**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ**  
**«ОПТИКО-МЕХАНІЧНИЙ ФАХОВИЙ КОЛЕДЖ КИЇВСЬКОГО**  
**НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКА»**  
Циклова комісія програмування та інформаційних технологій

**ЗВІТ**

**З НАВЧАЛЬНОЇ ПРАКТИКИ**

спеціальність *121 Інженерія програмного забезпечення*

освітньо-кваліфікаційний рівень *«Фаховий молодший бакалавр»*

|  |  |
| --- | --- |
| ОЦІНКА | Виконав:  студент 3-го курсу групи ІПЗ-31  *Барбара Владислав Андрійович*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис виконавця)  « \_» 20 року |
| (оцінка цифрою та прописом)  Керівник практики від коледжу: |
| (підпис)  « \_» 20 року |

КИЇВ — 2025

ЗАТВЕРДЖУЮ

Директор ВСП «ОМФК КНУ

ім. Тараса Шевченка»

\_\_\_\_\_\_\_\_\_\_\_Борис ГАПРИНДАШВІЛІ

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023р

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на період *Навчальної* практики

(навчальної, технологічної, виробничої)

(П І.Б студента )

Спеціальність *121 Інженерія програмного забезпечення* Група *ІПЗ-*

|  |  |  |
| --- | --- | --- |
|  | **Питання для вивчення** | **Відмітки про виконання** |
| 1 | Вступне заняття. Вивчення питань охорони праці. Ознайомлення з електронною бібліотекою коледжу. Вивчення правил оформлення звітної документації. |  |
| 2 | Знайомство з мовою програмування Python. Історія мови, особливості мови, сфери застосування мови |  |
| 3 | Основи мови Python. Синтаксис мови, середовище виконання, запуск програм |  |
| 4 | Змінні та літерали, типи даних. Вбудовані типи даних, ініціалізація змінних, змінювані і незмінювані типи |
| 5 | Програмування лінійних та розгалужених обчислювальних процесів |  |
| 6 | Програмування циклічних обчислювальних процесів |  |
| 7 | Типи даних визначені користувачем, магічні методи, виключення (Exceptions) та їх обробка |  |
| 8 | Вбудовані структури даних, складні типи |  |
| 9 | Функції |  |
| 10 | Потокове введення, виведення та обробка даних |  |
| 11 | Побудова проекту, модулі, пакети |  |
| 12 | Робота з файловою системою, оператор with, контекстний менеджер |  |
| 13 | Робота з протоколом HTTP, бібліотека requests |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Голова циклової комісії |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |
| Керівник практики від коледжу |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |

**Вступ**

Перелік пройдених тем під час практики:

1. Вступне заняття. Вивчення питань охорони праці. Ознайомлення з електронною бібліотекою коледжу. Вивчення правил оформлення звітної документації.
2. Знайомство з мовою програмування Python. Історія мови, особливості мови, сфери застосування мови
3. Основи мови Python. Синтаксис мови, середовище виконання, запуск програм
4. Змінні та літерали, типи даних. Вбудовані типи даних, ініціалізація змінних, змінювані і незмінювані типи
5. Програмування лінійних та розгалужених обчислювальних процесів
6. Програмування циклічних обчислювальних процесів
7. Типи даних визначені користувачем, магічні методи, виключення (Exceptions) та їх обробка
8. Вбудовані структури даних, складні типи
9. Функції
10. Потокове введення, виведення та обробка даних
11. Побудова проекту, модулі, пакети
12. Робота з файловою системою, оператор with, контекстний менеджер
13. Робота з протоколом HTTP, бібліотека requests
14. Створення HTTP API, бібліотека FastAPI

**1. Теоретичні відомості**

***1.1 Вступне заняття.***

Під час вступного заняття ми отримали фундаментальні знання з охорони праці та інструктаж з роботи за комп’ютерною технікою, що в подальшому допомогло ефективно та безпечно виконувати завдання під час проходження навчальної практики, також ми отримали список літератури до якої можна було в будь-який час звернутись та поглиблювати свої знання під час проходження всієї практики. Однією з найважливіших тем вступного заняття було розглядання правил оформлення звітньої документації, для зафіксування пройденої роботи під час навчальної практики.

***1.2******Знайомство з мовою програмування Python. Історія мови, особливості мови, сфери застосування мови****.*

Під час знайомства з мовою програмування Python ми занурюємось у цікаву історію її розвитку та особливості, які роблять її однією з найпопулярніших мов програмування на сьогодні. Python був створений Гвідо ван Россумом у кінці 1980-х років і вперше випущений у 1991 році. Відтоді мова швидко набувала популярності завдяки своїй простоті, гнучкості та можливостям, що роблять її чудовим вибором для багатьох сфер програмування.

Python відрізняється своєю динамічною типізацією, що означає, що типи змінних визначаються під час виконання програми, а не на етапі компіляції, як у мовах зі статичною типізацією. Це дає значну гнучкість у написанні коду, оскільки немає необхідності вказувати тип кожної змінної або параметра функції. Однак це також може призводити до помилок, які проявляються лише під час виконання, оскільки типи не перевіряються на етапі компіляції.

Мова Python є прикладом нестрогого типізування, що означає, що вона не накладає жорстких обмежень на типи змінних. Наприклад, можна змінити тип змінної в середині виконання програми, що підвищує зручність та швидкість розробки, хоча це може призвести до певних труднощів при масштабуванні великих проектів, де сувора типізація допомагає виявляти помилки на ранніх етапах розробки.

Python знаходить застосування в багатьох сферах програмування, що робить його універсальним інструментом для різноманітних завдань. Однією з основних сфер, де активно використовують Python, є Web-розробка. Завдяки таким фреймворкам, як Django та Flask, Python є популярним вибором для створення серверної частини веб-застосунків, оскільки він пропонує простоту написання коду і великі можливості для інтеграції з іншими технологіями.

Ще однією важливою сферою є IoT. Завдяки своїй простоті та ефективності Python часто використовується для програмування мікроконтролерів та мікропроцесорних пристроїв, таких як Raspberry Pi. Python дозволяє швидко розробляти прототипи та автоматизувати задачі, що є критичним для розвитку Інтернету речей.

Не менш популярною є сфера Data Science — Python є основною мовою для аналізу даних, завдяки своїм потужним бібліотекам, таким як NumPy, pandas, Matplotlib, і Scikit-learn. Python дозволяє ефективно обробляти великі обсяги даних, проводити статистичні аналізи та розробляти моделі машинного навчання.

У системному програмуванні Python також використовується для автоматизації задач, написання скриптів та адміністрування систем. Багато DevOps-інженерів використовують Python для автоматизації процесів розгортання та моніторингу систем, що робить його важливим інструментом у сфері DevOps.

Завдяки своїй універсальності, зручності та потужним бібліотекам Python стає оптимальним вибором для багатьох різних завдань, від веб-розробки до аналізу даних та роботи з Інтернетом речей. Його проста синтаксична структура дозволяє зосередитися на вирішенні конкретних задач, замість того, щоб витрачати час на вивчення складних конструкцій мови. Це робить Python ідеальним вибором для початківців та досвідчених розробників, які прагнуть досягти швидких результатів у своїй роботі.

***1.3 Основи мови Python. Синтаксис мови, середовище виконання, запуск програм.***

Під час вивчення основ мови на занятті було розглянуто її *алфавіт, лексеми, вирази,* а також ключові терміни, пов’язані з середовищем виконання.

В основі Python лежить набір символів, який складається з літер латинського алфавіту, цифр та спеціальних знаків. На основі цих символів формуються лексеми, до яких належать ключові слова, ідентифікатори, літерали, оператори та роздільники. Важливо зазначити, що Python є чутливою до регістру мовою, тому, наприклад, Variable і variable — це різні ідентифікатори. Вирази в Python будуються за допомогою змінних, операторів і функцій, дозволяючи виконувати обчислення та обробку даних у зручній формі.

Однією з ключових особливостей мови є її інтерпретована природа. Код Python виконується рядок за рядком без необхідності попередньої компіляції, що значно спрощує процес налагодження та тестування. Водночас це відрізняє Python від мов, що використовують транслятори (компілятори), які перед виконанням програми перетворюють вихідний код у машинний.

Ще одним важливим аспектом є кросплатформеність Python. Оскільки інтерпретатори мови доступні для різних операційних систем, написані програми можна без змін запускати на Windows, macOS і Linux. Це робить Python універсальним інструментом для розробників, які працюють у різних середовищах.

Під час вивчення теми також було розглянуто CPython — стандартну реалізацію Python, написану мовою C. Саме вона використовується найчастіше і підтримується Python Software Foundation. Окрім CPython, існують й альтернативні інтерпретатори, такі як PyPy, що забезпечує швидше виконання завдяки технології JIT-компіляції, Jython, який дозволяє інтегрувати Python-код із Java, та IronPython, призначений для платформи .NET.

Опанування цих основ закладає базу для подальшого вивчення Python і його застосування у програмуванні. Розуміння синтаксису, принципів роботи інтерпретатора та особливостей виконання коду допоможе не лише у навчанні, а й у подальшій професійній діяльності, особливо в розробці програмного забезпечення, аналізі даних та автоматизації процесів.

***1.4 Змінні та літерали, типи даних. Вбудовані типи даних, ініціалізація змінних, змінювані і незмінювані типи.***

Під час заняття ми багато часу приділили вивченню теми змінних та літералів, було детально розглянуто, як Python працює з даними, які існують вбудовані типи, як відбувається ініціалізація змінних, а також різницю між змінюваними та незмінюваними структурами. Важливим аспектом стало ознайомлення з концепцією строгої динамічної типізації, що є характерною особливістю Python і впливає на спосіб роботи з об'єктами в пам’яті.

Python підтримує широкий набір вбудованих типів даних, серед яких числові (int, float, complex), логічні (bool), текстові (str), колекційні (list, tuple, set, dict) та спеціальні (None). Кожен об'єкт у Python має певний тип, який визначає, як можна з ним працювати. При цьому мова використовує динамічну типізацію: змінним не потрібно вказувати тип під час оголошення, він визначається автоматично на основі присвоєного значення. Наприклад, x = 5 автоматично створює змінну x, яка є цілочисельною (int).

Принцип строгої типізації означає, що операції між несумісними типами спричинять помилку. Наприклад, 5 + "hello" призведе до TypeError, оскільки числа та рядки не можна безпосередньо додавати. Водночас Python дозволяє явне перетворення типів, що дає змогу адаптувати дані відповідно до потреб.

Окрему увагу було приділено поняттю змінюваності та незмінюваності типів. Незмінювані (immutable) типи, такі як int, float, str, tuple, після створення не можуть бути змінені. Натомість змінювані (mutable) типи, такі як list, dict, set, дозволяють змінювати свої елементи без створення нового об'єкта в пам’яті. Це має значення при передачі змінних у функції: незмінювані передаються за значенням, а змінювані – за посиланням.

Вивчаючи роботу чисел у Python, було розглянуто їхнє представлення в пам'яті та особливості обчислень. Цілі числа (int) можуть бути довільно великі, оскільки Python динамічно змінює їх розмір. Дійсні числа (float) використовують стандарт IEEE 754, що означає можливість втрати точності під час обчислень. Наприклад, 0.1 + 0.2 не дасть рівно 0.3, що пов’язано з особливостями зберігання чисел у двійковій системі.

Було також досліджено пріоритетність операцій у Python. Як і в математиці, множення та ділення виконуються перед додаванням і відніманням, але дужки можуть змінювати порядок обчислень. Використання оператора \*\* (піднесення до степеня) має вищий пріоритет, ніж множення, а порівняння (==, !=, <, >) відбуваються після арифметичних операцій.

Щоб спростити виконання математичних операцій, Python надає вбудовані функції, такі як abs() для модуля числа, round() для округлення, pow() для піднесення до степеня. Додаткові можливості надає модуль math, що містить функції для тригонометрії (sin(), cos()), логарифмів (log()), округлення (floor(), ceil()) та інших обчислень.

Розуміння змінних, типів даних і математичних операцій має важливе значення для написання ефективного коду. Це дозволяє правильно працювати з пам’яттю, уникати помилок типізації та вибирати оптимальні структури для зберігання даних. У практичній діяльності ці знання необхідні для створення алгоритмів, обробки інформації та оптимізації роботи програм.

**2. Програмування лінійних та розгалужених обчислювальних процесів.**

У процесі вивчення теми програмування лінійних та розгалужених обчислювальних процесів було розглянуто принципи побудови алгоритмів, що виконуються послідовно або з розгалуженням на основі певних умов. Одним із ключових аспектів стала робота з логічними операторами та конструкціями умовного виконання коду.

Python використовує три основні логічні оператори: and, or і not. Оператор and повертає True, якщо обидва операнди істинні, or повертає True, якщо хоча б один із них істинний, а not інвертує значення виразу. Наприклад, вираз (x > 0 and y < 10) буде істинним, якщо x більше нуля, а y менше десяти. Такі вирази дозволяють створювати складні логічні умови для керування виконанням програми.

Основним інструментом реалізації розгалужень є конструкція if-elif-else, яка дозволяє виконувати різні ділянки коду залежно від виконання певних умов.

Приклад її використання:

*if temperature > 30:*

*print("Дуже спекотно")*

*elif temperature > 20:*

*print("Тепла погода")*

*else:*

*print("Прохолодно")*

Тут важливо враховувати синтаксичну особливість Python — використання відступів замість фігурних дужок, як у деяких інших мовах. Відступи визначають блоки коду, і неправильне їх використання може призвести до помилок.

Вивчення цієї теми дозволяє ефективно керувати виконанням програми, вибудовувати логіку прийняття рішень і створювати більш гнучкі алгоритми. У практичній діяльності ці навички важливі для розробки будь-яких програм, що потребують перевірки умов, аналізу введених даних або обробки різних сценаріїв виконання.

***2.1 Програмування циклічних обчислювальних процесів.***

Вивчення циклічних обчислювальних процесів у Python дозволяє створювати алгоритми, що повторюють певні дії багаторазово, змінюючи значення змінних або виконуючи обробку послідовностей. Основними конструкціями для реалізації циклів є for і while, кожна з яких має свої особливості використання.

Цикл for застосовується для ітерації по послідовностям, таким як списки, рядки чи діапазони чисел. Його особливість полягає в тому, що він проходить по кожному елементу ітерабельного об’єкта, автоматично отримуючи значення без необхідності змінювати лічильник вручну.

Цикл while виконується доти, доки виконується певна умова. Він є більш гнучким, оскільки дозволяє виконувати повторення доти, доки виконується деяка логічна умова, проте потребує контролю за зміною змінних, щоб уникнути нескінченних циклів.

Важливою частиною цієї теми є ознайомлення з поняттями ітераторів та ітерабельних об’єктів. Ітерабельними називаються об'єкти, які можна перебирати в циклі for, наприклад, списки (list), кортежі (tuple), рядки (str) та словники (dict). Ітератори – це об'єкти, що реалізують методи \_\_iter\_\_() і \_\_next\_\_(), дозволяючи послідовно отримувати елементи. Окрім ітераторів, у Python існує механізм генераторів, який дозволяє створювати функції, що повертають значення поступово, зберігаючи свій стан між викликами за допомогою ключового слова yield. Це особливо корисно при роботі з великими обсягами даних, оскільки генератор не зберігає всі значення в пам’яті, а створює їх на льоту. Головна перевага генераторів – ефективність роботи з великими послідовностями, адже вони не потребують створення та збереження всіх елементів у пам’яті, як це роблять списки.

Застосування ітераторів і генераторів має важливе значення в обробці потокових даних, аналізі великих наборів інформації, реалізації механізмів відкладеного виконання та оптимізації програм, що працюють з ресурсомісткими операціями.

***2.2 Типи даних визначені користувачем, магічні методи, виключення (Exceptions) та їх обробка.***

Вивчаючи створення користувацьких типів даних, було детально розглянуто концепцію класів у Python, які дозволяють визначати власні структури даних і налаштовувати їх поведінку. Використання класів дозволяє створювати об’єкти з власними атрибутами та методами, що значно розширює можливості роботи з даними.

Кожен клас може містити спеціальні (або магічні) методи, які визначають поведінку об'єктів у різних ситуаціях. Найважливішим серед них є \_\_init\_\_, який виконується під час створення нового об’єкта та ініціалізує його параметри.

Методи \_\_str\_\_ і \_\_repr\_\_ визначають текстове представлення об'єкта. \_\_str\_\_ використовується для перетворення об’єкта у рядок у випадках, коли викликається print(), а \_\_repr\_\_ – для технічного представлення об'єкта, яке допомагає при налагодженні коду. Метод \_\_eq\_\_ дозволяє налаштовувати порівняння об'єктів. Наприклад, якщо потрібно порівнювати двох людей за іменем та віком. Якщо клас використовується як ключ у словнику або елемент множини, він повинен мати визначений метод \_\_hash\_\_, який повертає унікальне числове значення. Окрім створення класів, було розглянуто обробку виключень у Python, яка дозволяє перехоплювати помилки та запобігати аварійному завершенню програми. Основною конструкцією для цього є try-except, яка дає змогу обробити помилки та виконати альтернативний код у разі їх виникнення. Конструкція else виконується лише якщо у блоці try не виникло помилок, а finally виконується завжди, незалежно від того, сталася помилка чи ні.

Також розглянуто можливість створення власних виключень через наслідування від Exception. Це дозволяє визначати специфічні помилки для конкретних випадків у програмі. Вивчення цієї теми дозволяє створювати власні об'єкти, контролювати їхню поведінку, а також керувати винятковими ситуаціями, що значно підвищує надійність програмного забезпечення та робить його більш гнучким.

***2.3 Вбудовані структури даних, складні типи.***

Вивчення вбудованих структур даних у Python є важливою частиною роботи з мовою, оскільки правильний вибір структури може значно впливати на продуктивність програми. Основними структурами, що розглядаються, є списки, кортежі, словники, множини та строки.

Списки (list) у Python є динамічними масивами, які можуть змінювати свій розмір. Вони реалізовані як масиви PyObject, що означає, що вони зберігають посилання на об’єкти, а не самі значення. Це дозволяє працювати з елементами різних типів. Операції доступу за індексом виконуються за O(1), але вставка або видалення елементів у середину списку вимагає зсуву елементів (O(n)).

*numbers = [1, 2, 3, 4]*

*numbers.append(5) # Додає елемент у кінець списку*

*numbers.insert(2, 10) # Вставляє число 10 на позицію 2*

Кортежі (tuple) подібні до списків, але є незмінними. Це означає, що їх не можна змінювати після створення, що робить їх швидшими і зручнішими для використання в якості ключів у словниках та множинах.

*coordinates = (10, 20)*

Словники (dict) реалізовані як хеш-таблиці, що забезпечує швидкий доступ до значень за ключами (O(1)). Вони зберігають пари ключ-значення, де ключами можуть бути лише hashable об’єкти, тобто ті, що мають незмінне значення хешу.

Множини (set) використовують хеш-таблиці, що дозволяє виконувати операції перевірки належності за O(1). Вони не містять повторюваних елементів та можуть використовувати операції об'єднання, перетину тощо.

Окремо розглядається концепція hashable типів. Об'єкт вважається hashable, якщо його хеш не змінюється після створення. Наприклад, int, str та tuple – hashable, тому можуть використовуватися як ключі у словниках, тоді як list – ні.

Також було розглянуто концепцію зв’язаних списків, що відрізняються від масивів тим, що кожен елемент містить посилання на наступний. Це забезпечує швидке видалення або вставку (O(1)), але ускладнює довільний доступ (O(n)).

Правильний вибір структури даних залежить від конкретної задачі. Якщо важливий швидкий довільний доступ – краще використовувати список. Якщо потрібно зберігати унікальні значення – множину. Для асоціативного зберігання – словник, а коли необхідна незмінність – кортеж.

**3. Функції.**

Під час заняття було розглянуто основи роботи з функціями у Python. Було опрацьовано синтаксис визначення функцій за допомогою ключового слова def, принципи передачі параметрів, а також правила локалізації змінних. Окремо досліджувалася можливість використання функцій як об’єктів, зокрема їх передача як аргументів до інших функцій та повернення як результату.

Також було розглянуто анонімні функції (lambda), їхній синтаксис і можливі сценарії застосування у випадках, коли потрібно створити компактну функцію без повного визначення через def. Крім цього, значна увага була приділена рекурсії — її правилам, перевагам та можливим обмеженням у Python. Було проведено практичні випробування рекурсивних викликів і визначено ситуації, коли рекурсія є зручним інструментом, а коли її краще замінити на ітеративний підхід.

Засвоєні знання сприятимуть подальшому розвитку алгоритмічного мислення та оптимізації коду. У професійній діяльності навички роботи з функціями допоможуть покращити структуризацію програм, забезпечити повторне використання коду та підвищити його читабельність, що є важливим для розробки складних програмних рішень.

***3.1 Потокове введення, виведення та обробка даних.***

Під час опрацювання теми було досліджено принципи потокового введення та виведення у Python, зокрема використання модулів sys.stdin, sys.stdout та sys.stderr. Вивчено, як ці об’єкти дозволяють працювати зі стандартними потоками: sys.stdin використовується для отримання даних із введення (наприклад, з командного рядка), sys.stdout відповідає за виведення даних, а sys.stderr – за обробку помилок. Також було протестовано перенаправлення потоків у файли та інші програми для автоматизації обробки даних у реальному часі.

Додатково розглянуто поширені формати збереження та обробки даних: JSON Lines (NDJSON), Protocol Buffers (Protobuf), Parquet та CSV. Було визначено їхні основні відмінності та оптимальні сфери використання. Наприклад, CSVпідходить для роботи з табличними даними, JSON Lines дозволяє зберігати потокові дані построчно, що спрощує обробку великих обсягів інформації, Parquet забезпечує ефективне стиснення та швидкий доступ до структурованих даних, що робить його придатним для аналітики, а Protobuf є компактним і високопродуктивним форматом для передачі структурованих даних у розподілених системах. Виконано практичне тестування роботи з цими форматами у Python за допомогою бібліотек pandas, json, pyarrow та protobuf.

Окремо досліджено змінні оточення, які дозволяють зберігати конфігураційні параметри програми поза вихідним кодом, що є критично важливим для розгортання застосунків у різних середовищах. Було розглянуто способи їхнього зчитування через os.environ у Python, а також їхнє використання для налаштування бази даних, API-ключів та інших змінних без модифікації коду.

***3.2 Побудова проекту, модулі, пакети.***

Під час вивчення теми "Побудова проекту, модулі, пакети" було детально ознайомлено з концепціями модулів та пакетів у Python, а також із принципами організації структурованих проектів. Модулі в Python представляють собою файли з розширенням .py, які містять визначення функцій, класів та змінних. Вони дозволяють логічно розділяти код на окремі частини, що сприяє його повторному використанню та полегшує підтримку. Під час імпорту модуля в програму використовується ключове слово import, що забезпечує доступ до його вмісту.

Пакети, у свою чергу, є каталогами, які містять модулі та спеціальний файл \_\_init\_\_.py. Вони служать для об'єднання пов'язаних модулів у єдину структуру, створюючи простір імен для уникнення конфліктів між модулями з однаковими назвами. Це дозволяє будувати ієрархічні структури модулів та організовувати код у масштабних проектах.

Особливу увагу було приділено правилам організації проекту та іменуванню файлів і директорій. Рекомендується використовувати зрозумілі та описові назви для модулів і пакетів, дотримуючись стандартів PEP 8. Структура проекту повинна бути логічною та інтуїтивно зрозумілою, що спрощує навігацію та підтримку коду.

Також було досліджено роботу з менеджером пакетів pip, який є стандартним інструментом для встановлення та управління бібліотеками Python. За допомогою pip можна легко знаходити, встановлювати та оновлювати необхідні пакети з офіційного репозиторію Python Package Index (PyPI). Це значно спрощує процес інтеграції сторонніх бібліотек у проект та забезпечує їх актуальність.

На завершення було розглянуто процес створення власних пакетів. Це передбачає організацію коду у вигляді модулів, об'єднаних у пакети, з відповідною структурою директорій та файлів. Створення власних пакетів дозволяє розробляти багаторазово використовувані компоненти, які можна легко інтегрувати в інші проекти або розповсюджувати через PyPI.

***4 Робота з файловою системою, оператор with, контекстний менеджер.***

Робота з файловою системою є важливою частиною розробки програмного забезпечення, оскільки файли часто використовуються для зберігання та обміну даними. У Python робота з файлами здійснюється через вбудовані функції та методи, що дозволяють ефективно відкривати, читати, записувати та закривати файли.

Один із основних аспектів роботи з файлами в Python — це правильне управління ресурсами. Після завершення операцій з файлом важливо закрити його, щоб не виникали витоки пам'яті або блокування файлу. Замість того, щоб вручну викликати метод close(), можна використовувати оператор with, який автоматично закриває файл після завершення роботи з ним. Це важливо, оскільки навіть якщо в процесі роботи виникне помилка, файл буде закритий належним чином.

Використання оператора with має кілька переваг. Він дозволяє автоматично управляти ресурсами, тобто відкривати та закривати файли без необхідності вручну викликати метод close(). Це значно знижує ймовірність виникнення помилок, пов'язаних із неправильним закриттям файлів.

Окрім стандартних контекстних менеджерів для роботи з файлами, Python дозволяє створювати власні контекстні менеджери, реалізуючи два методи: \_\_enter\_\_ та \_\_exit\_\_. Метод \_\_enter\_\_ виконується перед початком виконання блоку коду в контексті, а метод \_\_exit\_\_ — після завершення виконання цього блоку. Це дає можливість налаштовувати поведінку під час входу та виходу з контексту.

Контекстні менеджери можуть бути дуже корисними, коли потрібно управляти ресурсами, такими як файли, мережеві з’єднання або бази даних, що вимагають закриття або звільнення ресурсів після використання.

Використання контекстних менеджерів — це важливий інструмент для забезпечення коректного управління ресурсами в програмі, що дозволяє підвищити її надійність і знизити ризик помилок.

***4.1 Робота з протоколом HTTP, бібліотека requests.***

У процесі роботи з протоколом HTTP ми зосередилися на його основних методах, які є критично важливими для взаємодії між клієнтами та серверами. Один з найбільш поширених методів — це GET, що використовується для отримання ресурсів із сервера. Наприклад, запит на отримання даних з віддаленого ресурсу можна здійснити за допомогою методу requests.get(), що дозволяє отримати текстову відповідь від сервера. Важливою особливістю є можливість безпосередньо працювати з відповіддю, використовуючи атрибути, наприклад, response.text.

Іншим важливим методом є POST, що застосовується для надсилання даних на сервер. Використовуючи бібліотеку requests, можна зручно передавати дані, як, наприклад, заповнення форми на веб-сайті. Для цього дані передаються через параметр data, і бібліотека автоматично перетворює їх на потрібний формат. Важливою частиною цієї роботи є правильне налаштування заголовків, таких як Content-Type, що вказує серверу, що саме передається у запиті.

Крім того, інтерфейс бібліотеки дозволяє ефективно використовувати методи POST, PUT, DELETEдля взаємодії з сервером. Наприклад, при оновленні або видаленні існуючих ресурсів ці методи дозволяють виконувати зміни на сервері за допомогою відповідних запитів. Такі операції передбачають не тільки передачу даних, але й важливу взаємодію з кодами статусів, які сигналізують про результат запиту, наприклад, код 200 свідчить про успішне виконання операції.

Надзвичайно важливим моментом є робота з параметрами запиту, які можна передавати через URL. Бібліотека requests підтримує параметр params, що дозволяє зручно фільтрувати запити на сервері. Це може бути корисним для отримання конкретної інформації або для здійснення пошукових запитів, де потрібно враховувати різні умови.

Особливу увагу варто приділити роботі з сесіями та cookies. Використання сесій дозволяє зберігати параметри між кількома запитами, що, наприклад, важливо при авторизації або збереженні певного стану між запитами. Завдяки цьому, при надсиланні наступних запитів, користувач може зберігати необхідні налаштування або ідентифікаційні дані без необхідності їх кожного разу вводити. Цей механізм дозволяє автоматично передавати cookies між запитами, що є важливим для більш складних інтерактивних веб-застосунків.

Під час роботи з статусами HTTP необхідно перевіряти відповідь на кожен запит. Це дозволяє визначити, чи було запитано ресурси успішно, чи виникли помилки. Наприклад, код статусу 404 свідчить про те, що ресурс не знайдений, тоді як 200 означає успішне виконання запиту. Завдяки можливості перевіряти статуси, можна ефективно обробляти помилки та виводити корисні повідомлення.

В кінці, не менш важливим є вміння працювати з форматом JSON, який є популярним для передачі даних у веб-аплікаціях. Бібліотека requests дозволяє автоматично перетворювати JSON-відповіді сервера в Python-об'єкти, що значно спрощує обробку отриманих даних.

Таким чином, бібліотека requests надає зручний інтерфейс для роботи з HTTP-запитами та дозволяє ефективно взаємодіяти з веб-сервісами. Її можливості, такі як налаштування параметрів запиту, робота з сесіями та cookies, обробка JSON-відповідей і перевірка статусів запитів, роблять процес розробки веб-застосунків значно простішим і ефективнішим.

**2. Опис постановки задачі.**

Під час виконання практики викладачами були надані матеріали та додаткові завдання для поглиблення наших знань з мови програмування Python. А саме:

**Завдання 1.** Написати функцію, яка: приймає на вхід строку і повертає кортеж з трьома елементами, в яких перший елемент має тип str і включає всі голосні літери з вхідної строки (зберігаючи порядок). Другий елемент - int - кількість приголосних літер, а останній - всі приголосні літери з вхідної строки (зберігаючи порядок).

Це завдання має навчальну цінність, оскільки розвиває ключові навички роботи з текстовими даними, включаючи:

* **Обробку рядків**: вміння витягувати певні символи за заданими критеріями.
* **Роботу з колекціями**: використання списків та кортежів для зберігання та обробки даних.
* **Перевірку символів**: визначення голосних та приголосних літер.
* **Структуроване збереження інформації**: застосування кортежів для збереження незмінних наборів даних.

### **Практичне застосування**

Подібні алгоритми корисні у багатьох сферах, зокрема:

* **Обробка природної мови (NLP)**: аналіз текстів, фільтрація даних.
* **Лінгвістичний аналіз**: підрахунок літер у словах, перевірка орфографії.
* **Розробка пошукових систем**: вилучення ключових слів із тексту.

Цей підхід може бути використаний у створенні **спам-фільтрів**, **аналізаторів тексту** та **систем автозаповнення**.

Завдання 2 Пропонується список задач для покращення навичок з програмування та вивчення Python зокрема.

Решетом Ератосфена називається процедура, при якiй спочатку записують послiдовнi числа від 2 до n. Залишивши число 2, викреслюють кожне друге. Переходять до наступного не викреслення числа. Це 3. Його залишають, а викреслюють кожне третє. Дiють так до тих пiр, поки не доберуться до кінця послiдовностi. Побудуйте решето для першої тисячі чисел.

Це завдання зосереджене на **обробці рядкових даних**, що є однією з ключових тем у програмуванні. Робота з текстом включає **аналіз символів, їхню класифікацію та структурування отриманої інформації**.

**Отримати рядок як вхідні дані.**

**Виділити з нього голосні літери та сформувати з них рядок.**

**Підрахувати кількість приголосних літер.**

**Створити рядок із приголосними літерами, зберігаючи їх порядок.**

**Оформити результати у вигляді кортежу з трьох елементів:**

Перший елемент (str) — голосні літери у вихідному порядку.Другий елемент (int) — кількість приголосних літер.Третій елемент (str) —приголосні літери у вихідному **Робота з текстовими даними**: Уміння виділяти символи за певними критеріями є важливим для обробки текстів. **Використання кортежів**: Це тренує навички роботи зі **структурами даних**, де важлива незмінність інформації. **Алгоритми та фільтрація даних**: Завдання вчить аналізувати символи, використовуючи **цикли, умови та спискові вирази**. **Практичне застосування** **Обробка природної мови (NLP)** – виділення потрібних літер або слів у текстах. **Фільтрація повідомлень** – алгоритми для розпізнавання ключових слів.**Створення пошукових систем** – аналіз текстових запитів для пошуку. **Лінгвістичний аналіз** – підрахунок кількості голосних та приголосних у словах.

**Завдання 3 Теми для розроблення програмного застосунку**

Повідомлення про повітряну тривогу в інший канал комунікації за вибором (Discord)

### **Що дає це завдання? (Отримані навички)**

1. **Робота з API:**
   1. Вміння підключатися до зовнішніх API (Telegram API, API для сповіщень про тривогу, Discord API).
   2. Обробка HTTP-запитів (GET, POST) для отримання та надсилання даних.
2. **Розробка ботів:**
   1. Створення та налаштування Discord-бота.
   2. Взаємодія бота з користувачами та каналами.
3. **Програмування (Python або JavaScript):**
   1. Робота з бібліотеками для роботи з API (наприклад, requests, discord.py, telethon).
   2. Обробка даних (JSON), робота з текстом та подіями в реальному часі.
4. **Автоматизація процесів:**
   1. Налаштування циклів для постійного моніторингу подій.
   2. Реалізація логіки для перевірки нових повідомлень.
5. **Основи DevOps:**
   1. Запуск скриптів на сервері для цілодобової роботи.
   2. Налаштування логування та моніторингу роботи бота.
6. **Обробка реального часу:**
   1. Оптимізація роботи для швидкого реагування на важливі події.
   2. Забезпечення надійної доставки критично важливої інформації.

### **Що використовується в завданні? (Інструменти та технології)**

1. **Мови програмування:**
   1. **Python** (бібліотеки: discord.py, requests, aiohttp, telethon для Telegram).
   2. Або **JavaScript** (бібліотека discord.js, axios для HTTP-запитів).
2. **API та інтеграції:**
   1. **Discord API** — для надсилання повідомлень у канал.
   2. **Telegram API** або інші джерела сповіщень.
3. **Системи для автоматизації:**
   1. Використання **cron** (Linux) або планувальників завдань для регулярного запуску скриптів.
   2. **Docker** (за потреби) для зручного розгортання.
4. **Хостинг/Сервер:**
   1. VPS, сервер Raspberry Pi або навіть хмарні рішення (Heroku, AWS, DigitalOcean) для безперервної роботи.
5. **Інструменти для тестування:**
   1. **Postman** для перевірки API-запитів.
   2. Консоль для налагодження коду.

**3. Розроблена програма та виконані завдання.**

***Завдання 1.***

def process\_string(s: str) -> tuple[str, int, str]:

vowels = "aeiouAEIOU"

vowel\_part = "".join([char for char in s if char in vowels])

consonant\_part = "".join([char for char in s if char.isalpha() and char not in vowels]) consonant\_count = len(consonant\_part)

return vowel\_part, consonant\_count, consonant\_part

input\_string = input("Введіть рядок: ")

result = process\_string(input\_string)

print(result)

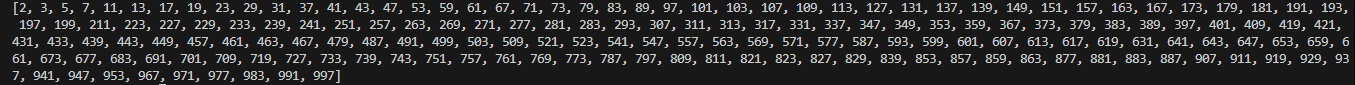


***Завдання 2.***

def sieve\_eratosthenes(n: int) -> list[int]: primes = [True] \* (n + 1) primes[0] = primes[1] = False # 0 і 1 не є простими числами

for i in range(2, int(n\*\*0.5) + 1):  
 if primes[i]:  
 for j in range(i \* i, n + 1, i):  
 primes[j] = False  
  
 return [i for i, is\_prime in enumerate(primes) if is\_prime]  
 prime\_numbers = sieve\_eratosthenes(1000)

print(prime\_numbers)

***Завдання 3***

import discord

import requests

import asyncio

TOKEN = "MTMzNzg3OTA4OTgxNzU4MzYyOA.GnYiF6.l1uYo38kWcCknj0wKSrl-XGuNGJBshaQJVduj0"

CHANNEL\_ID = 1337906608075964541

ALERTS\_API\_URL = "<http://ubilling.net.ua/aerialalerts/>"

MAP\_URL = "<https://ubilling.net.ua/aerialalerts/?map=webp>"

intents = discord.Intents.default()

client = discord.Client(intents=intents)

previous\_alerts = set()

alert\_message = None

async def check\_air\_alerts():

"""Перевіряє API тривог і оновлює повідомлення в Discord."""

global previous\_alerts, alert\_message

await client.wait\_until\_ready()

channel = await client.fetch\_channel(CHANNEL\_ID)

while True:

try:

response = requests.get(ALERTS\_API\_URL)

data = response.json()

current\_alerts = {region for region, info in data["states"].items() if info["alertnow"]}

if current\_alerts != previous\_alerts:

if current\_alerts:

alert\_text = "\n".join([f"🔴 {region}" for region in current\_alerts])

message\_text = f"🚨 \*\*АКТУАЛЬНІ ПОВІТРЯНІ ТРИВОГИ\*\* 🚨\n\n🗺 Карта тривог: {MAP\_URL}\n\n{alert\_text}"

else:

message\_text = f"✅ \*\*Повітряна тривога скасована у всіх областях!\*\*\n\n🗺 Карта: {MAP\_URL}"

if not alert\_message:

alert\_message = await channel.send(message\_text)

else:

await alert\_message.edit(content=message\_text)

previous\_alerts = current\_alerts

except Exception as e:

print(f"Помилка при отриманні даних: {e}")

await asyncio.sleep(60)

@client.event

async def on\_ready():

print(f"✅ Бот {client.user} запущений!")

client.loop.create\_task(check\_air\_alerts())

client.run(TOKEN)

