



Shaheed Zulfikar Ali Bhutto Institute of Science & Technology University

ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT

Total Marks: 04

Obtained Marks: _____

Programming for Artificial Intelligence

Assignment # 01

Last date of Submission: 3rd March 2024

Submitted To: Mr. Saad Irfan Khan

Student Name: Syed Muhammad Mustafa

Reg. Number: 22108130

ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT

Instructions: *Copied or shown assignments will be marked zero. Late submissions are not entertained in any case.*

CLO 1 – PLO A, E – C2

Question 01.

(4 Marks)

Consider yourself working as a developer for a large data analytics firm and you have been tasked with developing a Python program for analyzing sales data in some retail company. To complete this task, you are first required to:

1. Explain how Python's basic constructs like loops, if-else statements, and functions can be used to manipulate and analyze the data effectively.

Loops can be used in relation to running a specific formula or an operation multiple times at once. For example in the data analytics firms I am given data for SZABIST, and I am asked calculate the total fee accumulated/paid by the students, I will then use a loop to run through each student's fee and then sum them up at the end. This will allow me to use 3 – 4 lines of code to perform the summation, where if I were to sum them individually then that would practically take my hours.

If-else statements can be used where certain conditions are required to execute a task/operation. For example I am told to filter out all the students who have not yet paid their semester fees, I will use a loop and have a condition in it that if the student fee's attribute contains 'Not Paid' then print their registration or names. This allows me to easily filter out things that I need or don't need in one chunk of code.

Functions can be used to execute large blocks of code again in different parts of the program. For example if I required need to calculate the sum of all the student's fee, in multiple different parts/sections of the program then I will consolidate the code into a function and simply call the function whenever or wherever I need to use it. This will help me save time in writing the large chunk of code that performs the summation in multiple parts of the program.

2. Discuss the role of object-oriented programming concepts such as classes, inheritance, and encapsulation in organizing and managing the data processing tasks.

Object-Oriented Programming allows us to use the concepts of “Objects” in our code that allows us to consolidate code and make blueprints of entities containing multiple attributes/data types.

Classes, Inheritance and Encapsulation:

For example I can create a super class called “Person” and make it have attributes such as “Name”, “Age”, “Gender”, “Blood Type”. The class will also have its member functions that will be responsible for accessing the private attributes in the class. The reason for making the attributes private is so that they cannot be accessed outside the class. I can then create child classes that would inherit the super class. The child classes can be “Student” and “Faculty”. The child classes will have their own respective attributes such as “Registration Id”, “Department Name” for Student class, and “Id Number”, “Department Name”, “Role” and etc. The faculty class can also be further inherited by a number of classes too, such as “Professors”, “Peons”, “Non-Professors” and etc. This allows us to not to create same functions that each individual classes will have multiple times. Every class needs a “Name”, “Age”, “Gender” and “Blood Type”, and their respective setters and getter functions.

3. Explore how advanced Python topics like generators and decorators can enhance the efficiency of the program.

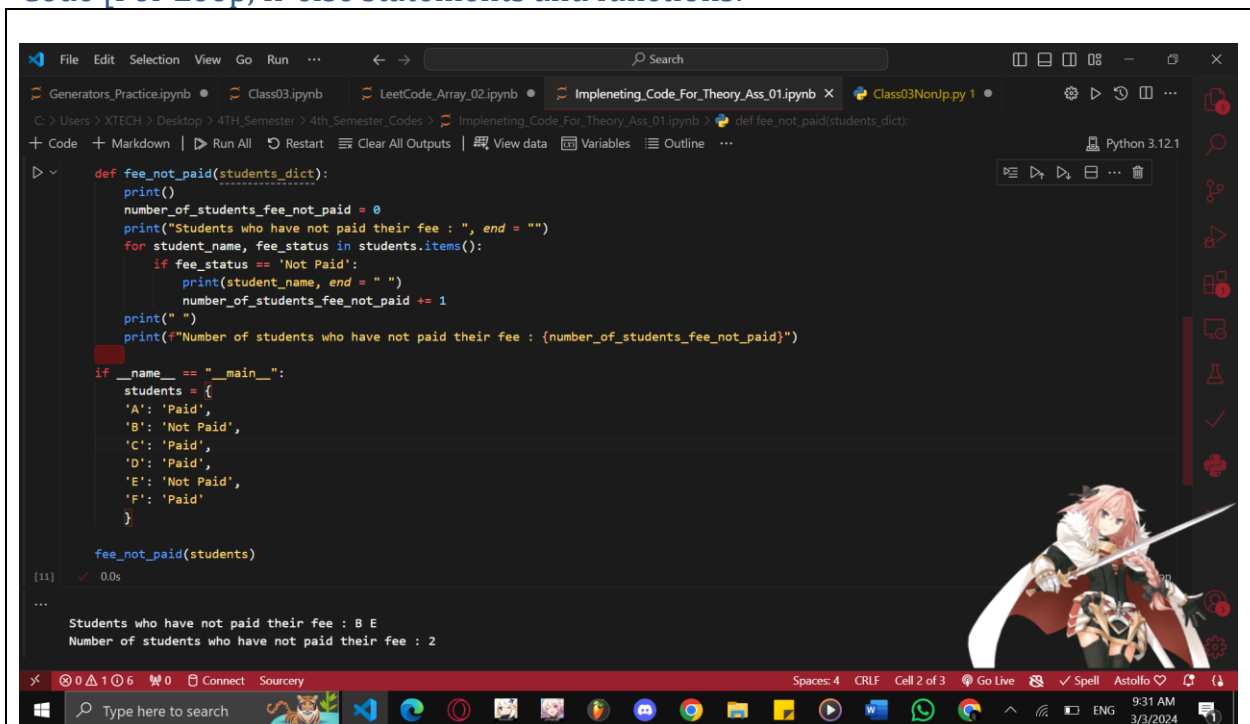
Generators are great when we want to iterate through something once by once, without loading the entire thing at once and iterating through it. This helps in memory efficiency since the CPU does not have to consume excess memory. An example would be that if I am given a dataset for SZABIST’s student’s fee status, and I am asked to check who have not yet paid the fee. Normally I would load the entire dataset and run a loop to iterate through it, but this consumes a lot of memory. A more memory efficient way to do this is to use a generator, which will not load the entire dataset and iterate through it, it will instead iterate through the rows one by one and only loading the data that is relevant, in this case the students whose fee status is “Not Paid”’s data will be loaded only.

Decorators can be used to modify or in override standalone functions. The reason for this is that often times you might need to change the functionality of a function that is not part of a class, so you will use a decorator to encapsulate the function. Decorators do not change the core logic of the function they instead enhance it, whereas function overriding usually changes the entire core logic. An example of this would be if I need to do validation checks of a dataset, so before the function to perform tasks on the data set is executed, it is encapsulated into a decorator which first validates whether the dataset is clean or not, or has the right data types for its attributes that are needed to be passed. The decorator performs the validation check and then executes the original function.

4. Provide a brief outline of how you would structure and implement such a program.

I will be using example codes of how I would structure or implement the program

Code [For Loop, If-else statements and functions:



```
def fee_not_paid(students_dict):
    print()
    number_of_students_fee_not_paid = 0
    print("Students who have not paid their fee : ", end = "")
    for student_name, fee_status in students.items():
        if fee_status == 'Not Paid':
            print(student_name, end = " ")
            number_of_students_fee_not_paid += 1
    print(" ")
    print(f"Number of students who have not paid their fee : {number_of_students_fee_not_paid}")

if __name__ == "__main__":
    students = {
        'A': 'Paid',
        'B': 'Not Paid',
        'C': 'Paid',
        'D': 'Paid',
        'E': 'Not Paid',
        'F': 'Paid'
    }

    fee_not_paid(students)

[11] ✓ 0.0s
...
Students who have not paid their fee : B E
Number of students who have not paid their fee : 2
```

```
File Edit Selection View Go Run ... Search
Generators_Practice.ipynb Class03.ipynb LeetCode_Array_02.ipynb Implemeting_Code_For_Theory_Ass_01.ipynb Class03NonIp.py 1 Python 3.12.1
C:\Users\XTECH\Desktop> 4TH_Semester > 4th_Semester_Codes > Implemeting_Code_For_Theory_Ass_01.ipynb > class Person:
+ Code + Markdown | Run All Restart Clear All Outputs View data Variables Outline ...

class Person:
    """
    Super Class
    """
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, new_name):
        self.__name = new_name

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, new_age):
        if new_age < 0:
            raise ValueError("Age cannot be negative.")
        self.__age = new_age
```

Code:

```
File Edit Selection View Go Run ... Search
Generators_Practice.ipynb Class03.ipynb LeetCode_Array_02.ipynb Implemeting_Code_For_Theory_Ass_01.ipynb Class03NonIp.py 1 Python 3.12.1
C:\Users\XTECH\Desktop> 4TH_Semester > 4th_Semester_Codes > Implemeting_Code_For_Theory_Ass_01.ipynb > class Person:
+ Code + Markdown | Run All Restart Clear All Outputs View data Variables Outline ...

def introduce(self):
    print(f"Hi, I'm {self.name} and I'm {self.age} years old.")

class Student(Person):
    """
    Child Class of Person
    """
    def __init__(self, name, age, major):
        super().__init__(name, age)
        self.__major = major

    @property
    def major(self):
        return self.__major

    @major.setter
    def major(self, new_major):
        self.__major = new_major

    def study(self):
        print(f"{self.name} is diligently studying {self.major}.")

class Faculty(Person):
    """
    Child Class of Person
    """
    def __init__(self, name, age, department):
        super().__init__(name, age)
```

```

File Edit Selection View Go Run ... Search
Generators_Practice.ipynb • Class03.ipynb • LeetCode_Array_02.ipynb • Implemeting_Code_For_Theory_Ass_01.ipynb • Class03NonIp.py 1 •
C:\Users\XTECH\Desktop\4TH_Semester\4th_Semester_Codes> Implemeting_Code_For_Theory_Ass_01.ipynb class Person:
+ Code + Markdown | ▶ Run All ▶ Restart Clear All Outputs | View data Variables Outline ... Python 3.12.1

@property
def department(self):
    return self.__department

@department.setter
def department(self, new_department):
    self.__department = new_department

def teach(self):
    print(f"{self.name} is teaching a class in the {self.department} department.")

class Professor(Faculty):
    """
    Child Class of Faculty
    """
    def __init__(self, name, age, department, research_area):
        super().__init__(name, age, department)
        self.__research_area = research_area

    @property
    def research_area(self):
        return self.__research_area

    @research_area.setter
    def research_area(self, new_research_area):
        self.__research_area = new_research_area

    def conduct_research(self):

```

Code:

```

File Edit Selection View Go Run ... Search
Generators_Practice.ipynb • Class03.ipynb • LeetCode_Array_02.ipynb • Implemeting_Code_For_Theory_Ass_01.ipynb • Class03NonIp.py 1 •
C:\Users\XTECH\Desktop\4TH_Semester\4th_Semester_Codes> Implemeting_Code_For_Theory_Ass_01.ipynb class Person:
+ Code + Markdown | ▶ Run All ▶ Restart Clear All Outputs | View data Variables Outline ... Python 3.12.1

def conduct_research(self):
    print(f"{self.name} is conducting research in {self.research_area}.")

class Peon(Faculty):
    """
    Child Class of Faculty
    """
    def __init__(self, name, age, department, role):
        super().__init__(name, age, department)
        self.__role = role

    @property
    def role(self):
        return self.__role

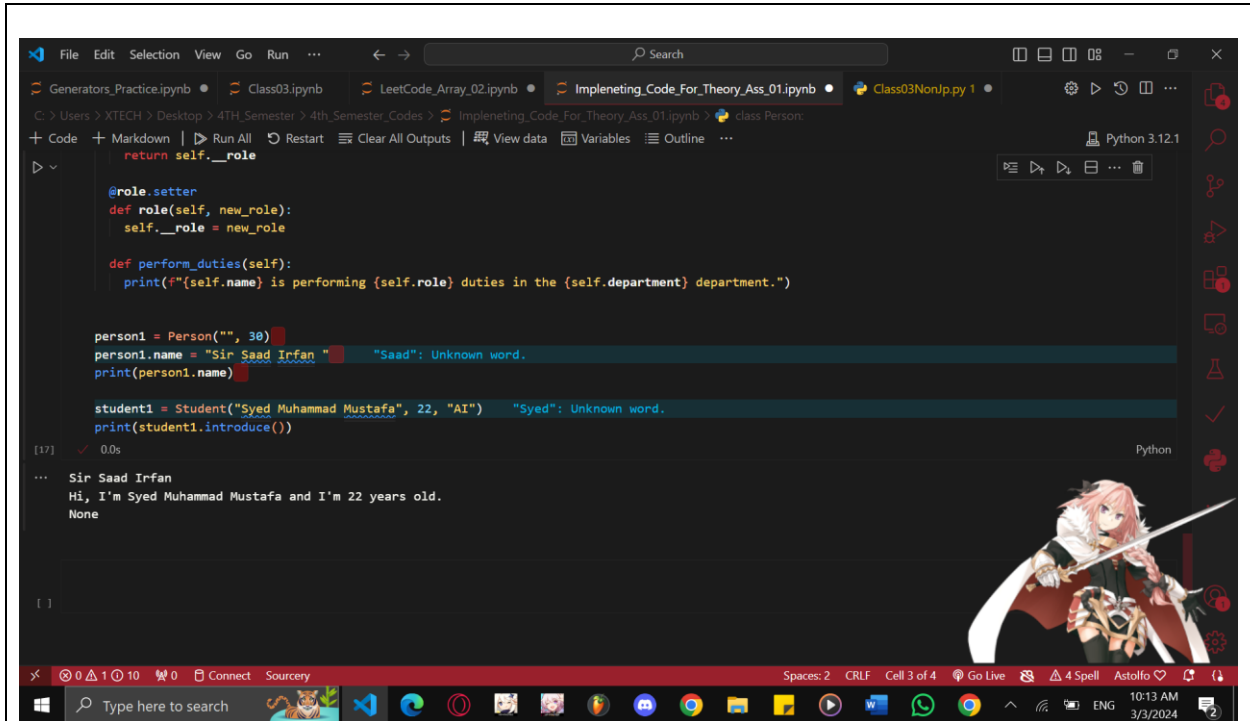
    @role.setter
    def role(self, new_role):
        self.__role = new_role

    def perform_duties(self):
        print(f"{self.name} is performing {self.role} duties in the {self.department} department.")

person1 = Person("", 30)
person1.name = "Sir Saad Irfan" "Saad": Unknown word.
print(person1.name)

student1 = Student("Syed Muhammad Mustafa", 22, "AI") "Syed": Unknown word.
print(student1.introduce())

```



```
File Edit Selection View Go Run ... Search
Generators_Practice.ipynb Class03.ipynb LeetCode_Array_02.ipynb Impleneting_Code_For_Theory_Ass_01.ipynb Class03Non/p.py 1
C:\Users\XTECH\Desktop\4TH_Semester\4th_Semester_Codes> Impleneting_Code_For_Theory_Ass_01.ipynb class Person:
+ Code + Markdown | Run All Restart Clear All Outputs View data Variables Outline ...
return self.__role

@role.setter
def role(self, new_role):
    self.__role = new_role

def perform_duties(self):
    print(f"{self.name} is performing {self.role} duties in the {self.department} department.")

person1 = Person("", 30)
person1.name = "Sir Saad Irfan" "Saad": Unknown word.
print(person1.name)

student1 = Student("Syed Muhammad Mustafa", 22, "AI") "Syed": Unknown word.
print(student1.introduce())

[17] ✓ 0.0s Python
... Sir Saad Irfan
Hi, I'm Syed Muhammad Mustafa and I'm 22 years old.
None

[ ]
```