

AI ASSISTED CODING LAB

ASSIGNMENT 6

ENROLLMENT NO :2503A51L16

BATCH NO: 20

NAME: Kamera Jashuva

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.

PROMPT 1: Generate a program that Start a Python class named Student with attributes name, roll_number, and marks. Add methods to display the student details and to check if the marks are above average.

```

L1T1.PY x
L1T1.PY
1  # Student class represents a student with basic attributes and methods.
2
3  class Student:
4      def __init__(self, name, roll_number, marks):
5          self.name = name
6          self.roll_number = roll_number
7          self.marks = marks
8
9      def display_details(self):
10         """Display the student's details."""
11         print(f"Name: {self.name}")
12         print(f"Roll Number: {self.roll_number}")
13         print(f"Marks: {self.marks}")
14
15         def is_above_average(self, average=50):
16             """Check if marks are above average."""
17             return self.marks > average
18
19 # Example usage:
20 if __name__ == "__main__":
21     name = input("Enter student name: ")
22     roll_number = input("Enter roll number: ")
23     marks = float(input("Enter marks: "))
24
25     student = Student(name, roll_number, marks)
26     student.display_details()
27     if student.is_above_average():
28         print("Marks are above average.")
29     else:
30         print("Marks are not above average.")

```

```

PS C:\Users\Charan\OneDrive\Desktop\python> PYTHON L1T1.PY
Enter student name: CHARAN
Enter roll number: 38
Enter marks: 95
Name: CHARAN
Roll Number: 38
Marks: 95.0
Marks are above average.

```

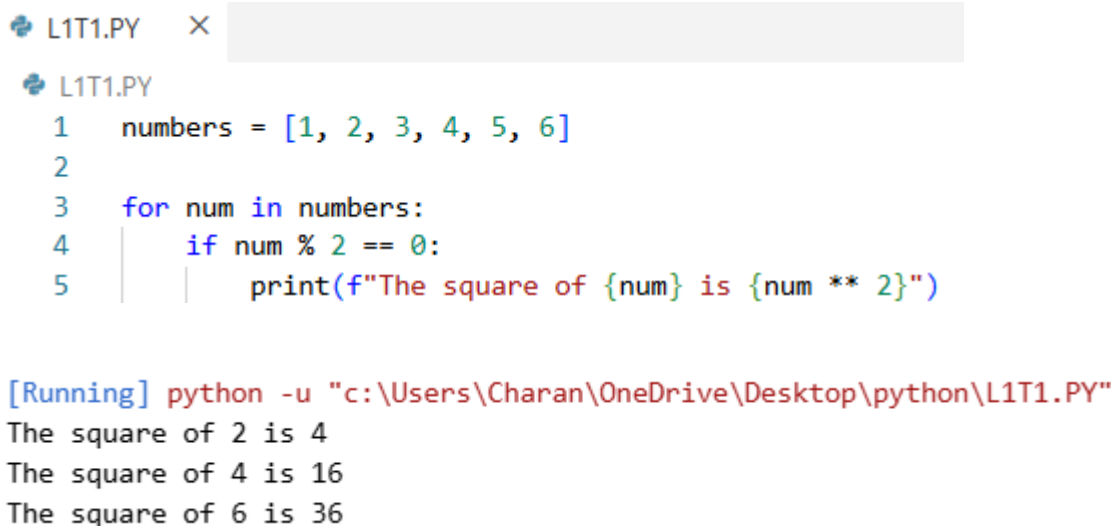
OBSERVATION: In this task, a Python class named Student was created with the attributes name, roll_number, and marks. Using GitHub Copilot, methods were generated to display the details of the student and to check whether the marks are above average. The prompt given guided Copilot to complete the class by adding the required methods. This demonstrates how AI-assisted coding tools can help in reducing development effort and improving productivity by quickly generating correct

and structured code.

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.

PROMPT 1: Generate a code to Create a list of numbers and write the first two lines of a for loop to iterate through that list. Add a comment inside the loop asking Copilot to suggest code that calculates and prints the square of even numbers only.



```
L1T1.PY X
L1T1.PY
1  numbers = [1, 2, 3, 4, 5, 6]
2
3  for num in numbers:
4      if num % 2 == 0:
5          print(f"The square of {num} is {num ** 2}")

[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
```

OBSERVATION: In this task, a list of numbers was created and the first two lines of a for loop were written to iterate through the list. A comment prompt was added inside the loop to guide GitHub Copilot in generating the code that calculates and prints the square of even numbers only. This showed how Copilot can interpret natural language comments and suggest the required logic, making the coding process more efficient and reducing manual effort.

TASK DESCRIPTION 3: Create a class called Bank Account with attributes `account_holder` and `balance`. Use Copilot to complete methods for `deposit ()`, `withdraw ()`, and check for insufficient balance.

PROMPT 1: Generate a python code to Create a Python class named Bank Account with attributes `account_holder` and `balance`. Ask Copilot to generate the methods `deposit ()` and `withdraw ()`, and to include a check that prevents withdrawal when the balance is insufficient.

```
L1T1.PY  X
L1T1.PY
1 class BankAccount:
2     def __init__(self, account_holder, balance=0):
3         self.account_holder = account_holder
4         self.balance = balance
5
6     def deposit(self, amount):
7         self.balance += amount
8         print(f"Deposited {amount}. New balance is {self.balance}")
9
10    def withdraw(self, amount):
11        if amount > self.balance:
12            print("Insufficient balance")
13        else:
14            self.balance -= amount
15            print(f"Withdrew {amount}. New balance is {self.balance}")
16
17    def get_balance(self):
18        return self.balance
19
20
21 # Example usage
22 account = BankAccount("Rahul", 1000)
23
24 account.deposit(500)    # Deposits 500
25 account.withdraw(200)  # Withdraws 200
26 account.withdraw(2000) # Attempt to withdraw more than balance
27 print(f"Current balance: {account.get_balance()}")
```

```
[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"
```

```
Deposited 500. New balance is 1500
```

```
Withdrew 200. New balance is 1300
```

```
Insufficient balance
```

```
Current balance: 1300
```

OBSERVATION: In this task, a Python class named Bank Account was created with the attributes `account_holder` and `balance`. Using the given prompt, GitHub Copilot was guided to generate the methods `deposit ()` and `withdraw ()`. Additionally, Copilot included a condition to check for

insufficient balance before allowing withdrawal. This demonstrates how Copilot can automate the creation of commonly used functionalities in object-oriented programming, ensuring correctness while saving time for the programmer.

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.

PROMPT 1: Generate a code to Define a list of student dictionaries where each dictionary contains the keys name and score. Ask Copilot to generate a while loop that prints the names of all students whose score is greater than 75.

```
L1T1.PY  X
L1T1.PY
1  # List of student dictionaries
2  students = [
3      {"name": "Rahul", "score": 80},
4      {"name": "Priya", "score": 70},
5      {"name": "Karan", "score": 90},
6      {"name": "Simran", "score": 65}
7  ]
8
9  # Initialize index
10 i = 0
11
12 # While loop to print names of students scoring more than 75
13 while i < len(students):
14     if students[i]["score"] > 75:
15         print(students[i]["name"])
16     i += 1
```

```
[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"
Rahul
Karan
```

OBSERVATION: In this task, a list of student dictionaries was defined with the keys name and score. GitHub Copilot was then prompted to generate a while loop that prints the names of students who scored more than 75. This activity highlighted how Copilot can understand structured data and apply conditional logic within loops to filter and display specific information, thereby simplifying the coding process.

TASK DESCRIPTION 5: Begin writing a class Shopping Cart 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.

PROMPT 1: Generate a code that Begin writing a Python class named ShoppingCart with an empty items list. Ask Copilot to generate methods add_item () and remove_item (), and to include a loop that calculates the total bill with conditional discounts applied.

```

L1T1.PY
1 class ShoppingCart:
2     def __init__(self):
3         self.items = [] # List to store items, each item is a dictionary
4
5     def add_item(self, name, price, quantity=1):
6         self.items.append({"name": name, "price": price, "quantity": quantity})
7         print(f"Added {quantity} x {name} at ${price} each to the cart.")
8
9     def remove_item(self, name):
10        original_count = len(self.items)
11        self.items = [item for item in self.items if item["name"] != name]
12        if len(self.items) < original_count:
13            print(f"Removed {name} from the cart.")
14        else:
15            print(f"{name} not found in the cart.")
16
17    def calculate_total(self):
18        total = 0
19        for item in self.items:
20            total += item["price"] * item["quantity"]
21
22        # Apply conditional discounts
23        if total > 1000:
24            total *= 0.9 # 10% discount
25        elif total > 500:
26            total *= 0.95 # 5% discount
27
28        return total
29
30
31 # Example usage
32 cart = ShoppingCart()
33 cart.add_item("Laptop", 700, 1)
34 cart.add_item("Headphones", 150, 2)
35 cart.add_item("Mouse", 50, 3)
36 cart.remove_item("Mouse")
37 print(f"Total bill: ${cart.calculate_total():.2f}")

```

[Running] python -u "c:\Users\Charan\OneDrive\Desktop\python\L1T1.PY"

```

Added 1 x Laptop at $700 each to the cart.
Added 2 x Headphones at $150 each to the cart.
Added 3 x Mouse at $50 each to the cart.
Removed Mouse from the cart.
Total bill: $950.00

```

OBSERVATION: In this task, a Python class named ShoppingCart was created with an empty items list. GitHub Copilot was prompted to generate the methods `add_item()` and `remove_item()`, and to implement a loop that calculates the total bill while applying conditional discounts. This exercise demonstrated how Copilot can assist in developing object-oriented programs with dynamic operations on lists and applying business logic, making the coding process more efficient and less error-prone.