**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  (оцінка цифрою та прописом)  Керівник практики від коледжу:  (підпис)  « \_» 20 року | Виконав:  студентка 3-го курсу групи ІПЗ-32  *Глушко Анна Федорівна*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис виконавця)  « \_» 20 року |

КИЇВ — 2025

ЗАТВЕРДЖУЮ

Директор ВСП «ОМФК КНУ

ім. Тараса Шевченка»

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

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025р

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на період *Навчальної* практики

(навчальної, технологічної, виробничої)

*Глушко Анна Федорівна*

(П І.Б студента )

Спеціальність *121 Інженерія програмного забезпечення* Група *ІПЗ-32*

|  |  |  |
| --- | --- | --- |
|  | **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) |

**Зміст**

**[ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ](#_Toc189510033)** [2](#_Toc189510033)

**[Вступ](#_Toc189510034)** [3](#_Toc189510034)

**[1](#_Toc189510035)****[Теоретичні відомості](#_Toc189510035)** [4](#_Toc189510035)

**[1.1 Вступне заняття](#_Toc189510036)** [4](#_Toc189510036)

**[1.3 Основи мови Python. Синтаксис мови, середовище виконання, запуск програм](#_Toc189510037)** [5](#_Toc189510037)

**[2 Програмування лінійних та розгалужених обчислювальних процесів](#_Toc189510038)** [9](#_Toc189510038)

**[2.1 Програмування циклічних обчислювальних процесів](#_Toc189510039)** [10](#_Toc189510039)

**[2.2 Типи даних визначені користувачем, магічні методи, виключення (Exceptions) та їх обробка](#_Toc189510040)** [13](#_Toc189510040)

**[2.3 Вбудовані структури даних, складні типи](#_Toc189510041)** [14](#_Toc189510041)

**[3 Функції](#_Toc189510042)** [16](#_Toc189510042)

**[3.1 Потокове введення, виведення та обробка даних](#_Toc189510043)** [17](#_Toc189510043)

**[3.2 Побудова проекту, модулі, пакети](#_Toc189510044)** [18](#_Toc189510044)

**[4 Робота з файловою системою, оператор with, контекстний менеджер](#_Toc189510045)** [20](#_Toc189510045)

**[4.1 Робота з протоколом HTTP, бібліотека requests](#_Toc189510046)** [22](#_Toc189510046)

**[5 Виконання завдань](#_Toc189510047)** [24](#_Toc189510047)

**[5.1 Завдання 1](#_Toc189510048)** [24](#_Toc189510048)

**[5.2 Завдання 2](#_Toc189510049)** [25](#_Toc189510049)

**[5.3 Завдання 3](#_Toc189510050)** [29](#_Toc189510050)

**[6 Висновки](#_Toc189510051)** [30](#_Toc189510051)

**[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ](#_Toc189510052)** [31](#_Toc189510052)

**[Додаток 1](#_Toc189510053)** [32](#_Toc189510053)

**[Додаток 2](#_Toc189510054)** [33](#_Toc189510054)

**[Додаток 3](#_Toc189510055)** [39](#_Toc189510055)

# **ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ**

1. **Скорочення:**

* *IDE (Integrated Development Environment)* – інтегроване середовище розробки
* *IoT (Internet of Things)* – Інтернет речей
* *JIT (Just-In-Time)* – технологія компіляції "на льоту"
* *OOP (Object-Oriented Programming)* – об'єктно-орієнтоване програмування

1. **Умовні позначення:**

* *x, y, z* – змінні
* *func\_name()* – виклик функції
* *class ClassName:* – визначення класу
* *#* – коментар у коді Python

1. **Символи:**

* = – оператор присвоювання
* == – оператор порівняння
* != – оператор нерівності
* \*\* – оператор піднесення до степеня
* // – оператор цілочисельного ділення

1. **Одиниці:**

* s – секунда
* KB, MB, GB – одиниці вимірювання обсягу пам’яті

1. **Основні терміни:**

* **Інтерпретатор** – програма, що виконує код Python рядок за рядком
* **Клас** – шаблон для створення об'єктів
* **Магічні методи** – спеціальні методи класу, що починаються і закінчуються подвійним підкресленням (\_\_init\_\_, \_\_str\_\_)
* **Виняток (Exception)** – помилка під час виконання програми
* **Ітератор** – об'єкт, що дозволяє перебирати елементи колекції
* **Генератор** – функція, що повертає значення по черзі, використовуючи yield
* **Список (List)** – змінювана структура даних, що містить впорядкований набір елементів

# **Вступ**

Автор: Глушко Анна Федорівна

Дата: 23.01.2025

Мета: Узагальнення вивчених тем

Перелік вивчених тем:

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.

# **Теоретичні відомості**

## **1.1 Вступне заняття**

Під час вступного заняття було розглянуто основні питання охорони праці, проведено відповідний інструктаж, що є необхідним для забезпечення безпечних умов навчання та роботи. Також ми ознайомилися з можливостями та правилами користування електронною бібліотекою коледжу, що сприятиме ефективному пошуку та використанню навчальних матеріалів. Окрім цього, вивчалися вимоги до оформлення звітної документації, що є важливим для правильного ведення та подання навчальних робіт.

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

Python – це високорівнева інтерпретована мова програмування, створена Гвідо ван Россумом у 1991 році. Основною ідеєю мови була простота читання коду та висока продуктивність розробки. Python отримав свою назву на честь британського комедійного гурту "Monty Python’s Flying Circus". З часом мова набула широкої популярності завдяки простому синтаксису, гнучкості та активній підтримці спільноти.

Python є динамічно типізованою та інтерпретованою мовою програмування, що робить її зручною для швидкої розробки. Основні характеристики:

* **Динамічна типізація** – типи змінних визначаються під час виконання програми.
* **Автоматичне керування пам’яттю** – Python використовує збирач сміття для автоматичного звільнення непотрібної пам’яті.
* **Лаконічний синтаксис** – код Python є зрозумілим і читається майже як звичайна англійська мова.
* **Кросплатформеність** – Python працює на більшості операційних систем, таких як Windows, Linux та macOS.

Python відноситься до мов з **слабкою (нестрогою) типізацією**, оскільки дозволяє виконувати операції між різними типами даних без явного приведення. Водночас мова має **динамічну типізацію**, що означає можливість зміни типу змінної в процесі виконання програми.

Python широко використовується у різних галузях, зокрема:

* **Web-розробка** – фреймворки Django та Flask дозволяють створювати веб-додатки різного рівня складності.
* **Інтернет речей (IoT)** – завдяки простоті використання Python підходить для програмування мікроконтролерів (наприклад, Raspberry Pi).
* **Data Science та машинне навчання** – бібліотеки NumPy, Pandas, TensorFlow, Scikit-learn застосовуються для аналізу даних та побудови моделей штучного інтелекту.
* **Системне програмування** – Python використовується для автоматизації адміністрування систем та написання скриптів.
* **DevOps** – у цій сфері Python допомагає у створенні автоматизованих скриптів для керування інфраструктурою, розгортання серверів та роботи з хмарними сервісами.

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

Python має лаконічний та інтуїтивно зрозумілий синтаксис. Наприклад, щоб вивести "Hello, World!", достатньо написати:

print("Hello, World!")

Основними елементами мови є:

* **Алфавіт мови** – набір символів, що використовується в коді (латинські літери, цифри, спеціальні символи, пробіли, табуляції).
* **Лексеми** – найменші одиниці коду, що мають сенс (ключові слова, ідентифікатори, оператори, роздільники).
* **Вирази** – комбінації значень, змінних, операторів, що обчислюються у певний результат.

Середовище виконання

Python-програми можуть запускатися у різних середовищах, таких як інтегровані середовища розробки (IDE), термінал чи командний рядок. Основний інтерпретатор для Python - це **CPython**, який є офіційною реалізацією мови. Окрім нього існують й інші інтерпретатори:

* **PyPy** – альтернатива CPython, яка використовує JIT-компіляцію для швидшого виконання коду.
* **Jython** – версія Python для Java-платформи, яка дозволяє взаємодіяти з Java-кодом.
* **IronPython** – реалізація для .NET-платформи, що інтегрується з C# та іншими мовами .NET.

Інтерпретатор і транслятор

Python є ***інтерпретованою мовою***, що означає, що код виконується послідовно, рядок за рядком, а не компілюється у виконуваний файл перед запуском. Інтерпретатор - це програма, що зчитує та виконує код Python. Транслятор же використовується у мовах, які перед виконанням потребують компіляції у машинний код.

Кросплатформеність

Python є ***кросплатформеною мовою***, що означає можливість її виконання на різних операційних системах (Windows, macOS, Linux) без зміни вихідного коду. Головне - мати встановлений інтерпретатор Python, який забезпечує сумісність між платформами.

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

Python є мовою зі строгою динамічною типізацією, що означає, що змінні не потребують оголошення типу під час ініціалізації, але їх тип визначається автоматично при присвоєнні значення.

Змінні та ініціалізація

Змінна – це іменований контейнер для зберігання даних. Присвоєння значення змінній виконується за допомогою оператора =:

x = 10 # Ціле число

pi = 3.14 # Дійсне число

name = "Anna" # Рядок

Вбудовані типи даних

Python має кілька основних вбудованих типів даних, які можна поділити на змінювані та незмінювані.

Незмінювані типи даних:

* int – цілі числа
* float – числа з плаваючою комою
* str – рядки
* tuple – кортежі
* bool – булеві значення (True або False)

Змінювані типи даних:

* list – списки
* dict – словники
* set – множини

Концепція посилань

У Python змінні є посиланнями на об'єкти в пам'яті, а не безпосередньо зберігають значення. Це означає, що змінювані об'єкти можуть змінювати свій стан без створення нової копії:

lst1 = [1, 2, 3]

lst2 = lst1 # lst2 посилається на той самий список

lst2.append(4)

print(lst1) # Виведе [1, 2, 3, 4]

Представлення чисел у пам’яті

Python автоматично керує пам’яттю для чисел. Цілі числа (int) можуть мати довільну довжину, тоді як числа з плаваючою комою (float) реалізовані на основі стандарту IEEE 754.

Пріоритетність обчислень

Python дотримується стандартної пріоритетності операторів:

1. () – дужки
2. \*\* – піднесення до степеня
3. \*, /, //, % – множення, ділення, цілочисельне ділення, остача
4. +, - – додавання і віднімання

result = 2 + 3 \* 4 # 14, множення виконується перед додаванням

result = (2 + 3) \* 4 # 20, дужки змінюють пріоритет

Вбудовані функції для обчислень

Python містить ряд вбудованих функцій для роботи з числами:

abs(-5) # 5, модуль числа

round(3.14159, 2) # 3.14, округлення до двох знаків

pow(2, 3) # 8, піднесення до степеня

max(5, 10) # 10, знаходження максимуму

Крім того, модуль math надає розширені математичні функції:

import math

print(math.sqrt(16)) # 4.0, квадратний корінь

print(math.pi) # 3.141592653589793

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

Python надає можливості для створення як лінійних, так і розгалужених обчислювальних процесів. Лінійний код виконується послідовно, тоді як розгалужений передбачає прийняття рішень на основі певних умов.

Логічні оператори у Python

Логічні оператори використовуються для створення складних умов у програмах:

* and – логічне "І" (усі умови мають бути істинними)
* or – логічне "АБО" (достатньо, щоб хоча б одна умова була істинною)
* not – логічне "НЕ" (змінює істинність виразу на протилежну)

x = 10

y = 20

print(x > 5 and y < 30) # True

print(x < 5 or y > 30) # False

print(not (x > 5)) # False

Умовні оператори (if-elif-else)

Для реалізації розгалужень у Python використовуються конструкції if, elif, else.

age = 18

if age < 18:

print("Молодший за 18")

elif age == 18:

print("Саме 18 років")

else:

print("Старший за 18")

Тернарний оператор

Для компактного запису умовних виразів використовується тернарний оператор:

x = 10

y = 20

min\_value = x if x < y else y

print(min\_value) # Виведе 10

Вкладені умови та відступи

Python використовує відступи для визначення блоків коду. При реалізації вкладених умов важливо правильно використовувати відступи.

score = 85

if score >= 50:

print("Проходить поріг")

if score >= 80:

print("Високий бал")

else:

print("Не проходить поріг")

## **2.1 Програмування циклічних обчислювальних процесів**

Циклічні обчислювальні процеси відіграють важливу роль у програмуванні, дозволяючи автоматизувати виконання однотипних операцій. У цій темі було розглянуто конструкції Python, такі як for і while, їх особливості, а також використання ітераторів та генераторів.

Основна частина

1. Цикли for і while

Python підтримує два основні типи циклів:

* for використовується для перебору ітерабельних об’єктів, таких як списки, кортежі, рядки тощо.
* while виконується доти, доки задана умова є істинною.

Приклад використання циклу for:

for i in range(5):

print(i)

Приклад використання циклу while:

x = 0

while x < 5:

print(x)

x += 1

2. Ітератори та ітерабельні об'єкти

Ітератори - це об'єкти, які реалізують методи \_\_iter\_\_() та \_\_next\_\_(). Вони використовуються для послідовного отримання елементів із колекцій.

Приклад створення власного ітератора:

class MyIterator:

def \_\_init\_\_(self, start, end):

self.current = start

self.end = end

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.current >= self.end:

raise StopIteration

self.current += 1

return self.current - 1

it = MyIterator(0, 5)

for num in it:

print(num)

3. Генератори

Генератори – це особливий тип ітераторів, які використовують ключове слово yield для збереження стану між викликами.

Приклад створення генератора:

def my\_generator():

for i in range(5):

yield i

for value in my\_generator():

print(value)

Головна перевага генераторів у тому, що вони дозволяють економити пам'ять, оскільки не зберігають усі елементи одразу, а створюють їх на вимогу.

4. Застосування ітераторів та генераторів

Ітератори та генератори корисні при роботі з великими обсягами даних. Наприклад, під час обробки великих файлів або потокової обробки даних у реальному часі.

Приклад обробки файлу за допомогою генератора:

def read\_large\_file(file\_path):

with open(file\_path, 'r') as file:

for line in file:

yield line.strip()

for line in read\_large\_file('data.txt'):

print(line)

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

У програмуванні часто виникає необхідність створювати власні типи даних, що дозволяє зробити код більш структурованим і гнучким. У мові Python це реалізується за допомогою класів. Окрім цього, важливу роль відіграють магічні методи, які дозволяють змінювати поведінку об'єктів, а також механізми обробки виключень, що забезпечують стійкість програм до помилок.

Основна частина

1. Типи даних, визначені користувачем

Python дозволяє створювати власні типи даних шляхом оголошення класів. Клас — це шаблон, який описує структуру та поведінку об'єктів. Наприклад, можна створити клас для представлення людей, який міститиме інформацію про ім'я та вік. Використання таких класів допомагає спростити управління даними та підвищити зручність роботи з ними.

2. Магічні методи

Магічні методи в Python — це спеціальні функції, які починаються і закінчуються подвійним підкресленням. Вони дозволяють налаштовувати поведінку об'єктів у різних ситуаціях. Наприклад, метод \_\_init\_\_ використовується для ініціалізації об'єкта, \_\_str\_\_ відповідає за текстове представлення об'єкта, а \_\_eq\_\_ визначає порівняння об'єктів. Завдяки цим методам можна зробити класи більш інтуїтивними та зручними у використанні.

3. Виключення та їх обробка

Під час виконання програм можуть виникати помилки, які призводять до зупинки роботи. Щоб запобігти цьому, у Python є механізм обробки виключень, що дозволяє контролювати виняткові ситуації та реагувати на них належним чином. Використання конструкцій try-except допомагає уникнути аварійного завершення програми та забезпечує можливість обробки різних типів помилок. Додатково існують блоки else та finally, які дозволяють виконувати додаткові дії після основного блоку коду.

4. Власні виключення

Python дозволяє створювати власні класи виключень, які можуть використовуватися для обробки специфічних ситуацій. Це корисно, коли потрібно визначити унікальні помилки, пов'язані з логікою програми. Наслідування власного класу від Exception дає змогу задавати власні повідомлення про помилки та покращує контроль за виконанням програми.

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

У мові Python є набір вбудованих структур даних, які забезпечують ефективне зберігання та обробку інформації. Основними з них є списки, кортежі, словники, множини та рядки. Кожна структура має свої особливості та підходить для вирішення певних задач. У цій роботі розглядаються основні характеристики цих структур, поняття хеш-функцій та їх використання, а також механізми реалізації списків у Python.

1. Вбудовані структури даних

* **Списки (**list**)**

Списки є динамічними структурами, що можуть змінюватися під час виконання програми. Вони дозволяють зберігати елементи різних типів та підтримують індексацію.

* **Кортежі (**tuple**)**

Кортежі — це незмінні структури, що забезпечують швидший доступ до елементів та можуть використовуватися як ключі у словниках.

* **Словники (**dict**)**

Словники реалізовані на основі хеш-таблиць і дозволяють швидко отримувати значення за ключем. Вони є зручними для зберігання асоціативних даних.

* **Множини (**set**)**

Множини забезпечують зберігання унікальних елементів та виконання операцій над множинами, таких як об’єднання, перетин і різниця.

* **Рядки (**str**)**

Рядки є незмінними послідовностями символів, що широко використовуються в обробці текстових даних.

2. Хеш-функції та хеш-таблиці

Хеш-функція — це алгоритм, що перетворює вхідні дані у числове значення (хеш). Хеш-таблиці дозволяють швидко отримувати доступ до даних, зберігаючи їх у форматі ключ-значення. Важливо, що дані, які використовуються як ключі у словниках або множинах, мають бути hashable — тобто незмінними та відповідати певним вимогам.

3. Реалізація списків у Python

Списки у Python реалізовані як масиви об’єктів PyObject, що дозволяє швидко додавати та вилучати елементи. Проте через особливості реалізації вони можуть споживати більше пам’яті. З іншого боку, зв’язані списки, що використовуються в інших мовах програмування, є ефективними для частого додавання та видалення елементів, хоча доступ до них займає більше часу.

4. Вибір оптимальної структури даних

Правильний вибір структури даних є важливим для ефективності алгоритмів. Наприклад, якщо потрібен швидкий доступ до елементів за індексом, доцільно використовувати список або кортеж. Для зберігання пар ключ-значення підходить словник, а множини корисні для унікальних елементів.

# **3 Функції**

Функції відіграють ключову роль у програмуванні, оскільки дозволяють структурувати код, уникати повторення та спрощувати його супровід. У Python функції визначаються за допомогою ключового слова def, після якого вказується ім’я функції та список параметрів у дужках.

* Створення та використання функцій

Функція оголошується за наступним синтаксисом:

*def назва\_функції(параметри):*

*# Тіло функції*

*return значення*

Функція може приймати параметри, які дозволяють передавати в неї необхідні дані. Також вона може повертати результат за допомогою ключового слова return.

* Передача параметрів та область видимості змінних

Python підтримує передачу параметрів за посиланням, тобто функція отримує доступ до змінної, що передана у виклик. При цьому важливо враховувати область видимості змінних: локальні змінні функції не впливають на глобальні змінні без спеціальних вказівок (global або nonlocal).

* Функції як об’єкти

Функції у Python є об’єктами першого класу, що означає можливість їх передачі як аргументів іншим функціям, збереження у змінних та використання як значень, що повертаються з функцій. Це дозволяє будувати гнучкі та розширювані рішення.

* Анонімні функції (lambda)

Для створення коротких функцій, які складаються з однієї інструкції, використовуються lambda-функції. Вони мають наступний синтаксис:

*lambda аргументи: вираз*

Це зручно для використання в якості аргументів функцій, наприклад, при сортуванні списків або у функціях вищого порядку.

* Рекурсія

Рекурсія — це метод, коли функція викликає саму себе. У Python рекурсивні виклики мають обмеження на глибину рекурсії, яке встановлене за замовчуванням. Використання рекурсії може бути зручним для вирішення задач, пов’язаних із деревоподібними структурами, але може поступатися за ефективністю ітеративним підходам.

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

Робота з потоками введення та виведення є важливою частиною програмування, особливо коли йдеться про обробку великих обсягів даних у реальному часі. Python надає зручні інструменти для роботи з потоками через sys.stdin, sys.stdout та sys.stderr, що дозволяє ефективно керувати введенням, виведенням та обробкою помилок.

Python працює з трьома основними потоками:

* **sys.stdin** — потік стандартного введення, який використовується для отримання даних від користувача або з іншого процесу.
* **sys.stdout** — потік стандартного виведення, призначений для виводу інформації.
* **sys.stderr** — потік для виведення помилок та діагностичних повідомлень.

Використання потокового введення дозволяє обробляти вхідні дані постійним потоком, не чекаючи завершення всього введення. Це корисно для аналізу логів або читання даних із сенсорів у режимі реального часу.

Для зберігання та обміну даними використовуються різні формати, серед яких:

* **JSON Lines (NDJSON)** — зручний формат для роботи з великими наборами JSON-об’єктів, кожен з яких записується на новому рядку.
* **Protocol Buffers (Protobuf)** — бінарний формат, що використовується для ефективного зберігання та передачі структурованих даних.
* **Parquet** — колонковий формат зберігання, який ідеально підходить для аналізу великих даних.
* **CSV** — популярний текстовий формат, що широко застосовується для табличних даних.

Змінні оточення дозволяють зберігати конфігураційні параметри поза кодом, що спрощує адаптацію застосунку до різних умов. Python надає доступ до змінних оточення через os.environ, що дозволяє легко змінювати параметри роботи без редагування вихідного коду.

## **3.2 Побудова проекту, модулі, пакети**

Мені потрібно написати велику програму. Замість того, щоб усе писати в одному файлі, Python дозволяє розділити код на частини, які називаються модулями. Наприклад, якщо я напишу калькулятор, можна створити окремий модуль math\_operations.py, де будуть функції додавання та віднімання.

Коли модулів стає багато, їх варто об’єднати в пакети. Пакет у Python — це звичайна папка з модулями та спеціальним файлом \_\_init\_\_.py, який робить її розпізнаваною як пакет.

Наприклад, структура проекту може виглядати так:

*my\_project/*

*│── main.py*

*│── my\_package/*

*│ │── \_\_init\_\_.py*

*│ │── module1.py*

*│ │── module2.py*

Це допомагає впорядкувати код і зробити його більш зрозумілим.

Python має велику кількість готових бібліотек, які можна встановити через pip. Наприклад, якщо потрібно працювати з запитами до веб-сайтів, можна встановити бібліотеку requests:

*pip install requests*

Це дозволяє легко інтегрувати сторонні рішення у свій проект.

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

Буває, що потрібно зберегти дані у файл або, навпаки, отримати з нього інформацію. Наприклад, уявіть, що у нас є текстовий файл data.txt, і потрібно його прочитати. У Python це можна зробити так:

*file = open("data.txt", "r")*

*data = file.read()*

*file.close()*

Але є проблема: якщо під час виконання коду виникне помилка, файл може залишитися відкритим, що може спричинити витік ресурсів. Саме для цього у Python є оператор with.

with автоматично закриває файл після виходу з блоку коду, навіть якщо сталася помилка:

*with open("data.txt", "r") as file:*

*data = file.read()*

Цей підхід робить код безпечнішим та зручнішим, тому його варто використовувати завжди при роботі з файлами.

Оператор with працює не лише з файлами. Наприклад, можна створити власний контекстний менеджер, який автоматично виконує певні дії при вході та виході з блоку коду. Для цього використовуються методи \_\_enter\_\_ і \_\_exit\_\_:

*class FileManager:*

*def \_\_init\_\_(self, filename, mode):*

*self.file = open(filename, mode)*

*def \_\_enter\_\_(self):*

*return self.file*

*def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):*

*self.file.close()*

*with FileManager("data.txt", "r") as file:*

*data = file.read()*

Так можна створювати свої контекстні менеджери для автоматизації управління ресурсами.

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

Коли ви вводите URL у браузері, він надсилає HTTP-запит на сервер, а сервер у відповідь повертає HTML-сторінку. Це все відбувається через протокол HTTP, який визначає, як клієнт і сервер взаємодіють.

HTTP підтримує кілька методів запитів:

* **GET** — отримання даних із сервера.
* **POST** — надсилання нових даних.
* **PUT** — оновлення існуючих даних.
* **DELETE** — видалення даних.

У Python для роботи з HTTP-запитами використовується бібліотека requests. Вона дозволяє легко виконувати запити та отримувати відповіді від серверів.

Приклад GET-запиту:

*import requests*

*response = requests.get("https://jsonplaceholder.typicode.com/posts/1")*

*print(response.status\_code) # Виведе код відповіді*

*print(response.json()) # Отримаємо дані у форматі JSON*

При отриманні відповіді важливо перевіряти код статусу, який повідомляє про успіх або помилки:

*if response.status\_code == 200:*

*print("Успішний запит")*

*else:*

*print("Помилка")*

Заголовки можна переглянути через *response.headers.*

Якщо потрібно передати параметри, наприклад, при фільтрації даних:

*params = {"userId": 1}*

*response = requests.get("https://jsonplaceholder.typicode.com/posts", params=params)*

Сесії дозволяють зберігати стан між запитами:

*session = requests.Session()*

*session.get("https://example.com/login") # Логін*

*response = session.get("https://example.com/dashboard") # Використання сесії*

# **5 Виконання завдань**

## **5.1 Завдання 1**

Код знаходиться у **додатку 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).

Приклад:

вхідна строка: abcdefg

вихід: (ae, 5, bcdfg)

**Пояснення:**

Цей код обробляє вхідний рядок і повертає кортеж, який складається з трьох елементів. Перший елемент - це рядок, що містить усі голосні літери з вхідної строки, зберігаючи їх порядок. Другий елемент - це кількість приголосних літер у рядку. Третій елемент - це рядок, що містить усі приголосні літери з вхідної строки, також зберігаючи порядок. Програма проходить по кожній літері рядка та перевіряє, чи вона є голосною або приголосною, додаючи її до відповідних змінних.

## **5.2 Завдання 2**

Код у **додатку 2**

Список задач для покращення навичок з програмування та вивчення Python зокрема.

**Пояснення:**

1. Функція `*find\_triples(n)*

Ця функція знаходить усі можливі трійки чисел *\( (a, b, c) \)*, які задовольняють рівняння:

*\[*

*a^2 + b^2 = c*

*\]*

*де \( c \leq n \).*

Метод роботи:

* Використовуються вкладені цикли для перебору всіх можливих значень \( a \) та \( b \), починаючи з 1 і до \( n \).
* Перевіряється умова, чи дорівнює сума квадратів чисел \( a \) та \( b \) числу \( c \).
* Якщо умова виконується і \( c \leq n \), трійка додається до списку.

2. Функція *`pascal\_triangle(n)`*

Генерує трикутник Паскаля до \( n \)-го рядка.

Алгоритм роботи:

* Створюється список списків, де кожен рядок ініціалізується одиницями.
* Для кожного наступного рядка, починаючи з третього, внутрішні елементи розраховуються як сума двох сусідніх елементів з попереднього рядка.
* Функція повертає повністю згенерований трикутник.

3. Функція *`eratosfen\_()`*

Реалізує решето Ератосфена для знаходження всіх простих чисел від 2 до 1000.

Принцип роботи:

* Створюється список усіх чисел від 2 до 1000.
* Алгоритм поступово викреслює всі числа, які діляться на вже знайдені прості числа.
* Коли всі непотрібні числа видалені, залишаються лише прості.

4. Функція *`prime\_factors(n)`*

Розкладає число \( n \) на прості множники.

Метод роботи:

* Починаємо перевірку з найменшого простого числа — 2.
* Поки число ділиться без остачі, додаємо його до списку множників і зменшуємо \( n \).
* Продовжуємо перевіряти більші числа, поки не отримаємо всі множники.

5. Функція *`palindroms()`*

Знаходить числа від 0 до 99, квадрати яких також є паліндромами.

Логіка роботи:

* Перевіряється, чи є число паліндромом (читається однаково в обидві сторони).
* Якщо так, перевіряється його квадрат.
* Якщо і квадрат є паліндромом, число додається до списку результатів.

6. Функція *`number\_to\_words(n)`*

Перетворює числа від 1 до 1000 у текстовий формат українською мовою.

Метод роботи:

* Використовуються три списки: для одиниць, десятків та сотень.
* Число розбивається на складові: сотні, десятки та одиниці.
* Формується рядок, що містить текстове представлення числа.
* Особливі випадки (наприклад, числа від 10 до 19) обробляються окремо.

7. Функції *`is\_prime(num)` та `find\_twin\_primes(n)`*

*Функція `is\_prime(num)`*

* Перевіряє, чи є число простим.
* Використовує оптимізований метод перевірки до \( \sqrt{n} \).

Функція *`find\_twin\_primes(n)`*

* Шукає всі пари "близнюків" серед простих чисел у діапазоні від \( n \) до \( 2n \).
* Простими числами-близнюками називаються числа, які відрізняються на 2 (наприклад, 11 і 13).

8. Функція *`format\_text(text, n)`*

Форматує текст, розбиваючи його на рядки довжиною \( n \).

Як працює:

* Розбиває текст на абзаци.
* Кожен абзац переноситься на нові рядки відповідно до максимальної ширини.
* Використовується модуль `textwrap`, що дозволяє зробити це автоматично.

Функція *`main()`*

Виконує вибір та запуск потрібної задачі, виходячи з введеного користувачем числа.

* Використовується `match case` для визначення обраної задачі.
* Для кожної задачі є відповідне число (від 1 до 8), яке користувач вводить.
* Якщо вибрана задача потребує вхідних даних, їх вводять перед викликом функції.

Ця функція є головною точкою входу для запуску програми.

## **5.3 Завдання 3**

Код у **додатку 3**

Розроблення програмного застосунку. Тема: Повідомлення про повітряну тривогу в інший канал комунікації за вибором (Telegram)

**Пояснення:**

Код реалізує Telegram-бота для моніторингу повітряних тривог в Україні через API alerts.in.ua. Бот отримує інформацію про тривоги та сповіщає користувачів про їх початок і завершення.

Основні компоненти

* **Бібліотеки**: імпортуються os, requests, time, telebot, datetime, pytz, threading, flask.
* **Ініціалізація**: використовується TOKEN для Telegram-бота та *API\_TOKEN* для отримання даних про тривоги.
* **Список регіонів**: містить назви всіх областей України.

Логіка роботи

1. **Отримання активних тривог** (*get\_active\_alerts*()): запит до API, повертає список тривог.
2. **Перевірка статусу регіону** (*check\_current\_alert\_status(region*)): фільтрує активні тривоги, визначає час початку.
3. **Моніторинг тривог** (*check\_region\_alerts(*)): запускається в окремому потоці, перевіряє зміни кожні 15 секунд, сповіщає користувача.

Взаємодія з користувачем

* **Команда /start**: надсилає список областей для вибору.
* **Вибір регіону** (*start\_monitoring(message*)): зупиняє попередній потік моніторингу, запускає новий, надсилає статус.

Запуск бота

При старті (*if \_\_name\_\_ == "\_\_main\_\_"*:):

* Запускається окремий потік *bot.polling(non\_stop=True*) для отримання повідомлень.
* Піднімається Flask-сервер для роботи у хмарі.

# **6 Висновки**

Протягом часу практики було розглянуто широкий спектр питань, пов’язаних із програмуванням на Python, починаючи від базових понять та закінчуючи складними аспектами розробки. Важливим початковим етапом стало знайомство з мовою Python, її історією, особливостями та сферами застосування. Я дізналася про гнучкість та універсальність цієї мови, її ключові характеристики, включаючи динамічну типізацію, автоматичне керування пам’яттю, а також зручність та читабельність коду. Було розглянуто основи синтаксису Python, середовище виконання та особливості запуску програм, що стало основою для подальшого навчання.

Окрему увагу було приділено змінним та типам даних, їх класифікації на змінювані та незмінювані, а також концепції посилань. Розглядалася робота з числовими типами, операторами та функціями для математичних обчислень, що дозволило краще зрозуміти механізми обробки даних у Python. Особливий акцент зроблено на вбудованих структурах даних, таких як списки, кортежі, множини та словники, а також методах їхнього використання.

Також було досліджено основи алгоритмізації та побудови програм, включаючи лінійні та розгалужені обчислювальні процеси. Було розглянуто логічні оператори, умовні конструкції (if-elif-else), тернарний оператор та вкладені умови, що є важливими для прийняття рішень у програмному коді. Значну увагу приділено циклічним обчислювальним процесам, що включають конструкції for і while, роботу з ітераторами та генераторами, а також їхні переваги у збереженні пам’яті та оптимізації роботи програм.

Не менш важливим аспектом стало вивчення механізмів обробки виключень та магічних методів. Було розглянуто основні принципи створення власних класів, особливості обробки винятків за допомогою конструкцій try-except, а також можливість створення власних виключень для підвищення стабільності програм. Вивчення магічних методів дозволило зрозуміти, як змінювати поведінку об'єктів, налаштовувати їхні атрибути та забезпечувати гнучкість роботи з класами.

Загалом, практика надала комплексні знання про Python, починаючи з базових концепцій та синтаксису і закінчуючи складними темами, такими як класи, обробка виключень та генератори. Освоєні концепції та методи програмування створюють міцний фундамент для подальшого розвитку у сфері програмування, дозволяючи застосовувати Python у різних галузях, таких як аналіз даних, веб-розробка, автоматизація та створення складних програмних рішень.

# **ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ**

1. Lutz M. Learning Python – Оксфорд: O'Reilly Media, 2013.
2. Guido van Rossum, Fred L. Drake Jr. The Python Language Reference. URL: [https://docs.python.org/3/reference/](https://docs.python.org/3/reference/" \t "_new) (дата звернення: 03.02.2025).
3. Matthes E. Python Crash Course – 2-ге видання. Оксфорд: No Starch Press, 2019.
4. Sweigart A. Automate the Boring Stuff with Python. URL: [https://automatetheboringstuff.com/](https://automatetheboringstuff.com/" \t "_new) (дата звернення: 03.02.2025).
5. Beazley D., Jones B. K. Python Cookbook – 3-тє видання. Оксфорд: O'Reilly Media, 2013.
6. Myers G. J. The Art of Software Testing / Second Edition. URL: [https://books.google.com.au/books?id=86rz6UExDEEC&printsec=frontcover&dq=he+art+of+Software+Testing+isbn&source=bl&ots=yQPrKcu7uO&sig=zebgmnVJm4tXt5P9392AiQ789SI&hl=en&ei=xWHBTcizMZDNrQfdrfXtBw&sa=X&oi=book\_result&ct=result#v=onepage&q&f=false](https://books.google.com.au/books?id=86rz6UExDEEC&printsec=frontcover&dq=he+art+of+Software+Testing+isbn&source=bl&ots=yQPrKcu7uO&sig=zebgmnVJm4tXt5P9392AiQ789SI&hl=en&ei=xWHBTcizMZDNrQfdrfXtBw&sa=X&oi=book_result&ct=result" \l "v=onepage&q&f=false" \t "_new) (дата звернення: 10.04.2023).

# **Додаток 1**

Код:

*string = input()*

*size = len(string)*

*string\_changed=string*

*line1=""*

*line2=0*

*line3=""*

*letters\_1=['a','e','i','o','u','y']*

*letters\_2=['b','c','d','f','g','h','j','k','l','m','n','p','q','r','s','t','v','w','x','z']*

*for i in range(0,6):*

*for y in range(0,size):*

*if string[y]==letters\_1[i]:*

*line1+=(letters\_1[i])*

*for i1 in range(0,len(letters\_2)):*

*for y1 in range(0,size):*

*if string[y1]==letters\_2[i1]:*

*line3+=(letters\_2[i1])*

*line2+=1*

*print("("+line1+ ", " +str(line2)+ ", "+line3+")")*

**Додаток 2**

**Код:**

*import textwrap*

*# <------TASK 1------>*

*def find\_triples(n):*

*"""*

*The find\_triples function generates all triples (a, b, c) such that a^2 + b^2 = c and c <= n.*

*:param n: The maximum value for c in the generated triples*

*:return: A list of tuples (a, b, c) satisfying the condition a^2 + b^2 = c, where 1 <= a <= b <= n and c <= n*

*"""*

*triples = []*

*for a in range(1, n + 1):*

*for b in range(a, n + 1):*

*c = a \*\* 2 + b \*\* 2*

*if c <= n:*

*triples.append((a, b, c))*

*return triples*

*# <------TASK 2------>*

*def pascal\_triangle(n):*

*"""*

*The pascal\_triangle function generates Pascal's triangle up to the n-th row.*

*:param n: The number of rows to generate in Pascal's triangle*

*:return: A list of lists representing Pascal's triangle, where each inner list is a row of the triangle*

*"""*

*triangle = [[1] \* (i + 1) for i in range(n)]*

*for i in range(2, n):*

*for j in range(1, i):*

*triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j]*

*return triangle*

*# <------TASK 3------>*

*def eratosfen\_():*

*"""*

*The eratosfen\_ function attempts to implement the Sieve of Eratosthenes to find all prime numbers between 2 and 1000.*

*- Inefficient list manipulation and indexing*

*- Logical error when checking divisibility (index mismatch)*

*- Potential for an out-of-range index during list operations*

*:return: This function does not return a value but prints steps and operations during the algorithm*

*"""*

*all\_numbers = []*

*numbers\_to\_remove = []*

*number\_sadist\_ID = 0*

*for i in range(2, 1001, 1):*

*all\_numbers.append(i)*

*finished = False*

*while finished == False:*

*for y in range(number\_sadist\_ID + 1, len(all\_numbers)):*

*if y % all\_numbers[number\_sadist\_ID] == 0:*

*if len(all\_numbers) > y:*

*numbers\_to\_remove.append(all\_numbers[y])*

*for u in range(0, len(numbers\_to\_remove)):*

*all\_numbers.remove(numbers\_to\_remove[u])*

*if number\_sadist\_ID >= len(all\_numbers):*

*print("It is over")*

*finished = True*

*else:*

*number\_sadist\_ID += 1*

*print("Changing\_number")*

*numbers\_to\_remove = []*

*# <------TASK 4------>*

*def prime\_factors(n):*

*"""*

*The prime\_factors function finds the prime factors of a given integer n.*

*:param n: An integer to factorize*

*:return: A list of prime factors of n*

*"""*

*factors = []*

*divisor = 2*

*while n >= divisor:*

*while n % divisor == 0:*

*factors.append(divisor)*

*n //= divisor*

*divisor += 1*

*return factors*

*# <------TASK 5------>*

*def palindroms():*

*"""*

*The palindroms function finds all numbers between 0 and 99 whose squares are also palindromes.*

*:return: A list of integers between 0 and 99 satisfying the palindromic square condition*

*"""*

*palindromic\_numbers = []*

*for num in range(0, 100):*

*mirror\_check = ""*

*a = num*

*for i in range(len(str(a)) - 1, -1, -1):*

*mirror\_check += str(a)[i]*

*if str(a) == mirror\_check:*

*mirror\_check = ""*

*a = a \* a*

*for i in range(len(str(a)) - 1, -1, -1):*

*mirror\_check += str(a)[i]*

*if str(a) == mirror\_check:*

*palindromic\_numbers.append(num)*

*return palindromic\_numbers*

*# <------TASK 6------>*

*def number\_to\_words(n):*

*"""*

*The number\_to\_words function converts an integer n (from 1 to 1000) into its Ukrainian text representation.*

*:param n: An integer in the range from 1 to 1000*

*:return: A string representing the number in Ukrainian words*

*"""*

*ones = ["", "один", "два", "три", "чотири", "п’ять", "шість", "сім", "вісім", "дев’ять"]*

*teens = ["десять", "одинадцять", "дванадцять", "тринадцять", "чотирнадцять", "п’ятнадцять", "шістнадцять",*

*"сімнадцять", "вісімнадцять", "дев’ятнадцять"]*

*tens = ["", "десять", "двадцять", "тридцять", "сорок", "п’ятдесят", "шістдесят", "сімдесят", "вісімдесят",*

*"дев’яносто"]*

*hundreds = ["", "сто", "двісті", "триста", "чотириста", "п’ятсот", "шістсот", "сімсот", "вісімсот", "дев’ятсот"]*

*if n == 1000:*

*return "тисяча"*

*result = []*

*h = n // 100*

*t = (n % 100) // 10*

*o = n % 10*

*if h:*

*result.append(hundreds[h])*

*if t == 1:*

*result.append(teens[o])*

*else:*

*if t:*

*result.append(tens[t])*

*if o:*

*result.append(ones[o])*

*return " ".join(result)*

*# <------TASK 7------>*

*def is\_prime(num):*

*"""*

*The is\_prime function checks whether a given number is a prime number.*

*:param num: The integer to check for primality*

*:return: True if num is a prime number, False otherwise*

*"""*

*if num < 2:*

*return False*

*for i in range(2, int(num \*\* 0.5) + 1):*

*if num % i == 0:*

*return False*

*return True*

*def find\_twin\_primes(n):*

*"""*

*The find\_twin\_primes function finds twin prime pairs starting from n and up to 2 \* n.*

*Twin primes are pairs of prime numbers that differ by 2.*

*:param n: The starting integer for the search*

*:return: A list of tuples (p1, p2) where p1 and p2 are twin primes*

*"""*

*twin\_primes = []*

*for num in range(n, 2 \* n):*

*if is\_prime(num) and is\_prime(num + 2):*

*twin\_primes.append((num, num + 2))*

*return twin\_primes*

*# <------TASK 8------>*

*def format\_text(text: str, n: int) -> str:*

*"""*

*The format\_text function reformats a block of text into lines of specified width.*

*:param text: The input text to be formatted*

*:param n: The maximum line width for each paragraph*

*:return: A string with formatted paragraphs, each wrapped to the specified width*

*"""*

*paragraphs = text.split("\n\n")*

*formatted\_paragraphs = ["\n".join(textwrap.wrap(par, width=n)) for par in paragraphs]*

*return "\n\n".join(formatted\_paragraphs)*

*def main():*

*"""*

*The main function serves as an entry point for executing different tasks based on user selection.*

*Available tasks:*

*1. Find triples (a, b, c) where a^2 + b^2 = c and c <= n*

*2. Generate Pascal's triangle up to n rows*

*3. Execute a modified version of the Sieve of Eratosthenes for primes up to 1000*

*4. Find prime factors of a given number n*

*5. Find numbers whose squares are palindromic*

*6. Convert an integer (1 ≤ n ≤ 1000) into its Ukrainian text representation*

*7. Find twin primes starting from n and up to 2 \* n*

*8. Format a block of text into lines of specified width*

*:return: None*

*"""*

*choose = int(input(f'Choose a number of task: '))*

*match choose:*

*case 1:*

*n = 50*

*result = find\_triples(n)*

*print(result)*

*case 2:*

*n = 10*

*triangle = pascal\_triangle(n)*

*for row in triangle:*

*print(" ".join(map(str, row)))*

*case 3:*

*eratosfen\_()*

*case 4:*

*n = 250*

*factors = prime\_factors(n)*

*print(f"{n} = {', '.join(map(str, factors))}")*

*case 5:*

*print(palindroms())*

*case 6:*

*n = int(input("Введіть число (≤1000): "))*

*if 1 <= n <= 1000:*

*print(number\_to\_words(n))*

*else:*

*print("Число повинно бути від 1 до 1000!")*

*case 7:*

*n = 10*

*print(find\_twin\_primes(n))*

*case 8:*

*text = """Привіт, як справи?.*

*Привіт! В мене все добре, а в тебе?."""*

*n = 60*

*formatted\_text = format\_text(text, n)*

*print(formatted\_text)*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*

**Додаток 3**

**Код:**

*import os*

*import requests*

*import time*

*import telebot*

*from telebot import types*

*from datetime import datetime*

*import pytz*

*import threading*

*from flask import Flask*

*TOKEN = '7957207504:AAGEKxZrEcAI1iLgdKw6bvrLQ9rylYuqK\_I'*

*bot = telebot.TeleBot(TOKEN)*

*API\_URL = "https://api.alerts.in.ua/v1/alerts/active.json"*

*API\_TOKEN = "3b5106a1058f3c29227de73d95c1dcabc2488358ab2203"*

*headers = {*

*"Authorization": f"Bearer {API\_TOKEN}",*

*"Cache-Control": "no-cache"*

*}*

*app = Flask(\_\_name\_\_)*

*REGIONS = [*

*"Автономна Республіка Крим", "Вінницька область", "Волинська область", "Дніпропетровська область", "Донецька область",*

*"Житомирська область", "Закарпатська область", "Запорізька область", "Івано-Франківська область",*

*"Київська область", "Кіровоградська область", "Луганська область", "Львівська область",*

*"Миколаївська область", "Одеська область", "Полтавська область", "Рівненська область",*

*"Сумська область", "Тернопільська область", "Харківська область", "Херсонська область",*

*"Хмельницька область", "Черкаська область", "Чернівецька область", "Чернігівська область", "м. Київ"*

*]*

*active\_monitoring\_threads = {}*

*def get\_active\_alerts():*

*response = requests.get(API\_URL, headers=headers)*

*if response.status\_code == 200:*

*return response.json().get("alerts", [])*

*return []*

*def check\_current\_alert\_status(region):*

*kyiv\_tz = pytz.timezone("Europe/Kyiv")*

*alerts = get\_active\_alerts()*

*oblast\_wide\_alerts = [*

*alert for alert in alerts*

*if alert.get("location\_oblast") == region and alert.get("location\_title", "").strip() == region*

*]*

*if not oblast\_wide\_alerts:*

*filtered\_alerts = [*

*alert for alert in alerts*

*if alert.get("location\_oblast") == region and "район" not in alert.get("location\_title", "").lower()*

*]*

*else:*

*filtered\_alerts = oblast\_wide\_alerts*

*if filtered\_alerts:*

*start\_time = filtered\_alerts[0].get("started\_at", "невідомо")*

*if start\_time != 'невідомо':*

*start\_time = pytz.utc.localize(datetime.strptime(start\_time, "%Y-%m-%dT%H:%M:%S.%fZ")).astimezone(kyiv\_tz)*

*start\_time = start\_time.strftime("%d/%m/%Y %H:%M:%S")*

*return True, f"🚨 {region}: активна тривога! Початок: {start\_time}."*

*else:*

*return False, f"✅ {region}: повітряна тривога не активна."*

*def check\_region\_alerts(monitoring\_state, last\_alert\_status, stop\_event):*

*while not stop\_event.is\_set():*

*region = monitoring\_state["region"]*

*chat\_id = monitoring\_state["chat\_id"]*

*current\_alert\_status, message = check\_current\_alert\_status(region)*

*if current\_alert\_status != last\_alert\_status[0]:*

*bot.send\_message(chat\_id, message)*

*last\_alert\_status[0] = current\_alert\_status*

*time.sleep(15)*

*@bot.message\_handler(commands=["start"])*

*def set\_monitored\_region(message):*

*keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)*

*for region in REGIONS:*

*keyboard.add(types.KeyboardButton(region))*

*bot.send\_message(message.chat.id, "Оберіть область для моніторингу:", reply\_markup=keyboard)*

*@bot.message\_handler(func=lambda message: message.text in REGIONS)*

*def start\_monitoring(message):*

*region = message.text*

*chat\_id = message.chat.id*

*if chat\_id in active\_monitoring\_threads:*

*stop\_event, thread = active\_monitoring\_threads[chat\_id]*

*stop\_event.set()*

*thread.join()*

*monitoring\_state = {"region": region, "chat\_id": chat\_id}*

*current\_alert\_status, initial\_message = check\_current\_alert\_status(region)*

*bot.send\_message(chat\_id, f"🔍 {region}: Моніторинг тривог розпочато.")*

*bot.send\_message(chat\_id, initial\_message)*

*last\_alert\_status = [current\_alert\_status]*

*stop\_event = threading.Event()*

*monitoring\_thread = threading.Thread(*

*target=check\_region\_alerts, args=(monitoring\_state, last\_alert\_status, stop\_event), daemon=True*

*)*

*monitoring\_thread.start()*

*active\_monitoring\_threads[chat\_id] = (stop\_event, monitoring\_thread)*

*if \_\_name\_\_ == "\_\_main\_\_":*

*threading.Thread(target=lambda: bot.polling(non\_stop=True), daemon=True).start()*

*port = int(os.environ.get("PORT", 5000))*

*app.run(host="0.0.0.0", port=port)*