Formative activities

Optional extension activities

1. **Write a Python program to achieve basic employee-related functionality, including retaining employee details and allowing an employee to book a day of annual leave. Extend the Python program you created to use protected and unprotected variables.**

Based on Python R. (N.D) examples, the software below was made to show how a Python programme can carry out basic employment-related tasks, including keeping employee data and letting them book annual leave days. It also demonstrates the usage of protected and unprotected variables:

```
# Creating Class Hierarchies

class Employee:
    def __init__(self, name: object, employee_id: object) -> object:
        self._name = name  # protected variable
        self.employee_id = employee_id  # unprotected variable
        self._annual_leave_balance = 20  # protected variable

    def book_annual_leave(self, days):
        if days <= self._annual_leave_balance:
            self._annual_leave_balance -= days
            print(f"{self._name} has booked {days} day(s) of annual leave.")
        else:
            print("Insufficient annual leave balance.")

    def get_annual_leave_balance(self) -> object:
        return self._annual_leave_balance


# Creating an employee instance
employee1: Employee = Employee("Sandra Cristina", 141977)

# Accessing protected and unprotected variables
print(f"Employee ID: {employee1.employee_id}")
print(f"Employee name: {employee1._name}")  # Protected variable
print(f"Annual leave balance: {employee1.get_annual_leave_balance()}")

# Booking annual leave
employee1.book_annual_leave(5)
print(f"Annual leave balance: {employee1.get_annual_leave_balance()}")
```

(Rodriguez, N.D.).

In this program, the **Employee** class represents an employee and has the following attributes and methods:

- **_name**: A protected variable that stores the employee's name.
- **employee_id**: An unprotected variable that stores the employee's ID.
- **_annual_leave_balance**: A protected variable representing the employee's days available for annual leave.

The **book_annual_leave** method allows the employee to book a specified number of days of annual leave. Then checks if the requested days are available based on the remaining balance and updates it accordingly.

The **get_annual_leave_balance** method returns the current annual leave balance days for an employee.

To test the program, an instance of the **Employee** class is created with a name and the employee's ID. The protected and unprotected variables are accessed and displayed. Then, the employee books five days of annual leave, and the remaining day's balance is displayed.

Reference:

Rodriguez, I. (N.D.). *Inheritance and Composition: A Python OOP Guide – Real Python*. [online] realpython.com. Available at: https://realpython.com/inheritance-composition-python/.

Unit 2 – Reflection

In my studies of the Object-oriented program module, I thoroughly reviewed the Codio module on Class, Functions, and Methods in Unit 2. This helped me better understand how to enhance the development of Python programs. Additionally, I learned valuable insights into the importance of Debugging: Classes and Objects that will benefit my coding endeavours.

In completing an extension activity, I wrote a Python program to manage basic employee tasks, such as storing employee information and enabling employees to request annual leave. I further developed this program by incorporating protected and unprotected variables. I have documented my findings in my e-portfolio.

The Summative Assessment was another significant aspect of this module, which required designing and implementing software to facilitate the operation of an autonomous car. Additionally, I prepared a pen portrait for a driverless car user and created a use case model to outline how users interact with the software system. I have recorded these activities in my ePortfolio.

Throughout this module, I have explored different aspects of Python programming and object-oriented analysis. I expanded my knowledge of analysing program domains using case and state machine diagrams. Understanding the interactions between software and actors and the functional requirements of a particular software system was crucial to my learning. This helped me identify the essential features and behaviours needed in a software program. Moreover, I have learned how to create use-case diagrams that illustrate various events and interactions in a software development context, providing a clear understanding of the interactions between actors and systems. Reading chapters 5 and 6 from The Unified Modeling Language

Reference Manual gave me a deeper appreciation for UML and its significance in relationships, dependencies, and modelling. This newfound knowledge has improved my abilities and will be invaluable in accurately conveying the system design in the upcoming summative assessment. I am thrilled to apply my new skills and excel in the upcoming assignment.