# ASSIGNMENT_6.4

## Course: AI Assisted Coding

## Course Code: 23CS002PC304.

## NAME : K.ABHINAY

## ROLL NO. : 2303A51899

## BATCH NO. : 09

## LAB – 6:

AI-Based Code Completion – Classes, Loops, and Conditionals
Lab Objectives:

The objectives of this lab are to:
• To explore AI-powered auto-completion features for core Python constructs.
• To analyze how AI suggests logic for class definitions, loops, and conditionals.
• To evaluate the completeness and correctness of code generated by AI assistants.
Lab Outcomes (LOs):
• After completing this lab, students will be able to:
• Use AI tools to generate and complete class definitions and methods.
• Understand and assess AI-suggested loops for iterative tasks.
• Generate conditional statements through prompt-driven suggestions.
• Critically evaluate AI-assisted code for correctness and clarity.

# Task 1: Student Performance Evaluation System

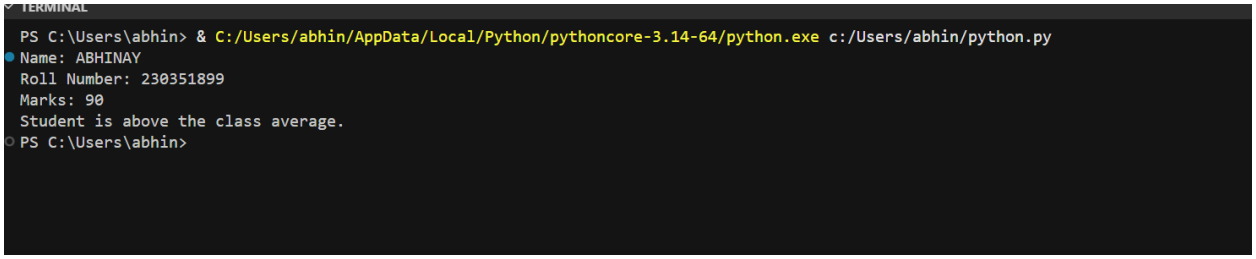
Scenario
You are building a simple academic management module for a university system where student performance needs to be evaluated automatically.
Task Description
Create the skeleton of a Python class named Student with the attributes:
• name
• roll_number
• marks
Write only the class definition and attribute initialization.
Then, using GitHub Copilot, prompt the tool to complete:
• A method to display student details
• A method that checks whether the student's marks are above the class average and returns an appropriate message
Use comments or partial method names to guide Copilot for code completion.
Expected Outcome
• A completed Student class with Copilot-generated methods
• Proper use of:
o self attributes
o Conditional statements (if-else)
• Sample output showing student details and performance status

# CODE :

```
# Student Performance Evaluation System

class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    # Method to display student details
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
```

```python
    # Method to check if marks are above class average
    def check_performance(self, class_average):
        if self.marks > class_average:
            return "Student is above the class average."
        else:
            return "Student is below the class average."


# Sample usage
s1 = Student("ABHINAY", 230351899, 90)
s1.display_details()
print(s1.check_performance(75))
```

# OUTPUT :




## Analysis :

This program creates a student class.
It stores name, roll number, and marks.
It checks if marks are above class average

# Task 2: Data Processing in a Monitoring System

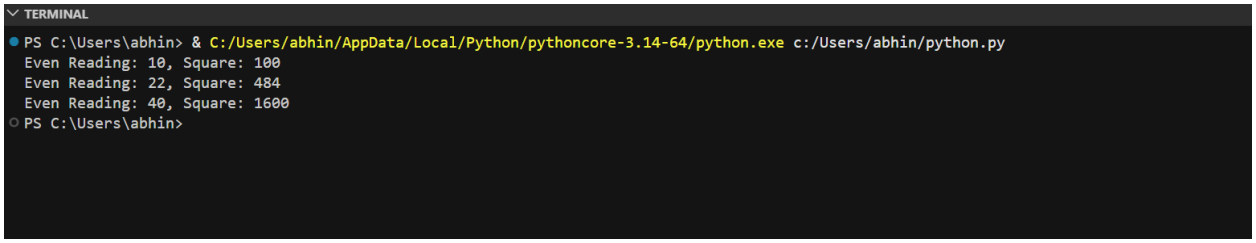
Scenario
You are working on a basic data monitoring script where sensor readings are collected as numbers. Only even readings need further processing.
Task Description
Write the initial part of a for loop to iterate over a list of integers

representing sensor readings.
Add a comment prompt instructing GitHub Copilot to:
• Identify even numbers
• Calculate their square
• Print the result in a readable format
Allow Copilot to complete the remaining loop logic.
Expected Outcome
• A complete for loop generated by Copilot
• Use of:
o Modulus operator to identify even numbers
o Conditional statements
• Correct and formatted output for valid inputs

# CODE :

```
# Sensor readings list
sensor_readings = [10, 15, 22, 33, 40, 55]

# Loop through sensor readings
for reading in sensor_readings:
    # Check if the number is even, calculate its square and print
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Even Reading: {reading}, Square: {square}")
```

# OUTPUT :




## Analysis :

This program takes a list of numbers.
It finds even numbers from the list.

It prints the square of each even number.

# Task 3: Banking Transaction Simulation

Scenario
You are developing a basic banking module that handles deposits and withdrawals for customers.
Task Description
Create the structure of a Python class named BankAccount with attributes:
• account_holder
• balance
Use GitHub Copilot to complete methods for:
• Depositing money
• Withdrawing money
• Preventing withdrawals when the balance is insufficient
Guide Copilot using method names and short comments.
Expected Outcome
• A fully functional BankAccount class
• Copilot-generated methods using:
o if-else conditions
o Class attributes via self
• Proper handling of invalid withdrawal attempts with user-friendly Messages

# CODE :

```
# Banking Transaction Simulation

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

    # Method to deposit money
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ₹{amount}. New balance: ₹{self.balance}")
```

```
    # Method to withdraw money with balance check
    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn ₹{amount}. Remaining balance:
₹{self.balance}")
        else:
            print("Insufficient balance. Withdrawal denied.")


# Sample usage
account = BankAccount("Nithin", 5000)
account.deposit(2000)
account.withdraw(3000)
account.withdraw(5000)
```

# OUTPUT :


## Analysis :

This program creates a bank account system.
It allows deposit and withdrawal of money.
It stops withdrawal if balance is not enough.

# Task 4: Student Scholarship Eligibility Check

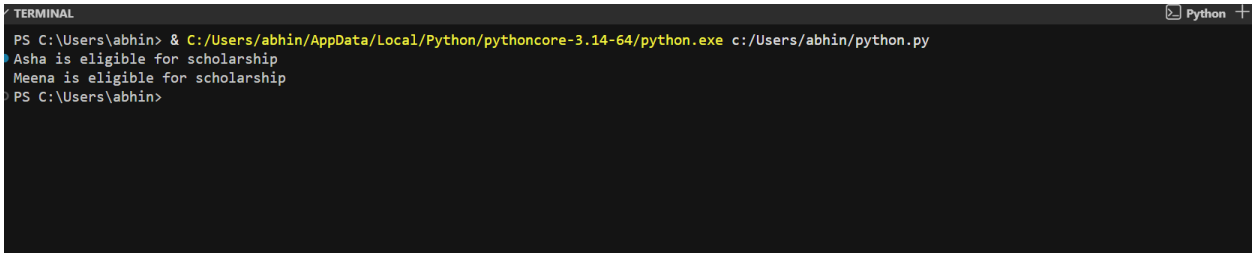
Scenario
A university wants to identify students eligible for a merit-based scholarship based on their scores.
Task Description
Define a list of dictionaries where each dictionary represents a student with:
• name

• score
Write the initialization and list structure yourself.
Then, prompt GitHub Copilot to generate a while loop that:
• Iterates through the list
• Prints the names of students who scored more than 75
Use comments to guide Copilot’s code completion.
Expected Outcome
• A complete while loop generated by Copilot
• Correct index handling and condition checks
• Cleanly formatted output listing eligible students

# CODE :

```
# List of students with scores

students = [

    {"name": "Asha", "score": 80},

    {"name": "Ravi", "score": 72},

    {"name": "Meena", "score": 90},

    {"name": "Kiran", "score": 68}

]



# While loop to check scholarship eligibility

index = 0

while index < len(students):

    # Print names of students scoring more than 75

    if students[index]["score"] > 75:

        print(students[index]["name"], "is eligible for scholarship")

    index += 1
```

# OUTPUT :




## Analysis :

This program stores student scores in a list.
It checks scores using a loop.
It prints names of students who get scholarship.

# Task 5: Online Shopping Cart Module

Scenario
You are designing a simplified shopping cart system for an e-commerce website that supports item management and discount calculation.
Task Description
Begin writing a Python class named ShoppingCart with:
• An empty list to store items (each item may include name, price, quantity)
Use GitHub Copilot to generate methods that:
• Add items to the cart
• Remove items from the cart
• Calculate the total bill using a loop
• Apply conditional discounts (e.g., discount if total exceeds a certain amount)
Use meaningful comments and method names to guide Copilot.
Expected Outcome
• A fully implemented ShoppingCart class
• Copilot-generated loops and conditional logic
• Correct handling of item addition, removal, and discount calculation
• Sample input/output demonstrating cart functionality

# CODE:

```
# Online Shopping Cart Module

class ShoppingCart:
    def __init__(self):
        self.items = []  # Each item: {name, price, quantity}

    # Method to add item to cart
    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity":
quantity})
        print(f"Added {name} to cart.")

    # Method to remove item from cart
    def remove_item(self, name):
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"Removed {name} from cart.")
                return
        print("Item not found in cart.")

    # Method to calculate total bill and apply discount
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]

        # Apply discount if total exceeds 1000
        if total > 1000:
            discount = total * 0.10
            total -= discount
            print(f"Discount applied: ₹{discount}")

        print(f"Total Bill Amount: ₹{total}")
        return total


# Sample usage
```

```python
cart = ShoppingCart()
cart.add_item("Laptop Bag", 800, 1)
cart.add_item("Mouse", 300, 2)
cart.calculate_total()
cart.remove_item("Mouse")
cart.calculate_total()
```

# OUTPUT :


## Analysis :

This program creates a shopping cart system.
It adds and removes items from the cart.
It calculates total bill and gives discount.