включають StringVar, IntVar та DoubleVar. Стрілки показують напрямки взаємодії між компонентами, демонструючи циклічний характер обробки подій та оновлення інтерфейсу.

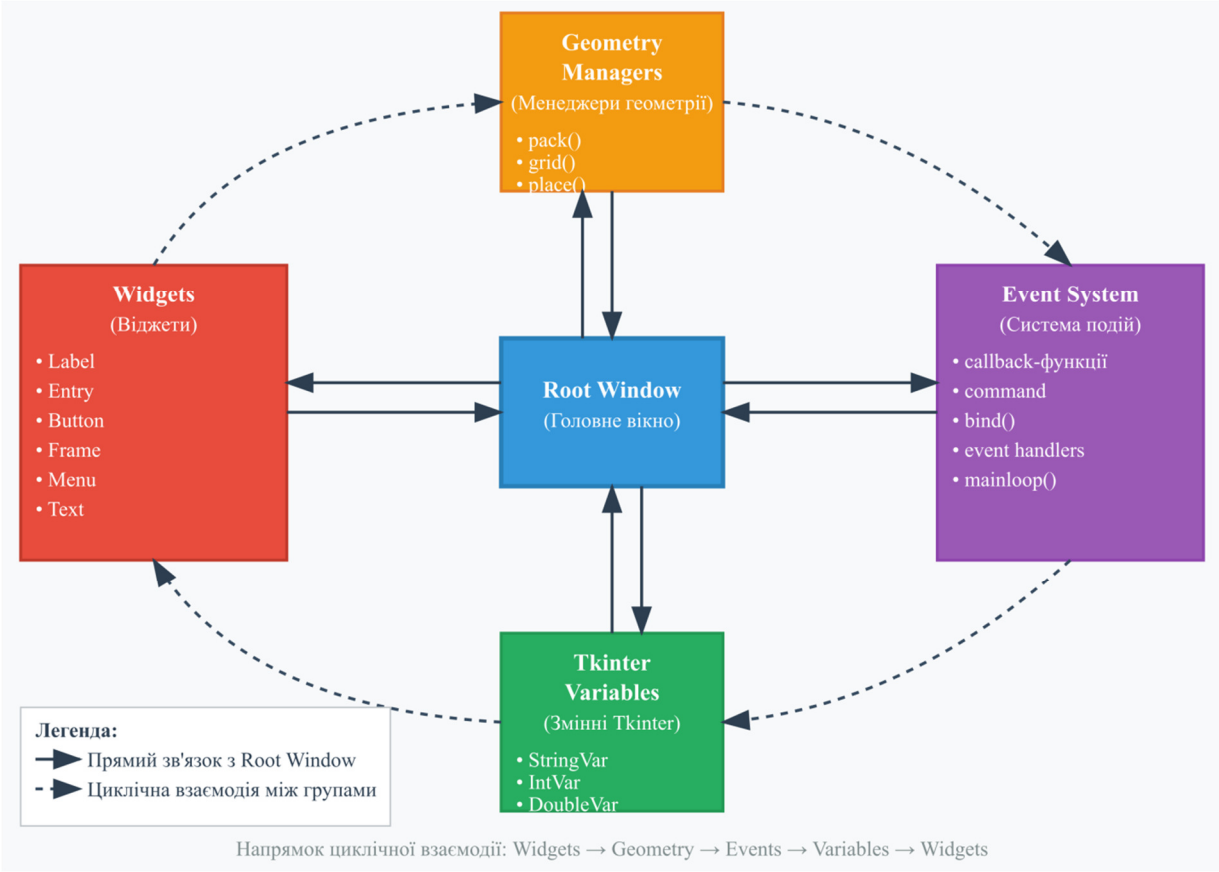


Рисунок 13.1 - Структурна схема взаємодії компонентів Tkinter-додатку

У таблиці 13.1 наведено основні функції та віджети бібліотеки Tkinter, які найчастіше використовуються при створенні графічних додатків для виконання обчислювальних завдань.

Таблиця 13.1 – Основні віджети та функції Tkinter

Віджет/Функція	Призначення	Основні параметри
Tk()	Створення головного вікна	title, geometry, resizable
Label	Відображення тексту або зображень	text, font, fg, bg, justify
Entry	Поле для введення тексту	textvariable, width, state
Button	Кнопка для виконання дій	text, command, state, width
Frame	Контейнер для групування віджетів	relief, borderwidth, bg
Menu	Створення меню додатку	tearoff, font
messagebox	Діалогові вікна повідомлень	showinfo, showerror, showwarning
StringVar	Змінна для зв'язування з віджетами	get(), set()
DoubleVar	Числова змінна	get(), set()

Головне вікно Tk() є фундаментальним елементом будь-якого Tkinter-додатку і створюється простим викликом функції tk.Tk(). Цей об'єкт представляє собою кореневе вікно операційної системи, в якому розміщуються всі інші елементи інтерфейсу. Після створення головного вікна можна налаштувати його основні властивості за допомогою відповідних методів. Метод title() встановлює заголовок вікна, який відображається в заголовковій панелі, наприклад root.title("Мій додаток"). Метод geometry() визначає розміри та початкове положення вікна у форматі "ширина x висота + x_позиція +

y_позиція", наприклад root.geometry("800x600+100+50"). Методи minsize() та maxsize() дозволяють встановити мінімальні та максимальні розміри вікна відповідно. Метод resizable() контролює можливість зміни розмірів вікна користувачем, приймаючи два булеві параметри для горизонтального та вертикального масштабування.

Таблиця 13.2 - Основні методи та властивості головного вікна Tk()

Метод/Властивість	Призначення	Синтаксис та приклад
title()	Встановлення заголовку вікна	root.title("Назва програми")
geometry()	Розміри та позиція вікна	root.geometry("800x600+100+50")
minsize()	Мінімальні розміри	root.minsize(400, 300)
maxsize()	Максимальні розміри	root.maxsize(1200, 800)
resizable()	Можливість зміни розмірів	root.resizable(True, False)
iconbitmap()	Іконка вікна	root.iconbitmap("icon.ico")
state()	Стан вікна	root.state("zoomed")
protocol()	Обробка системних подій	root.protocol("WM_DELETE_WINDOW", exit_func)

Віджет Label призначений для відображення статичного тексту або зображень і є одним з найпростіших елементів інтерфейсу. Він створюється конструктором tk.Label(parent, options), де parent - батьківський контейнер, а options - словник параметрів оформлення. Основним параметром є text, який визначає відображуваний текст, наприклад label = tk.Label(root, text="Введіть значення:"). Параметр font дозволяє налаштувати шрифт у форматі кортежу (назва_шрифту, розмір, стиль), наприклад font=("Arial", 14, "bold"). Кольори тексту та фону встановлюються параметрами fg (foreground) та bg (background) відповідно. Параметр justify контролює вирівнювання тексту і може приймати значення "left", "center" або "right". Для відображення багаторядкового тексту можна використовувати параметр wraplength, який визначає максимальну ширину рядка в пікселях перед автоматичним переносом.

Таблиця 13.3 - Параметри віджета Label

Параметр	Призначення	Можливі значення	Приклад
text	Текст для відображення	Рядок	text="Результат:"
font	Шрифт тексту	Кортеж (назва, розмір, стиль)	font=("Times", 12, "italic")
fg	Колір тексту	Назва або hex-код	fg="red", fg="#FF0000"
bg	Колір фону	Назва або hex-код	bg="yellow", bg="#FFFF00"
justify	Вирівнювання тексту	"left", "center", "right"	justify="center"
anchor	Позиція тексту в області	"n", "s", "e", "w", "center"	anchor="w"
relief	Стиль рамки	"flat", "raised", "sunken"	relief="raised"
borderwidth	Товщина рамки	Ціле число	borderwidth=2

Віджет Entry являє собою однорядкове поле для введення тексту і є основним засобом отримання даних від користувача. Створюється за допомогою конструктора tk.Entry(parent, options) і підтримує різноманітні методи для роботи з текстовими даними. Метод get() повертає поточний текст з поля, а метод insert(index, string) вставляє текст на вказану позицію. Метод delete(start, end) видаляє текст у вказаному діапазоні, де tk.END означає кінець рядка. Параметр textvariable дозволяє прив'язати поле до змінної Tkinter для автоматичної синхронізації даних. Параметр state може приймати значення "normal" для звичайного режиму, "disabled" для блокування вводу або "readonly" для режиму тільки для читання. Параметр show використовується для створення полів паролів, встановлюючи символ для маскування введених даних.

Таблиця 13.4 - Методи та параметри віджета Entry

Метод/Параметр	Призначення	Синтаксис	Приклад
get()	Отримання тексту	entry.get()	value = entry.get()
insert()	Вставка тексту	insert(index, text)	entry.insert(0, "default")
delete()	Видалення тексту	delete(start, end)	entry.delete(0, tk.END)
textvariable	Прив'язка до змінної	StringVar()	textvariable=var
width	Ширина поля	Ціле число	width=20
state	Стан поля	"normal", "disabled", "readonly"	state="normal"
show	Символ маскування	Символ	show="*"
validate	Валідація вводу	"key", "focus", "all"	validate="key"

Віджет Button створює інтерактивну кнопку, яка виконує певну дію при натисканні користувачем. Конструктор tk.Button(parent, options) приймає обов'язковий параметр command, який вказує на функцію-обробник, що викликається при натисканні кнопки. Ця функція повинна бути передана без дужок, наприклад command=calculate_function. Параметр text встановлює напис на кнопці, а параметри width та height визначають її розміри в символах та рядках відповідно. Кольорове оформлення контролюється параметрами bg для фону, fg для тексту та activebackground для кольору при наведенні курсора. Параметр state дозволяє блокувати кнопку, встановлюючи значення "disabled". Для додання клавіатурних скорочень можна використовувати параметр underline, який підкреслює вказаний символ.

Таблиця 13.5 – Параметри та методи віджета Button

Параметр/Метод	Призначення	Тип значення	Приклад
command	Функція-обробник	Функція без дужок	command=my_function
text	Текст кнопки	Рядок	text="Обчислити"
width	Ширина в символах	Ціле число	width=15
height	Висота в рядках	Ціле число	height=2
state	Стан кнопки	"normal", "disabled"	state="normal"
relief	Стиль кнопки	"raised", "sunken", "flat"	relief="raised"
activebackground	Колір при активації	Колір	activebackground="lightblue"
invoke()	Програмне натискання	button.invoke()	btn.invoke()

Віджет Frame служить контейнером для групування інших віджетів і не має власного видимого вмісту. Він створюється конструктором tk.Frame(parent, options) і використовується для організації складних інтерфейсів з логічним розділенням елементів. Frame може мати власне оформлення через параметри relief та borderwidth, які створюють візуальну рамку навколо групи елементів. Параметр bg встановлює колір фону фрейму, що може бути корисно для візуального виділення різних секцій інтерфейсу. Усередині фрейму можна розміщувати інші віджети, використовуючи фрейм як батьківський контейнер. Це особливо корисно при використанні різних менеджерів геометрії в одному вікні або для створення повторюваних груп елементів.

Віджет Menu дозволяє створювати меню додатку з розкривними списками команд та підменю. Створення меню відбувається у два етапи: спочатку створюється об'єкт головного меню tk.Menu(root), який прив'язується до вікна методом config(menu=menubar). Потім створюються окремі розкривні меню як об'єкти Menu з параметром tearoff=0 для заборони відокремлення меню. Команди додаються методом add_command() з параметрами label для назви пункту та command для функції-обробника. Метод add_separator() додає роздільну лінію між пунктами меню. Для створення підменю використовується метод add_cascade(), який дозволяє вкладати одне меню в інше. Меню автоматично підтримує клавіатурну навігацію та стандартні комбінації клавіш операційної системи.

Таблиця 13.6 – Методи класу Menu

Метод	Призначення	Параметри	Приклад
add_command()	Додавання команди	label, command, accelerator	add_command(label="Відкрити", command=open_file)
add_separator()	Додавання роздільника	Немає	add_separator()
add_cascade()	Додавання підменю	label, menu	add_cascade(label="Файл", menu=file_menu)
add_checkbutton()	Додавання прапорця	label, variable, command	add_checkbutton(label="Показати", variable=show_var)
add_radiobutton()	Додавання перемикача	label, variable, value	add_radiobutton(label="Опція1", variable=opt_var, value=1)

delete()	Видалення пункту	index	delete(0)
entryconfig()	Зміна властивостей	index, options	entryconfig(0, state="disabled")

Модуль messagebox надає набір готових діалогових вікон для відображення повідомлень користувачу та отримання простих відповідей. Основними функціями є showinfo() для інформаційних повідомлень, showwarning() для попереджень, showerror() для повідомлень про помилки. Функції askquestion(), askyesno(), askokcancel() та askretrycancel() дозволяють отримувати відповіді користувача у форматі булевих значень або рядків. Всі функції приймають два основні параметри: title для заголовка діалогового вікна та message для тексту повідомлення. Діалогові вікна є модальними, тобто блокують роботу з основним вікном до закриття діалогу. Це забезпечує належну обробку критичних ситуацій та гарантує, що користувач побачить важливе повідомлення.

Змінні Tkinter представляють собою спеціальні об'єкти для зв'язування даних між віджетами та програмним кодом. StringVar() використовується для текстових даних, IntVar() для цілих чисел, DoubleVar() для чисел з плаваючою комою, BooleanVar() для булевих значень. Ці змінні створюються викликом відповідного конструктора і прив'язуються до віджетів через параметр textvariable або variable. Метод get() повертає поточне значення змінної, а метод set() встановлює нове значення з автоматичним оновленням всіх прив'язаних віджетів. Змінні також підтримують callback-функції через метод trace(), який дозволяє відстежувати зміни значення та автоматично викликати обробники. Це забезпечує реактивність інтерфейсу та спрощує синхронізацію даних між різними елементами.

Таблиця 13.7 – Типи змінних Tkinter та їх методи

Тип змінної	Призначення	Методи	Приклад використання
StringVar	Текстові дані	get(), set(), trace()	name_var = StringVar(); entry.config(textvariable=name_var)
IntVar	Цілі числа	get(), set(), trace()	count_var = IntVar(); count_var.set(10)
DoubleVar	Дійсні числа	get(), set(), trace()	price_var = DoubleVar(); price_var.set(19.99)
BooleanVar	Булеві значення	get(), set(), trace()	check_var = BooleanVar(); check_var.set(True)

Розміщення віджетів у вікні здійснюється за допомогою менеджерів геометрії, серед яких найпоширенішими є pack(), grid() та place(). Менеджер pack() розміщує віджети послідовно один за одним у вертикальному або горизонтальному напрямку, що зручно для простих інтерфейсів. Менеджер grid() дозволяє розміщувати елементи у вигляді таблиці з рядками та стовпцями, забезпечуючи більш точний контроль над позиціонуванням.

Менеджер place() надає можливість вказати точні координати розміщення віджета, але рідко використовується через складність підтримки при зміні розмірів вікна.

Обробка подій у Tkinter реалізується через систему callback-функцій, які прив'язуються до віджетів за допомогою параметра command або методу bind(). Коли користувач взаємодіє з елементом інтерфейсу, викликається відповідна функція-обробник, яка може виконувати обчислення, оновлювати дані або змінювати стан інших віджетів. Для передачі даних між віджетами та програмним кодом використовуються спеціальні змінні Tkinter, такі як StringVar(), IntVar(), DoubleVar() та BooleanVar(). Ці змінні автоматично синхронізуються з віджетами, до яких вони прив'язані, що значно спрощує роботу з даними інтерфейсу.

Створення меню додатку здійснюється за допомогою класу Menu, який дозволяє додавати розкривні меню з різними пунктами та підменю. Типове меню містить пункти "Файл", "Редагування", "Довідка" тощо, кожен з яких може мати власні команди. Для забезпечення зручності використання рекомендується додавати стандартні пункти меню, такі як "Очистити" для скидання введених даних та "Вихід" для завершення роботи програми. Валідація введених даних є критично важливою для забезпечення стабільної роботи додатку та запобігання помилкам виконання. Вона може включати перевірку типу даних, діапазону значень, обов'язковості заповнення полів та інші критерії відповідно до специфіки завдання.

Розглянемо практичний приклад створення простого графічного додатку для обчислення суми двох чисел, який демонструє основні принципи роботи з Tkinter.

```
import tkinter as tk
from tkinter import messagebox

def calculate_sum():
    try:
        # Отримуємо значення з полів введення
        num1 = float(entry1.get())
        num2 = float(entry2.get())

        # Виконуємо обчислення
        result = num1 + num2

        # Відображаємо результат у поле вводу
        result_entry.config(state="normal")         #     Дозволяємо
редагування
        result_entry.delete(0, tk.END)        # Очищуємо поле
        result_entry.insert(0,    str(result))        #    Вставляємо
результат
        result_entry.config(state="readonly")   #   Знову   блокуємо
редагування

    except ValueError:
```

```
        messagebox.showerror("Помилка", "Введіть коректні числові
значення!")
    except Exception as e:
        messagebox.showerror("Помилка",     f"Сталася     помилка:
{str(e)}")

def clear_data():
    entry1.delete(0, tk.END)
    entry2.delete(0, tk.END)
    result_entry.config(state="normal")
    result_entry.delete(0, tk.END)
    result_entry.config(state="readonly")

def exit_app():
    root.quit()

# Створення головного вікна
root = tk.Tk()
root.title("Калькулятор суми двох чисел")
root.geometry("400x300")

# Створення меню
menubar = tk.Menu(root)
root.config(menu=menubar)

file_menu = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="Файл", menu=file_menu)
file_menu.add_command(label="Очистити", command=clear_data)
file_menu.add_separator()
file_menu.add_command(label="Вихід", command=exit_app)

# Створення елементів інтерфейсу
tk.Label(root,       text="Перше       число:",       font=("Arial",
12)).grid(row=0, column=0, padx=10, pady=10, sticky="w")
entry1 = tk.Entry(root, font=("Arial", 12), width=20)
entry1.grid(row=0, column=1, padx=10, pady=10)

tk.Label(root,       text="Друге       число:",       font=("Arial",
12)).grid(row=1, column=0, padx=10, pady=10, sticky="w")
entry2 = tk.Entry(root, font=("Arial", 12), width=20)
entry2.grid(row=1, column=1, padx=10, pady=10)

# Кнопка для обчислення
tk.Button(root, text="Обчислити суму", command=calculate_sum,
          font=("Arial",      12),      bg="lightblue").grid(row=2,
column=0, columnspan=2, pady=20)

# Поле для відображення результату
tk.Label(root,  text="Результат:",  font=("Arial",  12)).grid(row=3,
column=0, padx=10, pady=10, sticky="w")
result_entry   =   tk.Entry(root,   font=("Arial",   12),   width=20,
state="readonly", bg="lightgray")
result_entry.grid(row=3, column=1, padx=10, pady=10)
```

```
# Запуск головного циклу
root.mainloop()
```

Даний приклад демонструє ключові концепції створення графічного інтерфейсу: ініціалізацію головного вікна, створення та розміщення віджетів, обробку подій через callback-функції, валідацію введених даних та відображення результатів. Програма включає обробку винятків для запобігання аварійному завершенню при некоректному введенні даних, а також меню з функціями очищення та виходу. Використання менеджера grid() забезпечує акуратне розташування елементів у вигляді таблиці, що створює професійний вигляд інтерфейсу.

При розробці графічних додатків важливо дотримуватися принципів хорошого дизайну інтерфейсу користувача. Елементи повинні бути логічно згруповані та послідовно розташовані, щоб користувач міг інтуїтивно зрозуміти призначення кожного з них. Рекомендується використовувати зрозумілі назви для кнопок та міток, застосовувати однакові шрифти та кольорову схему в межах одного додатку, а також забезпечувати достатні відступи між елементами для зручності використання. Обов'язковою є реалізація механізмів валідації даних та інформування користувача про помилки через діалогові вікна або спеціальні поля повідомлень.

Особливу увагу слід приділити оптимізації продуктивності додатку, особливо при роботі з великими обсягами даних або складними обчисленнями. Tkinter працює в одному потоці, тому тривалі операції можуть призводити до "зависання" інтерфейсу. У таких випадках рекомендується використовувати методи after() для виконання операцій частинами або застосовувати багатопоточність з обережним оновленням інтерфейсу. Також важливо правильно організувати структуру коду, виділяючи окремі функції для обчислень, валідації даних та оновлення інтерфейсу, що полегшує подальше розширення та підтримку додатку.

2 КЛЮЧОВІ ПИТАННЯ

1. Які основні елементи графічного інтерфейсу можна створити за допомогою бібліотеки tkinter?
2. Як правильно ініціалізувати змінні tkinter для зв'язування даних між віджетами та програмним кодом?
3. Які менеджери геометрії використовуються в tkinter для розміщення елементів інтерфейсу та які їх особливості?
4. Як реалізується обробка подій та прив'язка функцій-обробників до елементів інтерфейсу в tkinter?
5. Як створити меню додатку з пунктами "Очистити" та "Вихід" за допомогою класу Menu?
6. Які методи валідації введених даних необхідно реалізувати для забезпечення стабільної роботи додатку?

7. Як організувати відображення результатів обчислень у графічному інтерфейсі з використанням полів виведення?
8. Які особливості роботи з різними типами віджетів (поля введення, кнопки, мітки) в tkinter?
9. Як реалізувати систему повідомлень про помилки та попередження для користувача?
10. Які принципи проектування графічного інтерфейсу забезпечують зручність використання додатку?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Кожен студент отримує індивідуальне завдання згідно з варіантом, який визначається за номером студента в журналі. Завдання передбачає створення графічного додатку з використанням бібліотеки tkinter, який повинен містити елементи введення даних, кнопки для запуску обчислень та відображення результатів. Всі варіанти завдань представлені в таблиці 13.8, де кожен номер варіанта відповідає певному типу обчислень або обробки даних.

При виконанні завдання студент повинен реалізувати повноцінний графічний інтерфейс, який забезпечує зручну роботу користувача з додатком. Крім основного функціоналу, кожен додаток повинен мати меню "Файл" з пунктами "Очистити" для скидання введених даних та "Вихід" для завершення роботи програми. Важливою вимогою є належна обробка помилок та некоректного введення даних, щоб забезпечити стабільну роботу додатку. Результатом виконання лабораторної роботи повинен бути повноцінний робочий додаток, який відповідає заданому варіанту завдання та вимогам до оформлення графічного інтерфейсу.

Таблиця 13.8 – Варіанти індивідуальних завдань

№ з/п	Завдання
1	Обчислення площі прямокутника за двома сторонами
2	Обчислення площі трикутника за формулою Герона
3	Обчислення об'єму куба за стороною
4	Обчислення довжини кола за радіусом
5	Обчислення площі кола за радіусом
6	Обчислення периметра прямокутника
7	Обчислення площі трапеції
8	Обчислення об'єму паралелепіпеда
9	Обчислення площі ромба за діагоналями
10	Обчислення гіпотенузи прямокутного трикутника
11	Обчислення середнього арифметичного двох чисел
12	Обчислення середнього геометричного двох чисел
13	Переведення температури з Цельсія у Фаренгейти
14	Переведення температури з Фаренгейта у Цельсія
15	Обчислення відстані між двома точками на площині
16	Обчислення дискримінанта квадратного рівняння

17	Обчислення факторіалу числа
18	Переведення кілометрів у милі
19	Переведення фунтів у кілограми
20	Обчислення суми арифметичної прогресії
21	Обчислення суми геометричної прогресії
22	Обчислення катета прямокутного трикутника
23	Обчислення площі паралелограма
24	Обчислення об'єму циліндра
25	Обчислення площі поверхні куба
26	Обчислення площі поверхні паралелепіпеда
27	Обчислення об'єму конуса
28	Обчислення площі поверхні сфери
29	Обчислення об'єму сфери
30	Обчислення відсоткової ставки від суми

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов’язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент розробляє простий графічний додаток за допомогою стандартної бібліотеки Python tkinter. Програма передбачає створення головного вікна, розміщення елементів інтерфейсу (міток, полів введення, кнопок), реалізацію обробників подій (зокрема — функції, що виконуються при натисканні кнопки), виконання обчислень згідно з індивідуальним варіантом та відображення результату у графічному вигляді. Особливу увагу приділяється організації компонування елементів, читабельності коду, валідації введених даних та належному оформленню інтерфейсу.

У протоколі обов’язково мають бути наведені три ключові елементи:

– по-перше, завдання, у якому чітко описано індивідуальний варіант (математична формула або тип обчислень);

– по-друге, лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють призначення GUI-елементів, обробників подій та логіки взаємодії;

– по-третє, результати виконання програми, представлені у вигляді скріншотів робочого вікна програми — як до, так і після виконання обчислень (включаючи нормальні сценарії, помилкове введення даних, роботу меню «Очистити» та «Вихід»), що підтверджує коректну роботу інтерфейсу.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

Лабораторна робота № 14

Тема: Апроксимація функцій одношаровою нейронною мережею з використанням бібліотеки TensorFlow/Keras.

Мета роботи: створити та навчити просту нейронну мережу з одним прихованим шаром для апроксимації заданої математичної функції.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Бібліотека TensorFlow була розроблена командою Google Brain і вперше представлена у листопаді 2015 року як відкрита платформа для машинного навчання. Початково TensorFlow створювався для внутрішніх потреб Google, але згодом став одним з найпопулярніших інструментів для розробки систем штучного інтелекту. Keras, розроблений Франсуа Шолле у 2015 році, спочатку був незалежною високорівневою бібліотекою для роботи з нейронними мережами, яка могла працювати поверх різних бекендів, включаючи TensorFlow, Theano та CNTK. У 2017 році Keras був інтегрований безпосередньо в TensorFlow як tf.keras, ставши його офіційним високорівневим API. Ця інтеграція значно спростила процес створення та навчання нейронних мереж, зробивши TensorFlow доступнішим для широкого кола дослідників та розробників.

Еволюція екосистеми TensorFlow/Keras характеризувалася постійним удосконаленням архітектури та розширенням функціональності. TensorFlow 2.0, випущений у 2019 році, кардинально змінив підхід до розробки, зробивши eager execution стандартним режимом роботи та спростивши API. Keras став центральним компонентом цієї нової версії, забезпечивши інтуїтивний інтерфейс для створення моделей різної складності. Сучасні версії TensorFlow підтримують широкий спектр застосувань - від простих регресійних моделей до складних архітектур глибокого навчання, включаючи згорткові мережі, рекурентні мережі та трансформери.

Робота з нейронними мережами в TensorFlow/Keras включає декілька ключових етапів, які забезпечують повний цикл розробки моделі машинного навчання. Перший етап передбачає підготовку даних, яка включає завантаження, очищення та нормалізацію вхідних даних, а також їх розділення на навчальну та тестову вибірки. Другий етап - це визначення архітектури моделі за допомогою Sequential або Functional API, де розробник специфікує кількість шарів, типи активаційних функцій та інші параметри мережі. Третій етап включає компіляцію моделі з вибором оптимізатора, функції втрат та метрик для оцінки якості. Четвертий етап - це процес навчання моделі на навчальних даних з можливістю моніторингу прогресу через callbacks та візуалізацію метрик.

П'ятий етап роботи з нейронними мережами охоплює оцінку якості навченої моделі на тестових даних та аналіз результатів. Цей етап включає перевірку моделі на нових даних, аналіз метрик якості та виявлення можливих

проблем, таких як перенавчання або недонавчання. Останній етап передбачає використання навченої моделі для прогнозування на нових даних та її подальше вдосконалення при необхідності. Весь цей процес є ітеративним, оскільки часто потрібне налаштування гіперпараметрів, зміна архітектури або додаткова обробка даних для досягнення оптимальних результатів.

Таблиця 14.1 – Етапи створення нейронної мережі в TensorFlow/Keras

Етап	Назва етапу	Основні функції TensorFlow/Keras
1	Підготовка даних	numpy arrays, train_test_split
2	Створення архітектури	Sequential(), Dense(), input_shape
3	Компіляція моделі	model.compile(), optimizers, loss functions
4	Навчання моделі	model.fit(), epochs, batch_size
5	Оцінка та прогнозування	model.predict(), model.evaluate()
6	Візуалізація результатів	history object, matplotlib

Практична реалізація створення нейронної мережі в TensorFlow/Keras демонструється на прикладі скрипта для апроксимації математичної функції, який починається з імпорту необхідних бібліотек, як показано в таблиці 14.1.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
from tensorflow.keras import initializers
import tensorflow as tf
```

Ці імпорти забезпечують доступ до основних інструментів для роботи з даними, візуалізації результатів та створення нейронної мережі. Функція визначає цільову математичну залежність, яку необхідно апроксимувати за допомогою нейронної мережі:

```
def user_function(X):
    Y = np.power(X - 5, 2)
    return Y
```

Налаштування змінної середовища дозволяє приховати додаткові повідомлення TensorFlow для зручності роботи:

```
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Етап підготовки даних в скрипті реалізується через створення навчальної та тестової вибірок за допомогою функції np.arange().

```
# ---------- input data -----------------
N_neurons=20
epochs=1000
batch_size=1
```

```
learning_rate=0.01

# train seq.
X_train=np.arange(-10,15,1)
Y_train=user_function(X_train)
# test seq.
X_test=np.arange(-5,10,0.1)
Y_calc=user_function(X_train)
```

Ці рядки генерують 25 навчальних точок з дискретним кроком для діапазону від -10 до 15, які використовуються для навчання мережі. Тестова вибірка створюється з більш густою сіткою з 150 точок для детальної перевірки якості апроксимації. Параметри навчання визначають архітектуру мережі та процес навчання. Правильне формування розмірностей даних є критичним, оскільки TensorFlow/Keras очікує певний формат вхідних масивів для ефективної обробки.

Створення архітектури нейронної мережі в скрипті здійснюється через послідовність команд, що формують модель з одним прихованим шаром.

```
# define and fit the final model
model = Sequential()
model.add(Dense(N_neurons,                          input_shape=(1,),
kernel_initializer='random_normal', use_bias=True,
                bias_initializer='zeros',     activation='sigmoid',
name='Hidden_L1'))
model.add(Dense(1, activation='linear', name='OUT_Layer'))


# summarize layers
model.summary()
```

Перший рядок ініціалізує контейнер для послідовного додавання шарів мережі. Прихований шар містить 20 нейронів, одномірний вхід, випадкову ініціалізацію ваг та сигмоїдну активаційну функцію. Вихідний шар створюється з одним нейроном з лінійною активацією, що відповідає задачі регресії. Метод model.summary() виводить детальну інформацію про створену архітектуру, включаючи кількість параметрів та структуру з'єднань.

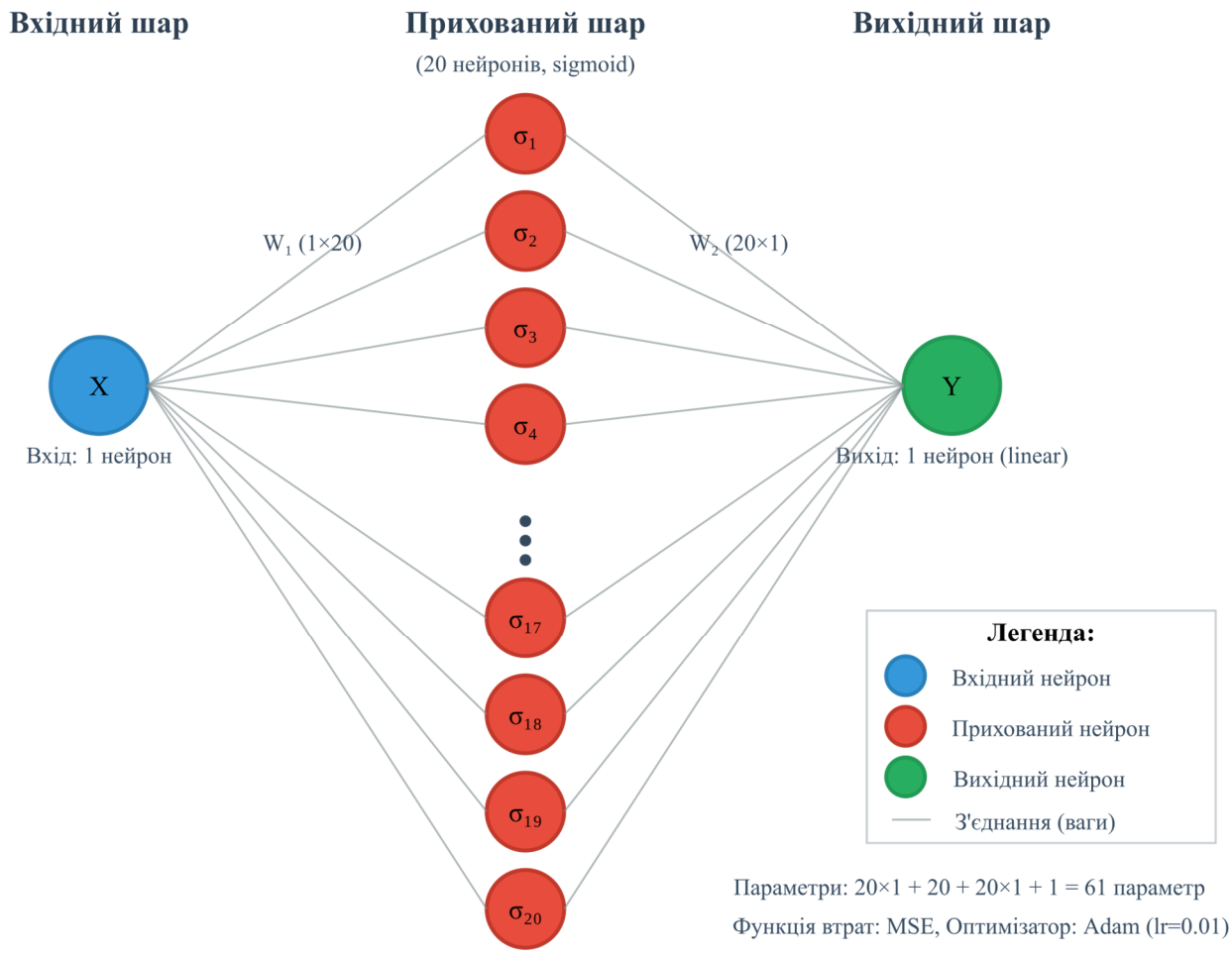


Рисунок 14.1 – Структура нейронної мережі для апроксимації функції

Структура нейронної мережі, що реалізується в скрипті, являє собою класичну архітектуру прямого поширення з одним прихованим шаром із 20 нейронів. Вхідний шар приймає одне значення координати X, яке через повні з'єднання подається на кожен з 20 нейронів прихованого шару. Кожен нейрон прихованого шару має власну вагу w_i, зміщення b_i та використовує сигмоїдну активаційну функцію σ(x) = 1/(1+e^(-x)) для нелінійного перетворення вхідного сигналу. Вихідні значення всіх нейронів прихованого шару з'єднуються з єдиним вихідним нейроном, який має лінійну активаційну функцію f(x) = x та видає фінальне передбачення значення функції Y. Зв'язки між шарами представлені матрицями ваг W1 (1×20) та W2 (20×1), які коригуються під час навчання методом зворотного поширення помилки для мінімізації функції втрат між передбаченими та справжніми значеннями функції.

Компіляція моделі в скрипті встановлює ключові параметри процесу навчання:

```
model.compile(optimizer=optimizers.Adam(learning_rate),
loss='mse', metrics=['mae'])
```

Оптимізатор Adam з швидкістю навчання 0.01 забезпечує ефективне та стабільне оновлення параметрів мережі, використовуючи адаптивні швидкості навчання для кожного параметра. Функція втрат 'mse' (середньоквадратична помилка) обчислює квадрат різниці між передбаченими та справжніми

значеннями, що особливо підходить для задач регресії. Метрика 'mae' (середня абсолютна помилка) надає додаткову інформацію про якість навчання, обчислюючи середнє абсолютне відхилення передбачень від цільових значень. Ці параметри визначають, як мережа буде навчатися та які показники будуть відстежуватися під час тренування.

Процес навчання моделі запускається командою, яка здійснює ітеративне оновлення параметрів мережі протягом 1000 епох:

```
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, verbose=1)
```

Параметр batch_size=1 означає, що ваги оновлюються після обробки кожного навчального зразка, що забезпечує детальне налаштування на малому наборі даних. Параметр verbose=1 забезпечує виведення детальної інформації про прогрес навчання, включаючи значення функції втрат та метрик для кожної епохи. Об'єкт history зберігає всю історію навчання, включаючи значення 'loss' та 'mae' для кожної епохи, що дозволяє проаналізувати динаміку збіжності та виявити можливі проблеми, такі як перенавчання або нестабільність процесу оптимізації.

Етап оцінки та прогнозування реалізується командою, яка використовує навчену модель для генерації передбачень на тестовому наборі даних:

```
y_predict = model.predict(X_test)  # прогноз
```

Метод predict() обробляє всі 150 тестових точок та повертає відповідні значення апроксимованої функції, які можуть бути порівняні з аналітичними значеннями для оцінки якості. Візуалізація результатів здійснюється за допомогою matplotlib через створення двох графіків:

```
# plot metrics
fig1 = plt.figure()
plt.plot(history.history['mae'])
plt.plot(history.history['loss'])
plt.legend(['mae', 'loss'], loc='upper right')
plt.ylabel('mae+loss')
plt.xlabel('epoch')
plt.grid()
fig1.savefig('figure1.png', dpi=300)


fig2 = plt.figure()
plt.plot(X_train,Y_train,'o')
plt.plot(X_test,y_predict)
plt.grid()
```

```
plt.legend(['Y_train','y_predict'])
plt.title('$y=(x-5)^{3}
fig2.savefig('figure2.png', dpi=300)
```

Перший графік показує динаміку метрик навчання, а другий порівнює навчальні точки з апроксимованою кривою. Збереження графіків у файли дозволяє зберегти результати для подальшого аналізу та документування.

Апроксимація функцій за допомогою нейронних мереж є фундаментальною задачею машинного навчання, яка демонструє здатність штучних нейронних мереж наближати довільні неперервні функції. Універсальна теорема апроксимації стверджує, що нейронна мережа з одним прихованим шаром та достатньою кількістю нейронів може апроксимувати будь-яку неперервну функцію на компактній множині з довільною точністю. Ця властивість робить нейронні мережі потужним інструментом для моделювання складних залежностей між вхідними та вихідними даними. В контексті регресійних задач нейронні мережі навчаються знаходити оптимальне відображення від простору вхідних змінних до простору цільових значень, мінімізуючи функцію втрат на навчальних даних.

Процес апроксимації функції нейронною мережею полягає в ітеративному налаштуванні ваг та зміщень мережі таким чином, щоб мінімізувати різницю між передбаченими та справжніми значеннями функції. Нелінійні активаційні функції в прихованих шарах дозволяють мережі моделювати складні нелінійні залежності, які неможливо описати простими лінійними моделями. Кількість нейронів у прихованому шарі визначає складність функції, яку може апроксимувати мережа - більше нейронів дозволяє моделювати більш складні залежності, але може призвести до перенавчання. Правильний вибір архітектури мережі, функції втрат та гіперпараметрів навчання є ключовим для досягнення високої якості апроксимації при збереженні здатності моделі узагальнювати на нових даних.

Таблиця 14.2 – Основні параметри нейронної мережі в скрипті

Параметр	Значення	Призначення
N_neurons	20	Кількість нейронів у прихованому шарі
epochs	1000	Кількість епох навчання
batch_size	1	Розмір батча для навчання
learning_rate	0.01	Швидкість навчання оптимізатора Adam
Активація прихованого шару	sigmoid	Функція активації нейронів
Активація вихідного шару	linear	Лінійна активація для регресії
Функція втрат	mse	Середньоквадратична помилка
Метрика	mae	Середня абсолютна помилка

Результати роботи скрипта, представлені в таблиці 1.2, візуалізуються за допомогою двох графіків, які зберігаються у файли для подальшого аналізу. Перший графік (figure1.png) показує динаміку метрик навчання - MAE та MSE - протягом 1000 епох, що дозволяє оцінити швидкість збіжності та стабільність

процесу навчання. Ідеальна крива навчання характеризується монотонним зменшенням обох метрик до стабілізації на низькому рівні без коливань. Другий графік (figure2.png) порівнює 25 навчальних точок з апроксимованою кривою, отриманою на 150 тестових точках, демонструючи якість апроксимації функції $Y = (X-5)^2$ навченою мережею. Висока якість апроксимації проявляється в точному проходженні кривої через навчальні точки та плавному відтворенні характеру цільової функції в усьому діапазоні значень.

2 КЛЮЧОВІ ПИТАННЯ

1. Яка роль прихованого шару в одношаровій нейронній мережі для задачі апроксимації функції?
2. Як кількість епох навчання впливає на процес навчання, і як можна визначити оптимальну кількість епох для конкретної функції?
3. Як відрізняються призначення навчального (`X_train`, `Y_train`) та тестового (`X_test`, `y_predict`) наборів даних у цій роботі?
4. Що характеризує функція втрат `mse` (середньоквадратична помилка), і як вона використовується під час навчання моделі?
5. Який фізичний зміст мають метрики `loss` і `mae`, відображені на графіку процесу навчання?
6. Як впливає кількість нейронів у прихованому шарі (`N_neurons`) на здатність моделі апроксимувати функцію?
7. Як зміна гіперпараметрів, таких як швидкість навчання (`learning_rate`) і кількість епох (`epochs`), впливає на процес та результат навчання?
8. Як за графіком результатів апроксимації (рис. 1.2) можна оцінити якість роботи навченої нейронної мережі?
9. Які можливі причини можуть призвести до поганої апроксимації функції навіть після тривалого навчання?
10. Яке практичне значення має вміння апроксимувати функції за допомогою нейронних мереж у інших галузях науки чи техніки?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

Студентам необхідно взяти за основу готовий скрипт NeuralN_regress_1.py, розглянутий у ключових положеннях, та модифікувати його для апроксимації індивідуального математичного виразу згідно з вихідними даними. Основне завдання полягає в аналізі процесу навчання нейронної мережі для конкретного випадку та дослідженні впливу параметрів навчання на якість апроксимації.

Початковим етапом є запуск програми з демонстраційною функцією Y = np.power(X - 5, 2) для розуміння структури коду та процесу навчання. Після цього студент обирає свій варіант функції з таблиці нижче та реалізує її в функції user_function, замінивши початковий математичний вираз. Усі

запропоновані функції підібрані для плавної зміни значень у діапазоні аргументу від -15 до 15, що забезпечує ефективне навчання мережі.

Центральна частина роботи передбачає експериментальне дослідження впливу гіперпараметрів на процес навчання. Студент повинен варіювати кількість нейронів у прихованому шарі (N_neurons), швидкість навчання (learning_rate) та кількість епох (epochs) для досягнення оптимальної апроксимації. Рекомендується почати з базових значень (20 нейронів, швидкість навчання 0.01, 1000 епох) та систематично їх змінювати, спостерігаючи за результатами.

Аналіз результатів здійснюється на основі двох графіків, які автоматично генерує програма: графік процесу навчання (зміна метрик mae та loss) та графік апроксимації (порівняння навчальних точок з передбаченою кривою). Якісна апроксимація характеризується стабілізацією метрик втрат на низькому рівні та візуальним збігом апроксимованої кривої з навчальними даними. Студент повинен зберегти найкращі результати та проаналізувати їх у звіті.

Таблиця 14.3 – Варіанти індивідуальних завдань

Номер	Вираз
1	$y = 0.1 \cdot x^2$
2	$y = 0.01 \cdot x^3$
3	$y = 2 \cdot \sin(0.3 \cdot x)$
4	$y = 2 \cdot \cos(0.3 \cdot x)$
5	$y = 3 \cdot \tanh(0.2 \cdot x)$ $y = 0.3 \cdot e^{0.1 \cdot x}$
6	$y = 1.5 \cdot \ln(0.5 \cdot x + 8)$
7	$y = 0.3 \cdot \lvert x \rvert$
8	$y = 0.4 \cdot x \cdot \sin(0.2 \cdot x)$
9	$y = 0.05 \cdot x^2 \cdot \cos(0.25 \cdot x)$
10	$y = 2.5 \cdot \sin(0.04 \cdot x^2)$
11	$y = 3 \cdot e^{-0.015 \cdot x^2}$
12	$y = 0.25 \cdot x \cdot e^{-0.008 \cdot x^2}$
13	$y = 0.002 \cdot (x^4 - 20 \cdot x^2 + 50)$
14	$y = 3 \cdot \arctan(0.25 \cdot x)$
15	$y = 0.008 \cdot (x - 2)^3$
16	$y = 0.15 \cdot x \cdot \ln(\lvert x \rvert + 1.5)$
17	$y = 0.4 \cdot \sqrt{x^2 + 9}$
18	$y = 0.08 \cdot x^2 + 0.3 \cdot x + 1$
19	$y = 1.5 \cdot \max(0, x - 3)$
20	$y = 2 \cdot \min(6, x + 8)$

21	$y = 0.06 \cdot \text{sign}(x) \cdot x^2$
22	$y = 0.3 \cdot x \cdot \cos(0.2 \cdot x)$
23	$y = 0.0008 \cdot x^4 + 0.08 \cdot x^2$
24	$y = 2.5 \cdot \sinh(0.12 \cdot x)$
25	$y = 1.8 \cdot \cosh(0.08 \cdot x) - 1.8$
26	$y = 0.06 \cdot (x+3)^2 - 2$
27	$y = 0.25 \cdot x \cdot \sin(0.12 \cdot x) \cdot \cos(0.08 \cdot x)$
28	$y = 0.0003 \cdot x^5 + 0.015 \cdot x^3$
29	$y = 0.8 \cdot \sqrt{\lvert x-2 \rvert} \cdot \text{sign}(x-2)$
30	$y = x^2$

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється згідно з загальноприйнятими вимогами до навчальної документації: шрифт Times New Roman, розмір 12 pt, міжрядковий інтервал 1,5. Протокол складається з трьох обов'язкових частин, які розташовуються у наступній послідовності.

1. Вступна частина. У цьому розділі зазначаються тема та мета лабораторної роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета формулюється у відповідності до навчальних цілей дисципліни та вимог методичних рекомендацій. Також у цьому розділі подаються відповіді на ключові питання, передбачені завданням. Кожна відповідь має бути чіткою, лаконічною, ґрунтуватися на теоретичному матеріалі лабораторної роботи та демонструвати розуміння студентом ключових положень теми.

2. Основна частина (виконання лабораторного завдання).

У ході виконання лабораторної роботи студент реалізує програму, що використовує бібліотеки TensorFlow та Keras для побудови та навчання простої нейронної мережі, призначеної для апроксимації заданої математичної функції. Програма генерує синтетичний набір даних на основі індивідуального варіанта, будує модель з одним прихованим шаром, виконує її навчання, а потім порівнює передбачені значення з точними. Особливу увагу приділяється підготовці даних, вибору архітектури моделі, налаштуванню процесу навчання та візуалізації результатів.

У протоколі обов'язково мають бути наведені три ключові елементи:

– по-перше, завдання, у якому чітко записано індивідуальний варіант — аналітичний вираз функції, що апроксимується;

– по-друге, лістинг програми, що містить повний, добре структурований код на мові Python з необхідними коментарями, які пояснюють етапи генерації даних, побудови моделі, навчання, оцінки точності та візуалізації результатів;

– по-третє, результати виконання програми, представлені у вигляді графіка, на якому одночасно відображено точну функцію та апроксимацію, отриману нейронною мережею, а також, за потреби, скріншоти виводу консолі з показниками помилки або прогресу навчання.

3. Висновкова частина. У цьому розділі студент формулює висновки щодо знань, практичних умінь та професійних компетенцій, отриманих у процесі виконання роботи. Аналізуються труднощі, які виникли під час виконання завдання, та способи їх подолання. Особлива увага приділяється практичному значенню набутих навичок для майбутньої професійної діяльності у сфері програмування, аналізу даних, мережевих технологій, кібербезпеки або інших напрямів, пов’язаних із вивченням дисципліни «Python-програмування».

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.

2. Основи програмування. Python. Частина 1: підручник для студ. спеціальності 122 "Комп'ютерні науки", спеціалізації "Інформаційні технології в біології та медицині" / А. В. Яковенко; КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.

3. Ерік М. Пришвидшений курс Python / Маттес Ерік. – Київ: Видавництво Старого Лева, 2021. – 600 с.

4. Васильєв О. Програмування мовою Python / Олексій Васильєв. – Тернопіль: Навчальна книга Богдан, 2019. – 204 с.

5. Naomi Ceder The Quick Python Book 3rd Edition / Naomi Ceder. – NY: Manning Publications Co., 2018 – 432 p.

6. Kenneth A. Lambert Fundamentals of Python: first programs / Kenneth A. Lambert. – NY: Cengage Learning, 2018 – 476 p.

7. Mark L. Learning Python, 5th Edition / L. Mark – Sebastopol: O'Reilly Media, 2019. – 648 p.