**ASSIGNMENT 6.4  
( 2403A52398 )**

**PROMPT01:**

Generate a Python class named student with attributes name, roll number, and marks, and then check if the marks are above the average marks

**CODE&OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**CODE EXPLANATION:**

1.class Student:: This line defines a new class named Student. Classes are blueprints for creating objects.

2.def \_\_init\_\_(self, name, roll\_number, marks):: This is the constructor of the Student class. It's a special method that gets called when you create a new Student object. It initializes the object's attributes:

* + self.name: Stores the student's name.
  + self.roll\_number: Stores the student's roll number.
  + self.marks: Stores the student's marks.

3.def is\_above\_average(self, average\_marks):: This is a method within the Student class. It takes the average\_marks as an argument and returns True if the student's marks (self.marks) are greater than the average marks, and False otherwise.

4.students = [...]: This creates a list of Student objects. Each element in the list is an instance of the Student class with a specific name, roll number, and marks.

5.total\_marks = sum(student.marks for student in students): This line calculates the sum of the marks of all students in the students list using a generator expression.

6.average\_marks = total\_marks / len(students): This line calculates the average marks by dividing the total\_marks by the number of students in the list (len(students)).

7. print(f"Average marks: {average\_marks:.2f}"): This line prints the calculated average marks, formatted to two decimal places.

8. for student in students:: This starts a loop that iterates through each student object in the students list.

9. if student.is\_above\_average(average\_marks):: Inside the loop, for each student, this line calls the is\_above\_average method we defined earlier, passing the calculated average\_marks.

10. print(f"{student.name}'s marks ({student.marks}) are above average."): If the is\_above\_average method returns True (meaning the student's marks are above average), this line prints a message indicating that.

11. else: print(f"{student.name}'s marks ({student.marks}) are not above average."): If the is\_above\_average method returns False, this line prints a message indicating that the student's marks are not above average.

**OUTPUT:A black background with white text

AI-generated content may be incorrect.**

**PROMPT 02:**

Write the first two lines of a for loop to iterate through a list of numbers. Calculate and print the square of even numbers only using Python.

**CODE&OUTPUT:A screenshot of a computer

AI-generated content may be incorrect.**

**CODE EXPLANATION:**

1. numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: This line creates a list named numbers containing integers from 1 to 10.
2. for number in numbers:: This is a for loop that iterates through each element in the numbers list. In each iteration, the current element is assigned to the variable number.
3. if number % 2 == 0:: This is an if statement that checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it means the number is even.
4. square = number \*\* 2: If the if condition is true (the number is even), this line calculates the square of the number using the exponentiation operator (\*\*) and stores the result in the variable square.
5. print(f"The square of {number} is {square}"): This line prints a formatted string. It includes the original even number and its calculated square.

**PROMPT 03:**

Generate a Python class called BankAccount with attributes account\_holder and balance. and complete methods for deposit(), withdraw(), and check for insufficient balance.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**CODE EXPLANATION:**

1. class BankAccount:: This line defines a new class named BankAccount.
2. def \_\_init\_\_(self, account\_holder, balance=0):: This is the constructor of the BankAccount class. It's called when you create a new BankAccount object.
   * self.account\_holder: Stores the name of the account holder.
   * self.balance: Stores the current balance of the account. It defaults to 0 if no initial balance is provided.
3. def deposit(self, amount):: This method handles depositing money into the account.
   * It checks if the amount is positive.
   * If it is, it adds the amount to the self-balance and prints a confirmation message with the new balance.
   * If the amount is not positive, it prints an error message.
4. def withdraw(self, amount):: This method handles withdrawing money from the account.
   * It checks if the amount is positive.
   * If it is, it then checks if the self-balance is greater than or equal to the amount (sufficient balance).
   * If there's sufficient balance, it subtracts the amount from the balance and prints a confirmation message with the new balance.
   * If there's insufficient balance, it prints an "Insufficient balance" message.
   * If the amount is not positive, it prints an error message.
5. def check\_balance(self):: This method simply prints the current account balance for the self.account\_holder.
6. # Example usage: This is a comment indicating the start of the example code that demonstrates how to use the BankAccount class.
7. my\_account = BankAccount("John Doe", 1000): This line creates a new BankAccount object named my\_account with the account holder "John Doe" and an initial balance of 1000.
8. my\_account.check\_balance(): This calls the check\_balance() method on the my\_account object, printing the initial balance.
9. my\_account.deposit(500): This calls the deposit() method to deposit 500 into the account.
10. my\_account.withdraw(200): This calls the withdraw() method to withdraw 200 from the account.
11. my\_account.withdraw(1500): This calls the withdraw() method to attempt to withdraw 1500, which will result in an "Insufficient balance" message since the balance is less than 1500 at this point.
12. my\_account.check\_balance(): This calls the check\_balance() method again to show the final balance after the transactions.

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**PROMPT 04:**

Using Python, define a list of student dictionaries with keys name and score, and write a while loop to print the names of students who scored more than 75

**CODE& OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**CODE EXPLANATION:**

1. students = [...]: This line creates a list named students. Each item in this list is a dictionary. Each dictionary has two key-value pairs:
   * "name": The student's name (a string).
   * "score": The student's score (an integer).
2. index = 0: This line initializes a variable index to 0. This variable will be used as a counter to keep track of our position in the students' list when using the while loop.
3. while index < len(students):: This is a while loop. The loop will continue to execute as long as the condition index < len(students) is true.
   * len(students) returns the number of items in the students list.
   * So, the loop will run as long as the index is less than the total number of students, ensuring we don't go out of bounds of the list.
4. student = students[index]: Inside the loop, this line accesses the element at the current index from the students list and assigns it to the variable student. Since the elements in the list are dictionaries, the student will be a dictionary in each iteration.
5. 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.
6. print(student["name"]): If the condition in the if statement is true (the student's score is greater than 75), this line prints the value associated with the key "name" from the current student dictionary, which is the student's name.
7. index += 1: This line increments the index by 1. This is crucial for the while loop to eventually terminate. It moves to the next student in the list for the next iteration.

**PROMPT 05:**

Generate a Python class ShoppingCart with an empty items list and add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**CODE EXPLANATION:**

1. class ShoppingCart:: This line defines the ShoppingCart class.
2. def \_\_init\_\_(self):: This is the constructor. It initializes an empty list called self.items when a new ShoppingCart object is created. This list will store the items in the cart.
3. def add\_item(self, item\_name, price, quantity):: This method adds an item to the shopping cart.
   * It creates a dictionary representing the item with its name, price, and quantity.
   * It then appends this item dictionary to the self. Item list.
4. def remove\_item(self, item\_name): This method removes an item from the shopping cart based on its name.
   * It iterates through the self. Items list.
   * If it finds an item whose name matches the item\_name provided, it removes that item from the list using self.items.remove(item).
   * The break statement is used to exit the loop after the first matching item is removed, so it only removes one instance of the item if there are duplicates.
5. def calculate\_total(self):: This method calculates the total cost of all items in the shopping cart, applying conditional discounts.
   * total\_bill = 0: Initializes a variable to keep track of the total bill.
   * for item in self.items: This loop iterates through each item dictionary in the self.item list.
   * item\_price = item["price"] \* item["quantity"]: Calculates the base price for the current item (price per item multiplied by the quantity).
   * if item["quantity"] >= 5:: Checks if the quantity of the current item is 5 or more. If true, a 10% discount is applied to the item\_price (item\_price \*= 0.9).
   * if item["name"] == "DiscountItem":: Checks if the name of the current item is "DiscountItem". If true, an additional 5% discount is applied to the item\_price (item\_price \*= 0.95). Note that if an item meets both conditions (quantity >= 5 AND name is "DiscountItem"), both discounts will be applied sequentially.
   * total\_bill += item\_price: Adds the (potentially discounted) item\_price to the total\_bill.
   * return total\_bill: After iterating through all items, the method returns the final total\_bill.
6. # Example usage: This section demonstrates how to use the ShoppingCart class.
   * cart = ShoppingCart(): Creates an instance of the ShoppingCart class.
   * cart.add\_item(...): Adds several items to the cart using the add\_item method.
   * print(f"Total bill: {cart.calculate\_total():.2f}"): Calls the calculate\_total method on the cart object and prints the returned total bill, formatted to two decimal places.

**OUTPUT:**

**A black and white screen with white text

AI-generated content may be incorrect.**