**Завдання №4**

**2.1. PHP/Python/Go  
Вам надано фрагмент коду на кожній з мов програмування (PHP, Python, Go). Ваше завдання:**

* + **Проаналізувати кожен фрагмент.**
  + **Написати короткий опис того, що робить цей код, виявити потенційні проблеми або покращення (не більше 200 слів для кожного фрагменту).**

**PHP Фрагмент**

<?php

function calculateAge($birthDate) {

    $birthDate = new DateTime($birthDate);

    $today = new DateTime('today');

    $age = $birthDate->diff($today)->y;

    return $age;

}

$age = calculateAge('1990-01-01');

echo "Age: $age";

?>

**Опис:**

Код визначає вік людини на основі дати її народження. Функція `calculateAge($birthDate)` приймає дату народження як рядок, створює об'єкт `DateTime` для цієї дати та для сьогоднішньої дати, а потім обчислює різницю в роках між ними. Нарешті, функція повертає значення віку.

**Потенційні проблеми та покращення:**

1. **Формат дати**: Код передбачає, що дата передається у форматі `Y-m-d`. Це неявне припущення, і варто було б додати перевірку формату вхідних даних.

2. **Обробка помилок**: Якщо дата введена неправильно або в неправильному форматі, це призведе до виключення. Було б краще додати обробку помилок або перевірку валідності вхідної дати перед створенням об'єкта `DateTime`.

3. **Оптимізація**: Код працює коректно, але може бути оптимізованим для перевірки того, чи об'єкт `DateTime` для поточної дати необхідний. Можна замість цього отримати поточну дату напряму через `new DateTime()` без додаткової змінної `$today`.

**Python Фрагмент**

from datetime import datetime

def calculate\_age(birth\_date):

    birth\_date = datetime.strptime(birth\_date, "%Y-%m-%d")

    today = datetime.today()

    age = today.year - birth\_date.year - ((today.month, today.day) < (birth\_date.month, birth\_date.day))

    return age

age = calculate\_age('1990-01-01')

print(f"Age: {age}")

**Опис:**

Код визначає вік людини на основі дати народження. Функція `calculate\_age(birth\_date)` приймає дату народження як рядок, перетворює її у формат `datetime` і обчислює вік шляхом вирахування років. Вона також перевіряє, чи відзначався день народження в цьому році, щоб коригувати вік.

**Потенційні проблеми та покращення:**

1. **Формат дати**: Код очікує дату у форматі `"%Y-%m-%d"`. Якщо вхідний рядок не відповідає цьому формату, код згенерує помилку. Додавання обробки виключень (`try-except`) було б корисним.

2. **Місяць і день**: Перевірка `(today.month, today.day) < (birth\_date.month, birth\_date.day)` гарантує, що враховується, чи був день народження вже цього року, що є важливою деталлю.

**Go Фрагмент**

package main

import (

    "fmt"

    "time"

)

func calculateAge(birthDate string) int {

    layout := "2006-01-02"

    bd, \_ := time.Parse(layout, birthDate)

    today := time.Now()

    age := today.Year() - bd.Year()

    if today.YearDay() < bd.YearDay() {

        age--

    }

    return age

}

func main() {

    age := calculateAge("1990-01-01")

    fmt.Println("Age:", age)

}

**Опис**:

Цей Go код визначає вік людини на основі дати її народження. Функція `calculateAge(birthDate string)` приймає дату народження як рядок, перетворює її у формат `time.Time`, а потім обчислює різницю в роках. Вона також перевіряє, чи відзначався день народження цього року, щоб коригувати вік.

**Потенційні проблеми та покращення:**

1. **Обробка** **помилок**: У цьому фрагменті код не обробляє помилки парсингу дати (`time.Parse`), що може спричинити неочікувану поведінку. Варто додати перевірку на помилки, щоб гарантувати коректну обробку даних.

2. **Оновлення** **часу**: Використання `time.Now()` для отримання поточного часу є ефективним рішенням, але важливо пам'ятати, що цей метод повертає локальний час. В деяких випадках може бути важливо використовувати `UTC` для уникнення помилок через часові зони.

**Завдання №5**

**Комунікація з командою:**

Уявіть, що вам потрібно пояснити команді, як ви реалізували тести та які потенційні проблеми ви виявили в існуючому коді. Напишіть короткий звіт (до 500 слів), який включає:

* + Опис підходу до написання тестів.
  + Виявлені проблеми та пропозиції щодо покращення.
  + Висновки та рекомендації.

**Звіт щодо реалізації тестів та виявлених проблем**

**Опис підходу до написання тестів**

Для тестування коду було використано підхід, заснований на модульному тестуванні (unit testing). Мета полягала в тому, щоб перевірити функціональність окремих модулів або функцій ізоляційно, що дозволило чітко визначити джерело проблеми у разі виявлення помилки. Тести були написані з використанням фреймворку `pytest`, який забезпечує зручний синтаксис та розширену функціональність для написання та виконання тестів.

**Основні кроки:**

1. **Підготовка середовища**: Налаштування тестового середовища включало конфігурацію тестових файлів, які дублюють типові сценарії використання коду, а також створення mock-об'єктів для відтворення залежностей між модулями.

2. **Покриття тестами**: Основну увагу було приділено ключовим функціям та класам, які забезпечують критично важливу функціональність. Покриття тестами сягнуло близько 80% коду, що дозволяє бути впевненим у стабільності основного функціоналу.

3**. Організація тестів**: Було створено окремі файли для тестів відповідно до модулів, що тестуються. Це забезпечило зрозумілу структуру та полегшило навігацію по тестовому коду.

**Виявлені проблеми та пропозиції щодо покращення**

**Під час написання тестів були виявлені кілька потенційних проблем у існуючому коді:**

1. **Слабка ізоляція модулів**: Деякі модулі занадто тісно пов'язані один з одним, що ускладнює їх тестування та підвищує ймовірність появи помилок через неочікувану поведінку залежностей. Пропозиція: варто розглянути можливість рефакторингу коду з метою поліпшення його модульності. Використання інтерфейсів або залежностей через ін'єкцію може значно підвищити ізольованість модулів.

2. **Недостатня перевірка обробки виключень**: У кількох місцях код не повністю враховує можливість виникнення виняткових ситуацій, що може призвести до непередбачуваної поведінки програми. Пропозиція: слід додати перевірку та обробку можливих виключень у відповідних місцях, а також забезпечити тестування цих випадків.

3. **Недостатня документація**: Деякі складні логічні частини коду недостатньо документовані, що ускладнює розуміння їхньої роботи і, як наслідок, тестування. Пропозиція: необхідно додати докладні коментарі та пояснення до ключових частин коду.

4. **Проблеми з продуктивністю**: Деякі тести вказали на можливе зниження продуктивності в певних функціях при великому обсязі даних. Пропозиція: оптимізувати ці функції або використовувати кешування результатів для зменшення часу обробки.

**Висновки та рекомендації**

Загалом, тестування продемонструвало достатню стабільність основного функціоналу, але також виявило кілька проблем, які потребують уваги. Для покращення якості коду рекомендую виконати такі дії:

1. Провести рефакторинг із метою покращення модульності та ізольованості компонентів.

2. Переглянути обробку виключень і додати відповідні тести.

3. Покращити документацію складних частин коду для полегшення його підтримки в майбутньому.

4. Розглянути можливості оптимізації функцій, які демонструють низьку продуктивність при обробці великих даних.

Виконання цих заходів сприятиме підвищенню стабільності та підтримуваності коду, що в майбутньому зменшить час на вирішення потенційних проблем.