Assignment-6.3

Task-1

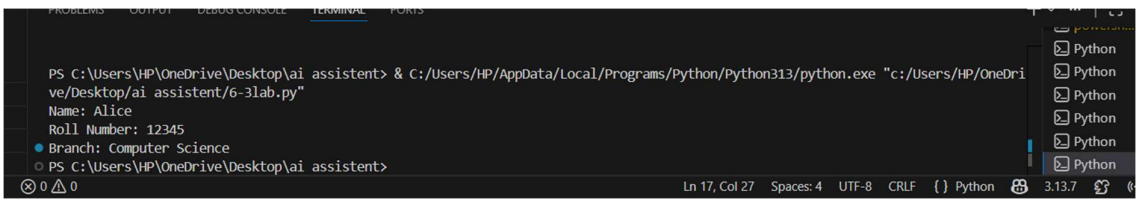
Prompt:

Generate a Python class named Student with attributes name, roll number, and branch. Include a constructor to initialize these attributes. Add a method display details that prints the student's information clearly. Create a sample object and call the method to show output.

Code:

```
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}")

# Create a sample object
student1 = Student("Alice", "12345", "Computer Science")

# Call the method to display student details
student1.display_details()
```

Output:



Analysis:

The AI-generated class correctly uses a constructor to initialize student details and a method to display them. The code is clear and demonstrates basic object-oriented concepts effectively.

Task-2

Prompt:

1)
Write a Python function that takes a number as input and prints the first 10 multiples of that number using a for loop.

2)

Now generate the same logic using a while loop instead of a for loop to print the first 10 multiples of a number.

Code:

1)

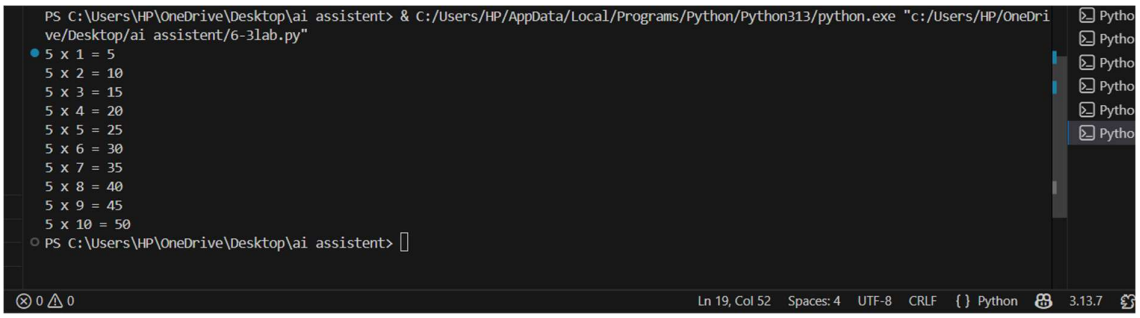
```
def print_multiples(number):
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")
# Example usage:
print_multiples(5)
```

2)

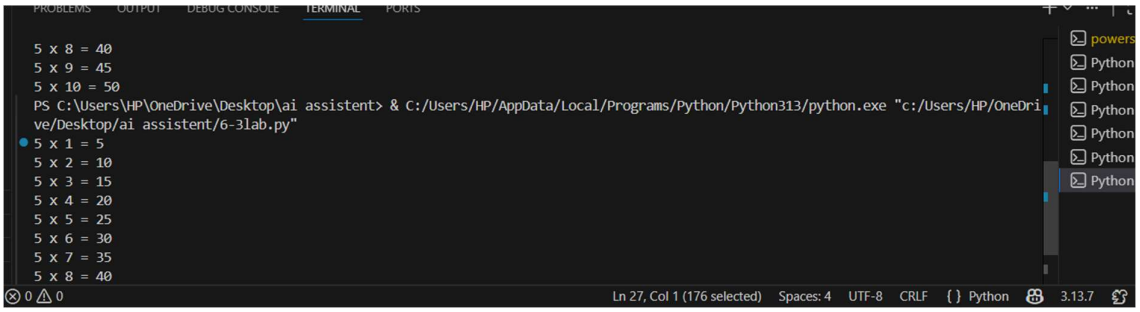
```
def print_multiples_while(number):
    i = 1
    while i <= 10:
        print(f"{number} x {i} = {number * i}")
        i += 1
# Example usage:
print_multiples_while(5)
```

Output:

1)



2)



Analysis:

1)

The for loop efficiently prints the first 10 multiples of a number. The logic is simple and works correctly.

2)

The while loop produces the same output but requires manual counter control. It is flexible but slightly less convenient than the for loop.

Task-3

Prompt:

1)

generate a Python function that takes input as age and classifies it into categories such as child, teenager, adult, and senior using nested if-elif-else statements.

2)

Rewrite the age classification program using simplified conditions or a dictionary-based approach and explain how it works.

Code:

1)

```
def classify_age(age):
    if age < 0:
        return "Invalid age"
    elif age <= 12:
        return "Child"
    elif age <= 19:
        return "Teenager"
    elif age <= 59:
        return "Adult"
    else:
        return "Senior"
# Example usage:
age = int(input("Enter your age: "))
category = classify_age(age)
print(f"You are classified as: {category}")
```

2)

```
def classify_age_dict(age):
    age_groups = {
        (0, 12): "Child",
        (13, 19): "Teenager",
        (20, 59): "Adult",
        (60, float('inf')): "Senior"
    }
    for (min_age, max_age), category in age_groups.items():
        if min_age <= age <= max_age:
            return category
    return "Invalid age"
```

Example usage:

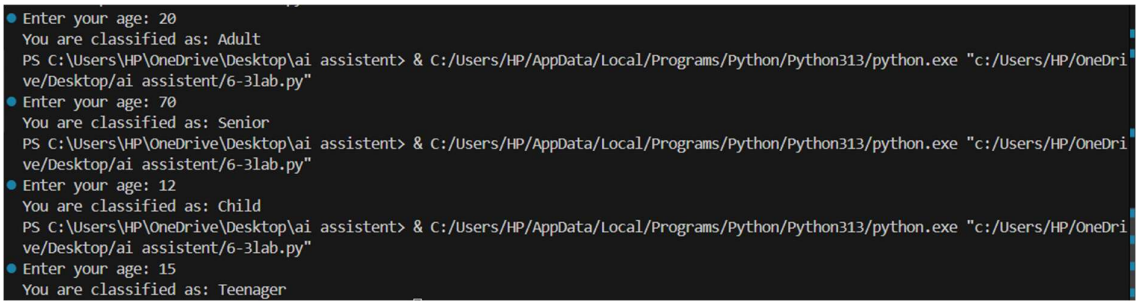
age = int(input("Enter your age: "))

category = classify_age_dict(age)

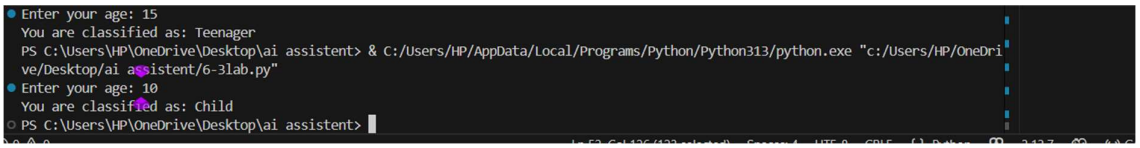
print(f"You are classified as: {category}")

Output:

1)



2)



Analysis:

1)

The conditional statements correctly classify age groups with proper logic and handle invalid input.

2)

This method is more compact and efficient, using ranges for classification. It shows an advanced alternative to traditional conditionals.

Task-4

Prompt:

1)

generate a Python function that calculates the sum of the first n natural numbers using a for loop and prints the result.

2)

Provide an alternative implementation of sum to n(n) using a while loop and also using a mathematical formula. Compare the approaches

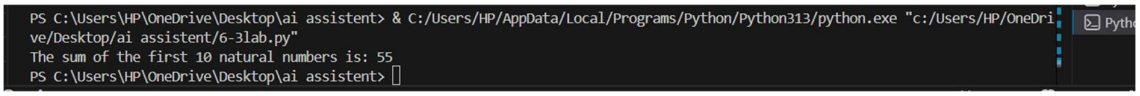
Code:

```
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    print(f"The sum of the first {n} natural numbers is: {total}")
# Example usage:
sum_to_n(10)
```

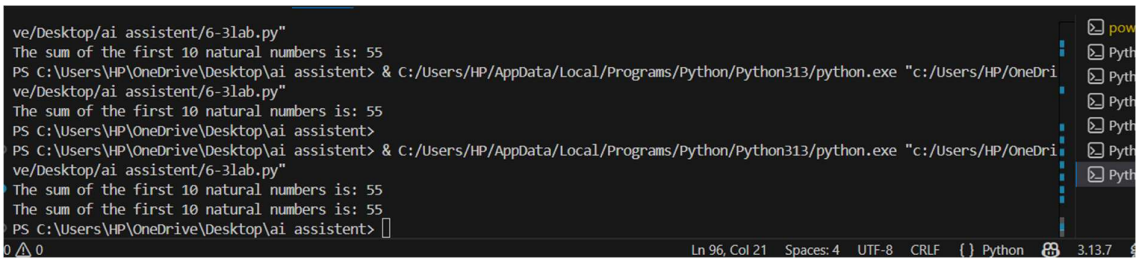
2)

```
def sum_to_n_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    print(f"The sum of the first {n} natural numbers is: {total}")


# Alternative using mathematical formula:
def sum_to_n_formula(n):
    total = n * (n + 1) // 2
    print(f"The sum of the first {n} natural numbers is: {total}")


# Example usage:
sum_to_n_while(10)
sum_to_n_formula(10)
```

Output:



2)



Analysis:

1)

The for loop correctly calculates the sum of first n numbers using simple iteration.

2)

The while loop works correctly, while the formula method is the fastest and most efficient.

Task-5

Prompt:

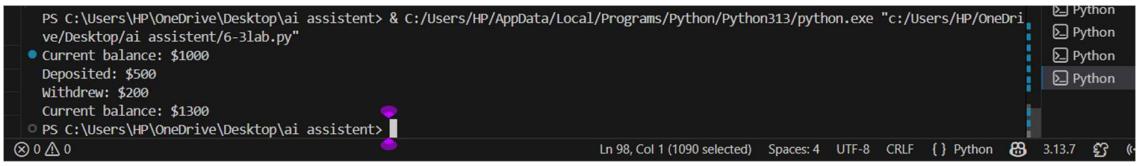
Generate a Python class called BankAccount with attributes such as account holder name and balance. Include methods: deposit(amount) withdraw(amount) check_balance() Demonstrate the class by creating an object and performing deposit and withdrawal operations. Add meaningful comments and explain how the class works.

Code:

```
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance
    def deposit(self, amount):
        """Deposit a specified amount into the account."""
        if amount > 0:
```

```
            self.balance += amount
            print(f"Deposited: ${amount}")
        else:
            print("Deposit amount must be positive.")
    def withdraw(self, amount):
        """Withdraw a specified amount from the account if sufficient balance exists."""
        if 0 < amount <= self.balance:
            self.balance -= amount


            print(f"Withdrew: ${amount}")
        else:
            print("Insufficient balance or invalid withdrawal amount.")
    def check_balance(self):
        """Check and return the current balance of the account."""
        print(f"Current balance: ${self.balance}")
# Demonstration of the BankAccount class
account = BankAccount("John Doe", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(200)
account.check_balance()
```

Output:



Analysis:

The class correctly handles deposits, withdrawals, and balance checks with clear structure and validation.