**ASSIGNMENT – 6.4**

**NAME : SAI CHARAN .P**

**ROLL:NO : 2403A52343**

**BATCH : AI 13**

**Prompt-1:** Start a Python class named Student with attributes name, roll\_number, and marks displaying details and checking if marks are above average.

**Program:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

**A close-up of a number

AI-generated content may be incorrect.**

**Explanation:**

This code defines a Python class called Student.

* class Student:: This line declares a new class named Student.
* \_\_init\_\_(self, name, roll\_number, marks): This is the constructor method. It's called when you create a new Student object. It initializes the attributes of the object: name, roll\_number, and marks. self refers to the instance of the class being created.
* self.name = name, self.roll\_number = roll\_number, self.marks = marks: These lines assign the values passed during the object creation to the object's attributes.
* display\_details(self): This method is defined to print the details of the student (name, roll number, and marks).
* is\_above\_average(self, average\_marks): This method takes an average\_marks value as input and compares it with the student's marks. It returns True if the student's marks are greater than the average, and False otherwise.
* student1 = Student("Alice", "101", 95): This line creates an instance (an object) of the Student class named student1, passing "Alice", "101", and 95 as the values for name, roll number, and marks respectively.
* student1.display\_details(): This line calls the display\_details method on the student1 object, which prints the student's information.
* average\_marks = 80: This line sets a variable average\_marks to 80.
* if student1.is\_above\_average(average\_marks):: This line calls the is\_above\_average method on student1 with the average\_marks value. The if statement checks if the method returns True.
* print(f"Marks are above average."): This line is executed if the student's marks are above average.
* else: print(f"Marks are not above average."): This line is executed if the student's marks are not above average.

In summary, the code defines a blueprint for a Student object, creates a specific student named Alice, displays her details, and then checks if her marks are above the average of 80, printing the corresponding message.

**Prompt-2:** Write the first two lines of a for loop to iterate through a list of numbers. Use a comment prompt how to calculate and print the square of even numbers only.

**Program:**

**A white paper with numbers and letters

AI-generated content may be incorrect.**

**Output:**

**A white background with black text

AI-generated content may be incorrect.**

**Explanation:**

This code iterates through a list of numbers and prints the square of only the even numbers.

* 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 that iterates through each element in the numbers list. In each iteration, the current element is assigned to the variable number.
* 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.
* print(number\*\*2): This line is executed only if the if condition is true (i.e., the number is even). It calculates the square of the number using the exponentiation operator (\*\* 2) and prints the result.

In summary, the code goes through each number in the list, checks if it's even, and if it is, it prints the square of that number.

**Prompt-3:** Create a class called BankAccount with attributes account\_holder and balance and complete methods for deposit(), withdraw(), and check for insufficient balance.

**Program:**

**A white background with text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

**A white background with black text

AI-generated content may be incorrect.**

**Explanation:**

This code defines a Python class called BankAccount and then demonstrates how to use it.

BankAccount Class Definition:

* class BankAccount:: This line declares a new class named BankAccount.
* \_\_init\_\_(self, account\_holder, balance=0): This is the constructor method. It's called when you create a new BankAccount object. It initializes the attributes of the object: account\_holder and balance. self refers to the instance of the class being created. The balance is set to a default value of 0 if not provided.
* self.account\_holder = account\_holder, self.balance = balance: These lines assign the values passed during the object creation to the object's attributes.
* deposit(self, amount): This method allows you to deposit money into the account. It checks if the amount is positive and, if so, adds it to the balance and prints a confirmation message.
* withdraw(self, amount): This method allows you to withdraw money from the account. It checks if the amount is positive and if there is enough balance. If both conditions are met, it subtracts the amount from the balance and prints a confirmation. Otherwise, it prints an error message.
* check\_insufficient\_balance(self, amount): This method checks if the current balance is less than the given amount. It returns True if the balance is insufficient for a withdrawal of that amount, and False otherwise.

Demonstration Code:

* account1 = BankAccount("Bob", 1000): This line creates an instance (an object) of the BankAccount class named account1, with "Bob" as the account holder and an initial balance of 1000.
* account1.deposit(500): This line calls the deposit method on account1 to deposit 500. You will see the updated balance printed.
* account1.withdraw(200): This line calls the withdraw method on account1 to withdraw 200. You will see the updated balance printed.
* account1.withdraw(1500): This line attempts to withdraw 1500, which is more than the current balance. The withdraw method will detect insufficient balance and print an error message.
* withdrawal\_amount = 1000 and withdrawal\_amount = 500: These lines set variables to represent potential withdrawal amounts.
* The if account1.check\_insufficient\_balance(withdrawal\_amount): ... else: ... blocks demonstrate how to use the check\_insufficient\_balance method to check if a withdrawal of a specific amount would be possible, and print a corresponding message.

In summary, the code defines a BankAccount object blueprint and then shows how to create an account, deposit and withdraw funds, and check for insufficient balance using the defined methods.

**Prompt-4:** 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.

**Program:**

**A computer screen shot of a code

AI-generated content may be incorrect.**

**Output:**

**A white background with black text

AI-generated content may be incorrect.**

**Explanation:**

This code defines a list of student records, where each record is a dictionary, and then uses a while loop to find and print the names of students who have a score greater than 75.

* students = [...]: This line creates a list named students. Each element in the list is a dictionary, with keys "name" and "score" representing a student's name and their 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 (which is determined by len(students)). This ensures that the loop iterates through all the elements in the list.
* student = students[i]: Inside the loop, this line accesses the element at the current index i from the students list and assigns it to the variable student. Since each element is a dictionary, student will be a dictionary in each iteration.
* 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(student["name"]): If the condition in the if statement is true (i.e., 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.
* i += 1: This line increments the value of i by 1 after each iteration. This is crucial for the while loop to eventually terminate, as it moves the index to the next element in the list.

In summary, the code goes through each student record in the list one by one. For each student, it checks their score, and if the score is above 75, it prints the student's name.

**Prompt-5:** Begin writing a class ShoppingCart with an empty items list and add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts.

**Program:**

**A white background with red green and blue text

AI-generated content may be incorrect.**

**A close-up of a computer screen

AI-generated content may be incorrect.**

**Output:**

**A close up of a number

AI-generated content may be incorrect.**

**Explanation:**

This code defines a Python class called ShoppingCart to manage items and calculate the total bill with discounts, and then demonstrates how to use it.

ShoppingCart Class Definition:

* class ShoppingCart:: This line declares a new class named ShoppingCart.
* \_\_init\_\_(self): This is the constructor method. It's called when you create a new ShoppingCart object. It initializes an empty list called self.items which will store the items added to the cart.
* add\_item(self, item): This method takes an item (presumably a dictionary with details like name, price, and quantity) and appends it to the self.items list.
* remove\_item(self, item\_name): This method attempts to remove an item from the cart based on its item\_name. It iterates through the self.items list, and if it finds an item with a matching name, it removes the first occurrence of that item and prints a confirmation message. If the item is not found after checking all items, it prints a "not found" message.
* calculate\_total(self): This method calculates the total bill. It initializes total\_bill to 0 and then iterates through each item in the self.items list. For each item, it retrieves the price and quantity (defaulting to 0 and 1 if not found) and adds the product of price and quantity to the total\_bill. After calculating the initial subtotal, it applies conditional discounts: a 10% discount if the total\_bill is greater than 100, and a 5% discount if the total\_bill is greater than 50 (but not greater than 100, due to the elif). Finally, it returns the calculated total\_bill.

Demonstration Code:

* cart = ShoppingCart(): This line creates an instance (an object) of the ShoppingCart class named cart.
* cart.add\_item(...): These lines add several items (Laptop, Mouse, Keyboard, Monitor) as dictionaries to the cart using the add\_item method.
* cart.remove\_item("Mouse"): This line calls the remove\_item method to remove the item with the name "Mouse" from the cart.
* total\_bill = cart.calculate\_total(): This line calls the calculate\_total method on the cart object and stores the returned total bill in the total\_bill variable.
* print(f"Total Bill: ${total\_bill:.2f}"): This line prints the final calculated total\_bill, formatted to two decimal places.

In summary, the code defines a ShoppingCart object that can hold items, add and remove them, and calculate a discounted total based on the subtotal. The demonstration part shows how to create a cart, add and remove items, and then get the final bill.