ASSIGNMENT 6.3

D.Charmi

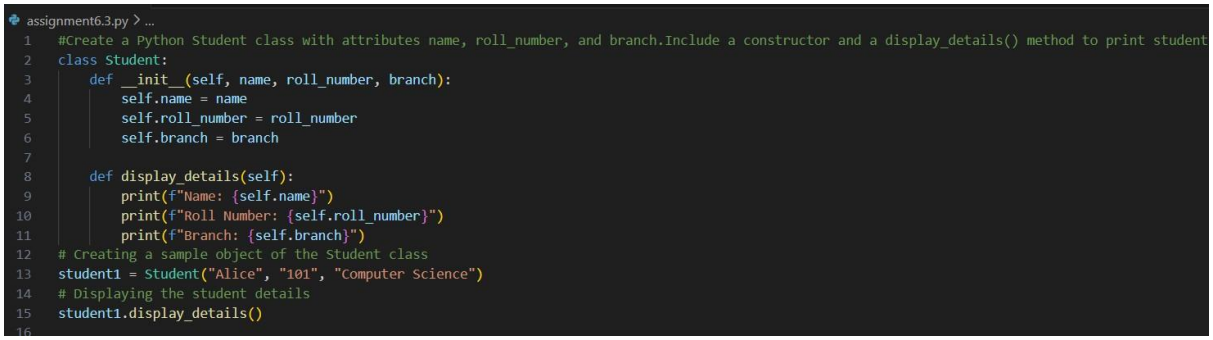
2303A51314

Batch – 05

TASK – 01 :

Prompt : Create a Python Student class with attributes name, roll_number, and branch. Include a constructor and a display_details() method to print student information. Create a sample object, run the code, show the output, and briefly analyze the code.

Code :



```python
#Create a Python Student class with attributes name, roll_number, and branch.Include a constructor and a display_details() method to print student
class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")
# Creating a sample object of the Student class
student1 = Student("Alice", "101", "Computer Science")
# Displaying the student details
student1.display_details()
```

Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Cod
ing/assignment6.3.py"
Name: Alice
Roll Number: 101
Branch: Computer Science
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

This code defines a Student class to store and manage student information. The __init__ method initializes the student's name, roll number, and branch when an object is created. The display_details() method prints these details in a readable format. Finally, a sample Student object is created and its details are displayed on the console, verifying that the class works correctly.

TASK – 02 :

Prompt : Write a Python function that displays first ten multiples of a given number using a loop. Show the output and explain the logic briefly. Then implement the same function using a different looping method**.**

Code :

```python
#Write a Python function that displays first ten multiples of a given number using a loop. Show the output and explain the logic briefly.Then implem
def multiples_for_loop(number):
    print(f"First ten multiples of {number} using for loop:")
    for i in range(1, 11):
        print(number * i)
def multiples_while_loop(number):
    print(f"First ten multiples of {number} using while loop:")
    i = 1
    while i <= 10:
        print(number * i)
        i += 1
# Example usage
num = 5
multiples_for_loop(num)
multiples_while_loop(num)

```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Cod
ing/assignment6.3.py"
First ten multiples of 5 using for loop:
5
10
15
20
25
30
35
40
45
50
First ten multiples of 5 using while loop:
5
10
15
20
25
30
35
40
45
50
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

- In the first function 'multiples_for_loop', we use a for loop that iterates from 1 to 10.
- For each iteration, we multiply the given number by the loop variable 'i' and print the result.
- In the second function 'multiples_while_loop', we use a while loop that continues as long as 'i' is less than or equal to 10.
- We initialize 'i' to 1 and increment it by 1 in each iteration, printing the product of the number and 'i'.

Comparison of Approaches:

- The for loop is more concise and easier to read when the number of iterations is known beforehand.
- The while loop provides more flexibility for scenarios where the number of iterations may not be predetermined.
- Both approaches effectively achieve the same result, but the choice between them can depend on the specific use case and personal preference.

TASK – 03 :

Prompt : Create a Python function that categorizes a person based on age using conditional statements. Explain how the conditions work, then show another way to implement the same logic in a simpler or alternative form.

Code :

```python
#Create a Python function that categorizes a person based on age using conditional statements.Explain how the conditions work, then show another way
def categorize_age(age):
    if age < 0:
        return "Invalid age"
    elif age <= 12:
        return "Child"
    elif age <= 19:
        return "Teenager"
    elif age <= 64:
        return "Adult"
    else:
        return "Senior"
def categorize_age_alternative(age):
    categories = {
        "Invalid age": lambda x: x < 0,
        "Child": lambda x: 0 <= x <= 12,
        "Teenager": lambda x: 13 <= x <= 19,
        "Adult": lambda x: 20 <= x <= 64,
        "Senior": lambda x: x >= 65
    }

    for category, condition in categories.items():
        if condition(age):
            return category
# Example usage:
print(categorize_age(10))  # Output: Child
print(categorize_age(16))  # Output: Teenager
print(categorize_age(30))  # Output: Adult
print(categorize_age(70))  # Output: Senior
print(categorize_age_alternative(10))  # Output: Child
print(categorize_age_alternative(16))  # Output: Teenager
print(categorize_age_alternative(30))  # Output: Adult
print(categorize_age_alternative(70))  # Output: Senior
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Co
ing/assignment6.3.py"
Child
Teenager
Adult
Senior
Child
Teenager
Adult
Senior
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

The function checks the age against a series of conditions:

1. If the age is less than 0, it returns "Invalid age".
2. If the age is between 0 and 12 (inclusive), it returns "Child".
3. If the age is between 13 and 19 (inclusive), it returns "Teenager".
4. If the age is between 20 and 64 (inclusive), it returns "Adult".
5. If the age is 65 or older, it returns "Senior".

Alternative implementation using a dictionary and a loop

The alternative implementation uses a dictionary to map categories to their respective conditions. It iterates through the dictionary and applies each condition to the age until it finds a match.

TASK – 04 :

Prompt : Write a Python function that calculates the total of numbers from 1 to n using a loop. Explain the logic, then show another way to achieve the same result using a different method.

Code :

```python
#Write a Python function that calculates the total of numbers from 1 to n using a loop.
#Explain the logic, then show another way to achieve the same result using a different method.
def sum_numbers_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Logic: The function initializes a variable 'total' to 0. It then iterates through each number from 1 to n (inclusive) using a for loop. In each it
def sum_numbers_formula(n):
    return n * (n + 1) // 2
# Another way to achieve the same result is by using the mathematical formula for the sum of the first n natural numbers, which is n(n + 1)/2. This
# Example usage:
n = 10
print("Sum using loop:", sum_numbers_loop(n))
print("Sum using formula:", sum_numbers_formula(n))

```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Cod
ing/assignment6.3.py"
Sum using loop: 55
Sum using formula: 55
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

This program calculates the sum of numbers from 1 to *n* in two ways. The first function uses a loop to add each number one by one until *n*. The second function uses a mathematical formula to compute the sum directly. Both methods give the same result, but the formulabased approach is faster since it avoids looping.

TASK – 05 :

Prompt : Create a Python class for a bank account with methods to add money, remove money, and display the current balance. Show example usage, include comments in the code.

Code :

```python
#Create a Python class for a bank account with methods to add money, remove money, and display the current balance.
#Show example usage, include comments in the code.

class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def add_money(self, amount):
        self.balance += amount

    def remove_money(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient funds")

    def display_balance(self):
        print(f"Current balance: ${self.balance}")

# Example usage:
account = BankAccount(100)
account.display_balance()  # Output: Current balance: $100
account.add_money(50)
account.display_balance()  # Output: Current balance: $150
account.remove_money(30)
account.display_balance()  # Output: Current balance: $120
```

Output :

Explanation :

This code defines a BankAccount class to manage basic banking operations. The constructor initializes the account with an initial balance. The add_money() method increases the balance when money is deposited. The remove_money() method subtracts money only if sufficient balance is available; otherwise, it displays an error message. The display_balance() method shows the current account balance. The example usage demonstrates depositing and withdrawing money and how the balance updates after each operation.