AI ASSISTED CODING

LAB ASSIGNMENT-6.4

NAME : K.SUPRIYA

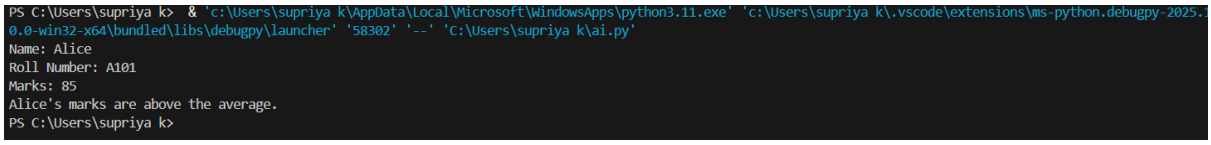
HALL TICKET NO : 2403A52395

BATCH : 14

TASK 1:

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

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    def is_above_average(self, average_marks):
        return self.marks > average_marks

# Example usage:
student1 = Student("Alice", "A101", 85)
student1.display_details()

average_marks = 70
if student1.is_above_average(average_marks):
    print(f"{student1.name}'s marks are above the average.")
else:
    print(f"{student1.name}'s marks are not above the average.")
```

OUTPUT:



EXPLANATION :

- **class Student:**: This line starts the definition of a new class named Student. Think of a class as a blueprint for creating objects (in this case, student objects).
- **def __init__(self, name, roll_number, marks):**: This is the constructor method. It's called when you create a new Student object.
    - self: Refers to the instance of the class being created.
    - name, roll_number, marks: These are the parameters you pass when creating a Student object.
    - Inside the constructor, self.name = name, self.roll_number = roll_number, and self.marks = marks assign the values passed in to the corresponding attributes of the Student object.
- **def display_details(self):**: This method is defined within the Student class.
    - self: Again, refers to the instance of the class.
    - It prints the name, roll number, and marks of the student using f-strings for formatted output.
- **def is_above_average(self, average_marks):**: This is another method in the Student class.
    - self: Refers to the instance of the class.
    - average_marks: This is a parameter representing the average marks to compare against.
    - It returns True if the student's marks are greater than average_marks, and False otherwise.
- **student1 = Student("Alice", "A101", 85)**: This line creates an instance (an object) of the Student class. It names the object student1 and initializes it with the name "Alice", roll number "A101", and marks 85.
- **student1.display_details()**: This line calls the display_details method on the student1 object, which prints the details of Alice.
- **average_marks = 70**: This line sets a variable average_marks to 70.

- **if student1.is_above_average(average_marks):**: This line calls the is_above_average method on student1 and passes the average_marks (70) as an argument. The if statement checks if the method returns True.
- **print(f"{student1.name}'s marks are above the average.")**: This line is executed if is_above_average returns True.
- **else:**: This indicates what to do if the if condition is false.
- **print(f"{student1.name}'s marks are not above the average.")**: This line is executed if is_above_average returns False.

In summary, the code defines a structure for representing a student and provides functions (methods) to interact with that data. The example usage demonstrates how to create a student, display their information, and check if their marks are above a given average.

TASK 2:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


for number in numbers:
  # Calculate and print the square of even numbers only
  if number % 2 == 0:
    square = number ** 2
    print(f"The square of {number} is {square}")
```

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
PS C:\Users\supriya k>
```

EXPLANATION:

- **numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**: This line creates a list named numbers containing integers from 1 to 10.
- **for number in numbers:**: This is a for loop. It iterates through each element in the numbers list. In each iteration, the current element is assigned to the variable number.
- **# Calculate and print the square of even numbers only**: This is a comment explaining the purpose of the following code block.

- **if number % 2 == 0:**: This is an if statement that checks if the current number is even.
    - The modulo operator (%) gives the remainder of a division.
    - If number % 2 is equal to 0, it means the number is perfectly divisible by 2, and therefore it is an even number.
- **square = number ** 2**: If the if condition is true (the number is even), this line calculates the square of the number by raising it to the power of 2 (** 2) and assigns the result to the variable square.
- **print(f"The square of {number} is {square}")**: If the if condition is true, this line prints a formatted string indicating the original even number and its calculated square.

In essence, the code goes through each number in the list, checks if it's even, and if it is, it computes and displays its square. Odd numbers are skipped.

TASK 3:

```
class BankAccount:
  def __init__(self, account_holder, balance=0):
    self.account_holder = account_holder
    self.balance = balance

  def deposit(self, amount):
    if amount > 0:
      self.balance += amount
      print(f"Deposited {amount}. New balance is {self.balance}")
    else:
      print("Deposit amount must be positive.")

  def withdraw(self, amount):
    if amount > 0:
      if self.balance >= amount:
        self.balance -= amount
```

```
            print(f"Withdrew {amount}. New balance is {self.balance}")
        else:
            print("Insufficient balance.")
    else:
        print("Withdrawal amount must be positive.")

  def check_balance(self):
    print(f"Current balance for {self.account_holder}: {self.balance}")

# Example usage:
account1 = BankAccount("Alice")
account1.check_balance()
account1.deposit(100)
account1.withdraw(50)
account1.withdraw(70)
```

OUTPUT:

EXPLANATION:

- **class BankAccount:**: This line starts the definition of the BankAccount class. This is the blueprint for creating bank account objects.
- **def __init__(self, account_holder, balance=0):**: This is the constructor method. It's called when you create a new BankAccount object.
    - self: Refers to the instance of the class being created.
    - account_holder: A parameter for the name of the account holder.
    - balance=0: A parameter for the initial balance, with a default value of 0 if not provided.

    - Inside the constructor, self.account_holder = account_holder and self.balance = balance assign the provided values to the corresponding attributes of the BankAccount object.

- **def deposit(self, amount):**: This method handles deposits into the account.
    - self: Refers to the instance of the class.
    - amount: The amount to deposit.
    - if amount > 0:: Checks if the deposit amount is positive.
    - self.balance += amount: If the amount is positive, it's added to the account's balance.
    - print(f"Deposited {amount}. New balance is {self.balance}"): Prints a confirmation message with the new balance.
    - else:: If the amount is not positive, it prints an error message.
- **def withdraw(self, amount):**: This method handles withdrawals from the account.
    - self: Refers to the instance of the class.
    - amount: The amount to withdraw.
    - if amount > 0:: Checks if the withdrawal amount is positive.
    - if self.balance >= amount:: If the amount is positive, this checks if there are sufficient funds.
    - self.balance -= amount: If there are sufficient funds, the amount is subtracted from the balance.
    - print(f"Withdrew {amount}. New balance is {self.balance}"): Prints a confirmation message with the new balance.
    - else:: If there are insufficient funds, it prints an "Insufficient balance" message.
    - else:: If the withdrawal amount is not positive, it prints an error message.
- **def check_balance(self):**: This method prints the current balance of the account.
    - self: Refers to the instance of the class.
    - print(f"Current balance for {self.account_holder}: {self.balance}"): Prints the account holder's name and their current balance.

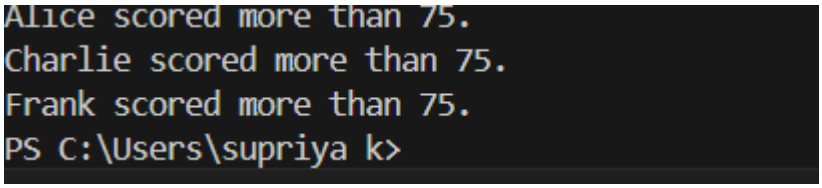
- **account1 = BankAccount("Alice")**: This line creates an instance of the BankAccount class named account1 with "Alice" as the account holder and the default balance of 0.
- **account1.check_balance()**: Calls the check_balance method on account1 to show the initial balance.
- **account1.deposit(100)**: Calls the deposit method on account1 to deposit 100.
- **account1.withdraw(50)**: Calls the withdraw method on account1 to withdraw 50.
- **account1.withdraw(70)**: Calls the withdraw method on account1 to withdraw 70. Since the balance is 50, this withdrawal will fail due to insufficient funds, and the corresponding message will be printed.
- **account1.check_balance()**: Calls the check_balance method again to show the final balance after the attempted transactions.

In summary, this code defines a basic bank account object with methods for managing deposits and withdrawals while ensuring that withdrawals do not exceed the available balance. The example usage demonstrates how these methods work in practice.

TASK:4

```
students = [
  {"name": "Alice", "score": 85},
  {"name": "Bob", "score": 70},
  {"name": "Charlie", "score": 92},
  {"name": "David", "score": 75},
  {"name": "Eve", "score": 68},
  {"name": "Frank", "score": 88}
]


i = 0
while i < len(students):
  student = students[i]
  if student["score"] > 75:
    print(f"{student['name']} scored more than 75.")
```

OUTPUT:



EXPLANATION:

- **students = [...]**: This line creates a list named students. Each element in this list is a dictionary, where each dictionary represents a student and has two key-value pairs: "name" (the student's name) and "score" (the student's score).
- **i = 0**: This line initializes a variable i to 0. This variable will be used as an index to access elements in the students list.
- **while i < len(students):**: This is a while loop. The loop will continue to execute as long as the value of i is less than the total number of elements in the students list (obtained using len(students)). This ensures that the loop iterates through all the students in the list.
- **student = students[i]**: Inside the loop, this line accesses the dictionary at the current index i from the students list and assigns it to the variable student. So, in each iteration, student will hold the dictionary of the current student being processed.
- **if student["score"] > 75:**: This is an if statement that checks if the value associated with the key "score" in the current student dictionary is greater than 75.
- **print(f"{student['name']} scored more than 75.")**: If the if condition is true (the student's score is greater than 75), this line prints a formatted string indicating the name of the student (accessed using student['name']) and the message "scored more than 75.".
- **i += 1**: This line increments the value of i by 1 after each iteration. This is crucial for the while loop to eventually terminate and to move to the next student in the list.

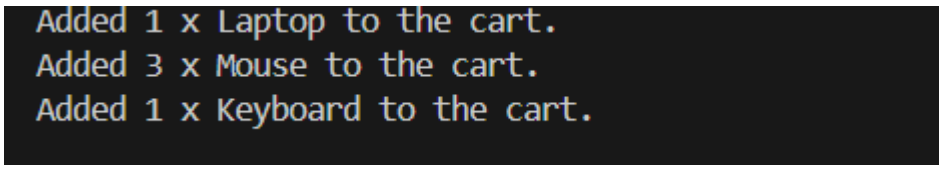
TASK 5:

```
class ShoppingCart:
    def __init__(self):
        self.items = []
```

```
def add_item(self, item_name, item_price, quantity=1):
    """Adds an item to the shopping cart."""
    self.items.append({"name": item_name, "price": item_price, "quantity": quantity})
    print(f"Added {quantity} x {item_name} to the cart.")

def remove_item(self, item_name):
    """Removes an item from the shopping cart."""
    initial_item_count = len(self.items)
    self.items = [item for item in self.items if item["name"] != item_name]
    if len(self.items) < initial_item_count:
        print(f"Removed {item_name} from the cart.")
    else:
        print(f"{item_name} not found in the cart.")

def calculate_total(self):
    """Calculates the total bill with conditional discounts."""
    total_bill = 0
    for item in self.items:
        item_total = item["price"] * item["quantity"]
        # Apply discount based on item name (example)
        if item["name"] == "Laptop":
            item_total *= 0.9  # 10% discount on Laptops
        elif item["name"] == "Mouse" and item["quantity"] >= 2:
            item_total *= 0.95 # 5% discount on Mouse if buying 2 or more
        total_bill += item_total

    # Apply overall discount based on total bill (example)
    if total_bill > 200:
```

```
        total_bill *= 0.85 # 15% discount if total bill is over $200

        # Example usage:
cart = ShoppingCart()

cart.add_item("Laptop", 1200, 1)

cart.add_item("Mouse", 25, 3)

cart.add_item("Keyboard", 75, 1)


print(f"Total bill: ${cart.calculate_total():.2f}")


cart.remove_item("Mouse")

print(f"Total bill after removing Mouse: ${cart.calculate_total():.2f}")
```

OUTPUT:



EXPLANATION:

1. **class ShoppingCart:**: This line defines the beginning of the ShoppingCart class.
2. **def __init__(self):**: This is the constructor of the class. When you create a new ShoppingCart object, this method is automatically called. It initializes an empty list called self.items. This list will store the items added to the cart. Each item in the list will be represented as a dictionary.
3. **def add_item(self, item_name, item_price, quantity=1):**: This method is used to add an item to the cart.
    - It takes the item_name, item_price, and an optional quantity (defaulting to 1) as input.
    - It creates a dictionary with these details and appends it to the self.items list.
    - It then prints a confirmation message.
4. **def remove_item(self, item_name):**: This method is used to remove an item from the cart based on its name.

- It first stores the initial number of items in initial_item_count.
- It then creates a new list self.items containing only the items whose names *do not* match the item_name you want to remove. This effectively removes all occurrences of that item.
- It checks if the number of items has decreased to confirm if the item was found and removed, and prints a corresponding message.

5. **def calculate_total(self):**: This is the core method for calculating the total bill with discounts.
    - It initializes total_bill to 0.
    - It then iterates through each item in the self.items list.
    - For each item, it calculates the item_total by multiplying the price and quantity.
    - **Conditional Discounts:** It applies discounts based on the item name:
        - If the item is a "Laptop", a 10% discount is applied (item_total *= 0.9).
        - If the item is a "Mouse" and the quantity is 2 or more, a 5% discount is applied (item_total *= 0.95).
    - The item_total (after any item-specific discounts) is added to the total_bill.
    - **Overall Discount:** After summing up all item totals, it checks if the total_bill is greater than $200. If it is, a 15% overall discount is applied (total_bill *= 0.85).
    - Finally, it returns the calculated total_bill