**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.

1. class PersonAnalysis:
2. def \_\_init\_\_(self, name, age, height, weight):
3. self.name = name
4. self.age = age
5. self.height = height
6. self.weight = weight
7. def calculate\_bmi(self):
8. return self.weight / ((self.height / 100) \*\* 2)
9. def get\_health\_status(self):
10. bmi = self.calculate\_bmi()
11. if bmi < 18.5:
12. return "Underweight"
13. elif 18.5 <= bmi < 24.9:
14. return "Normal weight"
15. elif 25 <= bmi < 29.9:
16. return "Overweight"
17. else:
18. return "Obesity"
19. def \_\_str\_\_(self):
20. return f"Name: {self.name}, Age: {self.age}, Height: {self.height}, Weight: {self.weight}, BMI: {self.calculate\_bmi()}, Health Status: {self.get\_health\_status()}"
21. *# Приклад використання*
22. person = PersonAnalysis("John Doe", 30, 175, 70)
23. print(person)

**Звіт щодо реалізації тестів для класу `PersonAnalysis` та виявлених проблем**

**Опис підходу до написання тестів**

Для тестування класу `PersonAnalysis` було обрано підхід модульного тестування з використанням фреймворку `pytest`. Основна мета тестування полягала в тому, щоб перевірити коректність роботи методів `calculate\_bmi()` та `get\_health\_status()`, а також переконатися, що метод `\_\_str\_\_()` генерує правильний текстовий вихід.

**Тести були реалізовані з наступними кроками:**

1. **Покриття основних сценаріїв:** Було створено декілька тестових випадків для перевірки кожного з методів класу. Це включало тестування коректності розрахунку індексу маси тіла (BMI) та відповідності статусу здоров'я згідно з різними значеннями BMI.

2. **Перевірка граничних умов:** Особлива увага приділялася граничним значенням для категорій "Underweight", "Normal weight", "Overweight" і "Obesity". Це забезпечило точність визначення статусу здоров'я при значеннях BMI, що наближаються до меж цих категорій.

3. **Перевірка текстового виходу:** Метод `\_\_str\_\_()` був протестований на предмет правильності формування вихідного тексту, що включає ім'я, вік, зріст, вагу, значення BMI та статус здоров'я.

**Виявлені проблеми та пропозиції щодо покращення**

**В процесі тестування були виявлені декілька потенційних проблем:**

1**. Жорстко закодовані значення**: Граничні значення для визначення статусу здоров'я (наприклад, 18.5, 24.9, 29.9) жорстко закодовані в методі `get\_health\_status()`. Це може ускладнити підтримку і модифікацію коду в майбутньому, якщо ці значення зміняться. \*\*Пропозиція:\*\* Винести ці значення в окремі константи або конфігураційний файл, щоб полегшити їх зміну та покращити читабельність коду.

2. **Неперевірена робота з невалідними даними**: Код не враховує можливість введення невалідних даних, таких як негативні значення віку, росту або ваги, що може призвести до некоректних результатів. \*\*Пропозиція:\*\* Додати перевірку вхідних даних у конструкторі класу та виводити відповідні винятки, якщо дані не відповідають очікуваним критеріям.

3. **Обмежена розширюваність:** Якщо в майбутньому виникне необхідність розширити клас, наприклад, додати інші показники здоров'я, код може потребувати значного рефакторингу. \*\*Пропозиція:\*\* Розглянути можливість використання спадкування або композиції для легшого додавання нових функціональних можливостей.

4. **Тестування текстових форматів:** Метод `\_\_str\_\_()` повертає строку з форматованим текстом, що включає числові значення. Проте, тестування може бути ускладнене, якщо зміниться формат виведення. **Пропозиція**: Використовувати форматування з використанням шаблонів або окремого метода для генерації тексту, що спростить тестування та забезпечить гнучкість у зміні формату.

**Висновки та рекомендації**

В результаті тестування класу `PersonAnalysis` було підтверджено коректну роботу основних методів, проте виявлені деякі потенційні проблеми, які можуть вплинути на підтримку та розширюваність коду в майбутньому.

**Для покращення якості коду рекомендую:**

1. Винести граничні значення для категорій статусу здоров'я в окремі константи.

2. Додати перевірку вхідних даних на валідність.

3. Розглянути можливість покращення архітектури класу для легшого додавання нових функцій.

4. Оптимізувати метод `\_\_str\_\_()` для полегшення тестування та зміни формату виведення.

Ці рекомендації допоможуть покращити підтримку коду, підвищити його гнучкість і забезпечити стабільність при внесенні змін у майбутньому.