

**AI-Powered Coding Lab**

**Assignment 4**

**Enrollment No: 2503A51L44**

**Batch: 20**

**Student Name: Meer Burhan Ali Hashmi**

---

### **Task 1**

#### **Task Overview:**

Create a Python class named `BankAccount`. Begin by adding a descriptive class-level comment and partially writing the constructor with attributes for `account_holder` and `balance`. Then, use GitHub Copilot to auto-complete the rest of the class, ensuring it includes:

- A method to deposit a given amount.
  - A method to withdraw money, with proper error handling for insufficient balance.
  - A method to display the current account balance.
- 

#### **Prompt Provided:**

"Implement a Python class named `BankAccount` with a descriptive comment, a constructor receiving `account_holder` and `balance`, and methods for `deposit`, `withdraw` (with error handling), and `display_balance`. Include a complete implementation along with an example demonstrating its usage."

```

1 class BankAccount:
2     """
3     A class to represent a bank account.
4     Attributes:
5         account_holder (str): The name of the account holder.
6         balance (float): The current balance of the account.
7     Methods:
8         deposit(amount): Deposits the specified amount into the account.
9         withdraw(amount): Withdraws the specified amount if sufficient funds exist.
10        display_balance(): Displays the current balance.
11    """
12    def __init__(self, account_holder, balance=0.0):
13        """Initialize account holder and balance."""
14        self.account_holder = account_holder
15        self.balance = balance
16
17    def deposit(self, amount):
18        """Deposit the specified amount into the account."""
19        if amount > 0:
20            self.balance += amount
21            print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
22        else:
23            print("Deposit amount must be positive.")
24
25    def withdraw(self, amount):
26        """Withdraw the specified amount if sufficient funds exist."""
27        if amount > self.balance:
28            print("Insufficient funds.")
29        elif amount <= 0:
30            print("Withdrawal amount must be positive.")
31        else:
32            self.balance -= amount
33            print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
34
35    def display_balance(self):
36        """Display the current balance."""
37        print(f"Account Holder: {self.account_holder}, Balance: ${self.balance:.2f}")
38
39 if __name__ == "__main__":
40     # Example usage
41     account = BankAccount("Alice", 100.0)
42     account.display_balance()
43     account.deposit(50)
44     account.withdraw(30)
45     account.withdraw(150)
46     account.deposit(-10)
47     account.withdraw(-5)
48     account.display_balance()

```

OUTPUT:-

```

PS C:\Users\khaja\Downloads\A4.1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/A4.1/t1.py
Account Holder: Alice, Balance: $100.00
Deposited $50.00. New balance: $150.00
Withdrew $30.00. New balance: $120.00
Insufficient funds.
Deposit amount must be positive.
Withdrawal amount must be positive.
Account Holder: Alice, Balance: $120.00
PS C:\Users\khaja\Downloads\A4.1>

```

- Task 2

- Task Overview:

Begin by writing a comment and the first line of a for loop that iterates through a list of numbers. Then, use GitHub Copilot to auto-complete the remaining code so that it calculates the sum of all even numbers contained in the list.

- 

- Prompt Used:

"Write a Python comment and the initial for loop line to iterate over a list of numbers. Then, allow GitHub Copilot to complete the logic that sums all even numbers in the list. Provide a complete implementation along with an example."

```

1 # Iterate over the list and sum all even numbers
2 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3 sum_even = 0
4 for num in numbers:
5     if num % 2 == 0:
6         sum_even += num
7 print(f"Sum of even numbers: {sum_even}")
8

```

OUTPUT:-

```

PS C:\Users\khaja\Downloads\A4.1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/A4.1/t2.py
Sum of even numbers: 30
PS C:\Users\khaja\Downloads\A4.1>

```

### Task 3

#### Task Overview:

Begin creating a Python function that accepts an age value as input. The function should determine whether the person falls into one of four categories — **child**, **teenager**, **adult**, or **senior** — using if, elif, and else statements. Then, allow GitHub Copilot to complete the conditional logic.

---

#### Prompt Provided:

"Create a Python function that takes an age as input and returns whether the individual is a child, teenager, adult, or senior, using if-elif-else statements. Provide the complete implementation along with a clear usage example."

#### CODE:-

```

1 def age_group(age):
2     """Return the age group for a given age."""
3     if age < 13:
4         return "Child"
5     elif age < 20:
6         return "Teenager"
7     elif age < 60:
8         return "Adult"
9     else:
10        return "Senior"
11
12 if __name__ == "__main__":
13     # Clear example usage
14     ages = [5, 16, 35, 70]
15     for a in ages:
16         print(f"Age {a}: {age_group(a)}")
17

```

OUTPUT:-

```

PS C:\Users\khaja\Downloads\A4.1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/A4.1/t4.py
Age 5: Child
Age 16: Teenager
Age 35: Adult
Age 70: Senior
PS C:\Users\khaja\Downloads\A4.1>

```

## TASK4

**TASK4 DESCRIPTION:-** Auto-Complete a While Loop to Reverse Digits of a Number

Write a comment and start a while loop to reverse the digits of a number. Let Copilot complete the loop logic.

**PROMPT:-**Generate a Python comment and start a while loop to reverse the digits of a number. Then use GitHub Copilot to auto-complete the loop logic and implement it with example

**CODE:-**

```
1 # Reverse the digits of a number using a while loop
2 num = 12345
3 original_num = num # Save original for display
4 reversed_num = 0
5 while num > 0:
6     digit = num % 10
7     reversed_num = reversed_num * 10 + digit
8     num //= 10
9 print(f"Reversed digits of {original_num} is {reversed_num}")
10
```

**OUTPUT:-**

```
PS C:\Users\khaja\Downloads\A4.1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/A4.1/t4.py
Reversed digits of 12345 is 54321
PS C:\Users\khaja\Downloads\A4.1>
```

## TASK5

**TASK5 DESCRIPTION:-** Auto-Complete Class with Inheritance (Employee → Manager)

Begin a class Employee with attributes name and salary. Then, start a derived class Manager that inherits from Employee and adds a department. Let GitHub Copilot complete the methods and constructor chaining

**PROMPT:-** Write the start of a Python class Employee with attributes name and salary, then begin a derived class Manager that inherits from Employee and adds a department attribute. Let GitHub Copilot auto-complete the constructors and any necessary methods, including proper constructor chaining implement with example.

**CODE:-**

```

1  # Base class Employee with name and salary
2  class Employee:
3      def __init__(self, name, salary):
4          self.name = name
5          self.salary = salary
6
7      def __str__(self):
8          return f"Employee: {self.name}, Salary: {self.salary}"
9
10 # Derived class Manager inherits from Employee and adds department
11 class Manager(Employee):
12     def __init__(self, name, salary, department):
13         super().__init__(name, salary)
14         self.department = department
15
16     def __str__(self):
17         return f"Manager: {self.name}, Salary: {self.salary}, Department: {self.department}"
18
19 if __name__ == "__main__":
20     emp = Employee("John Doe", 50000)
21     mgr = Manager("Jane Smith", 80000, "HR")
22     print(emp)
23     print(mgr)
24

```

## OUTPUT:-

```

PS C:\Users\khaja\Downloads\A4.1> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/A4.1/t5.py
Employee: John Doe, Salary: 50000
Manager: Jane Smith, Salary: 80000, Department: HR
PS C:\Users\khaja\Downloads\A4.1>

```

**OBSERVATION:-** I observed how GitHub Copilot can be effectively used to auto-complete Python code when given the correct prompts. By providing only the initial structure such as a class definition, function header, or loop starter, Copilot was able to generate complete implementations with logical flow.

- In **Task 1**, I noticed that Copilot could generate a full Python class with constructor, methods, and proper error handling just from a descriptive prompt.
- In **Task 2**, it correctly completed the loop logic to filter and sum even numbers, showing its ability to understand conditional iteration.
- In **Task 3**, the function for age classification highlighted how Copilot handles nested conditionals and returns meaningful results.
- In **Task 4**, the while loop to reverse digits showed Copilot's capability to handle mathematical logic inside loops.
- In **Task 5**, I observed how Copilot implements object-oriented concepts like inheritance and constructor chaining effectively.