**HTNO**:-2403A52066

**ASSIGNMENT – 6.4**

**Task Description #1:**

Start a Python class named Student with attributes name, roll\_number, and marks. Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average.

# CODE

class Student:

def \_\_init\_\_(self, name, roll\_number, marks):

self.name = name self.roll\_number = roll\_number self.marks = marks def display\_details(self): """Display student details""" print(f"Name: {self.name}") print(f"Roll Number: {self.roll\_number}") print(f"Marks: {self.marks}") def is\_passed(self, average=40):

"""Check if student passed based on average marks""" if self.marks >= average:

return True else:

return False

# Example usage student1 = Student("Alice", 101, 75) student2 = Student("Bob", 102, 35) student1.display\_details() print("Passed:", student1.is\_passed()) print() student2.display\_details() print("Passed:", student2.is\_passed())

# OUTPUT

Name: Alice

Roll Number: 101

Marks: 75

Passed: True

Name: Bob

Roll Number: 102

Marks: 35

Passed: False

**Task Description #2:**

Write the first two lines of a for loop to iterate through a list of numbers. Use a comment prompt to let Copilot suggest how to calculate and print the square of even numbers only.

## CODE

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] for num in numbers:

# calculate and print the square of even numbers only if num % 2 == 0:

print(f"The square of {num} is {num \*\* 2}")

## OUTPUT

The square of 2 is 4

The square of 4 is 16

The square of 6 is 36

The square of 8 is 64

The square of 10 is 100

**Task Description #3:**

Create a class called BankAccount with attributes account\_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance. **CODE**

class BankAccount: def \_\_init\_\_(self, account\_holder, balance=0.0):

self.account\_holder = account\_holder self.balance = balance def deposit(self, amount):

"""Add amount to the account if it's positive.""" if amount > 0:

self.balance += amount print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")

else:

print("Deposit amount must be positive.") def withdraw(self, amount):

"""Withdraw amount from the account if there is sufficient balance and amount is positive.""" if amount <= 0:

print("Withdrawal amount must be positive.") elif amount > self.balance: print("Insufficient funds.") else:

self.balance -= amount print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}") def display\_balance(self):

"""Display the account holder and current balance.""" print(f"Account Holder: {self.account\_holder}, Balance: ${self.balance:.2f}")

# OUTPUT

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.

**Task Description #4:**

Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75. **CODE**

students = [

{"name": "Alice", "score": 82},

{"name": "Bob", "score": 67},

{"name": "Charlie", "score": 91},

{"name": "David", "score": 73},

{"name": "Eva", "score": 88}

] i = 0 while i < len(students): if students[i]["score"] > 75:

print(f"{students[i]['name']} scored {students[i]['score']}") i += 1

# OUTPUT

Alice scored 82

Charlie scored 91

Eva scored 88

**Task Description #5:**

Begin writing a class ShoppingCart with an empty items list. Prompt Copilot to generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts. **CODE**

class ShoppingCart:

def \_\_init\_\_(self):

self.items = [] # empty list to hold items (each item as dict with name & price) def add\_item(self, name, price): """Add an item to the shopping cart.""" self.items.append({"name": name, "price": price}) print(f"Added {name} for ${price:.2f}") def remove\_item(self, name):

"""Remove an item by name from the shopping cart."""

for item in self.items: if item["name"].lower() == name.lower():

self.items.remove(item) print(f"Removed {name}")

return

print(f"{name} not found in cart.") def calculate\_total(self):

"""Calculate total with conditional discounts."""

total = 0 for item in self.items: total += item["price"]

# Apply discounts based on total value

if total > 500:

discount = 0.20 # 20% discount elif total > 200:

discount = 0.10 # 10% discount

else:

discount = 0.0 # no discount discounted\_total = total - (total \* discount) print(f"Total before discount: ${total:.2f}") print(f"Discount applied: {discount \* 100:.0f}%") print(f"Final total: ${discounted\_total:.2f}") return discounted\_total

# OUTPUT

Added Shoes for $250.00

Added Shirt for $150.00

Added Watch for $220.00

Removed Shirt

Total before discount: $470.00

Discount applied: 10%

Final total: $423.00