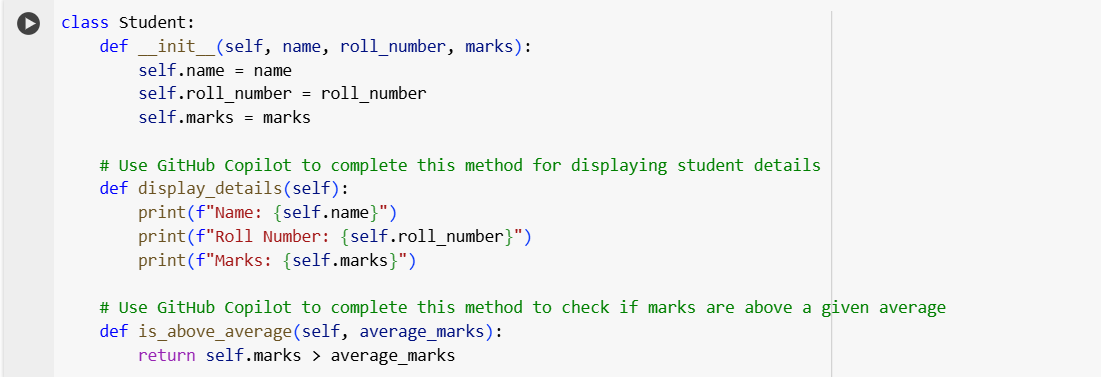
ASSIGNMENT-6.4

TASK-1

CODE:



A screen shot of a computer

AI-generated content may be incorrect.

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

EXPLANATION:

This cell defines a Python class named Student.

* **class Student:**: This line declares the start of a new class named Student.
* **\_\_init\_\_(self, name, roll\_number, marks):**: This is the constructor method of the class. It's called when you create a new Student object. It takes self (which refers to the instance of the class being created) and three other arguments: name, roll\_number, and marks. Inside the constructor, these values are assigned to the corresponding attributes of the Student object (self.name, self.roll\_number, and self.marks).
* **display\_details(self):**: This method is used to print the details of a student object to the console. It accesses the name, roll\_number, and marks attributes of the object using self and prints them in a formatted string.
* **is\_above\_average(self, average\_marks):**: This method checks if a student's marks are above a given average. It takes self and average\_marks as arguments. It compares the student's self.marks with the average\_marks and returns True if the student's marks are greater than the average, and False otherwise.

**Cell 78dff90f:**

This cell demonstrates how to use the Student class.

* **student1 = Student("Alice", "A001", 85)**: This line creates a new instance (object) of the Student class named student1. It passes the values "Alice", "A001", and 85 to the \_\_init\_\_ method.
* **student2 = Student("Bob", "B002", 70)**: This line creates another instance of the Student class named student2 with the provided details.
* **print("Details for Student 1:")**: This line prints a header before displaying student1's details.
* **student1.display\_details()**: This line calls the display\_details method on the student1 object, which prints its name, roll number, and marks.
* **print("-" \* 20)**: This line prints a separator line for better readability.
* **average = 75**: This line sets a variable average to 75, which will be used as the average marks for comparison.
* **print(f"Is Student 2 above the average of {average}? {student2.is\_above\_average(average)}")**: This line calls the is\_above\_average method on the student2 object, passing the average variable (75) as an argument. The result (True or False) is then printed along with a descriptive message.

In summary, these cells define a basic Student class and show how to create student objects, display their details, and perform a simple comparison of their marks against an average.

TASK-2:

CODE:

A white background with green text

AI-generated content may be incorrect.

OUTPUT:

A white background with black numbers

AI-generated content may be incorrect.

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 number % 2 is equal to 0, it means the number is divisible by 2 and is therefore even.
4. **print(number \*\* 2)**: If the if condition is true (i.e., the number is even), this line calculates the square of the number using the exponentiation operator (\*\* 2) and then prints the result to the console.

In summary, this code iterates through a list of numbers and prints the square of only the even numbers in the list.

TASK-3

CODE:

A screenshot of a computer program

AI-generated content may be incorrect.

Output:

A white background with black text

AI-generated content may be incorrect.

Explanation:

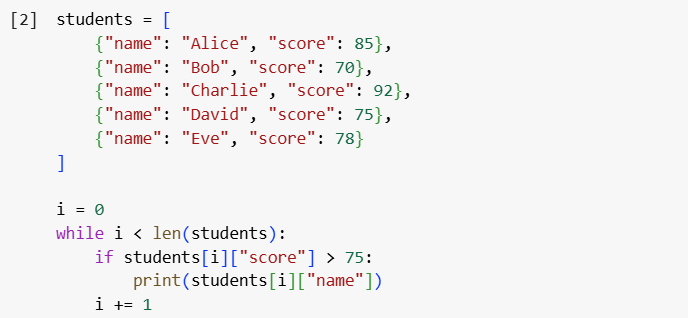
This code defines a Python class called BankAccount which simulates a simple bank account.

* \_\_init\_\_(self, account\_holder, balance=0): This is the constructor of the class. It's called when you create a new BankAccount object. It takes the account\_holder's name as a required argument and an optional balance (defaulting to 0). It initializes the account\_holder and balance attributes of the object.
* 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. Otherwise, it prints an error 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 sufficient balance. If both conditions are met, it subtracts the amount from the balance and prints a confirmation. Otherwise, it prints an appropriate error message (either for a non-positive amount or insufficient balance).
* check\_balance(self): This method simply prints the current account\_holder and their balance.

The code then demonstrates how to use the BankAccount class by creating an instance named account1, depositing 100, withdrawing 50, attempting to withdraw 200 (which should fail due to insufficient balance), and finally checking the balance.

TASK-4

Code:



Output:

A close up of a name

AI-generated content may be incorrect.

Explanation:

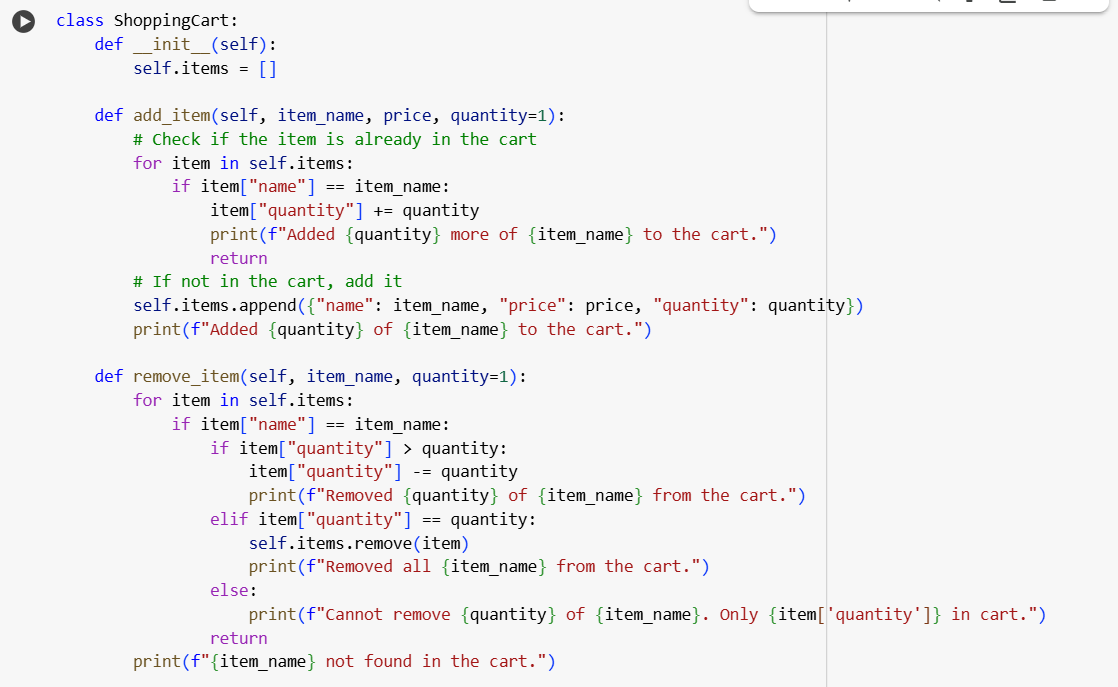
This code snippet works with a list of dictionaries, where each dictionary represents a student and contains their 'name' and 'score'.

1. **students = [...]**: This line initializes a list named students. Each element in the list is a dictionary.
2. **i = 0**: This line initializes a variable i to 0. This variable will be used as an index to iterate through the students list.
3. **while i < len(students):**: This is a while loop that continues as long as the value of i is less than the total number of elements in the students list (len(students)). This ensures that the loop iterates through each student in the list.
4. **if students[i]["score"] > 75:**: Inside the loop, this line accesses the dictionary at the current index i in the students list (students[i]). It then accesses the value associated with the key "score" within that dictionary (students[i]["score"]). It checks if this score is greater than 75.
5. **print(students[i]["name"])**: If the condition in the if statement is true (the student's score is greater than 75), this line accesses the value associated with the key "name" in the current student's dictionary (students[i]["name"]) and prints it to the console.
6. **i += 1**: This line increments the value of i by 1 after each iteration. This is crucial for moving to the next student in the list and eventually terminating the while loop when all students have been checked.

In summary, the code iterates through the list of students and prints the name of any student whose score is above 75.

TASK-5

Code:



A screen shot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Output:

A screenshot of a computer program

AI-generated content may be incorrect.

Explanation:

This code defines a Python class called ShoppingCart to simulate a simple shopping cart system.

* \_\_init\_\_(self): This is the constructor. It initializes an empty list called self.items which will store the items in the cart. Each item in the list will be a dictionary.
* add\_item(self, item\_name, price, quantity=1): This method adds an item to the cart. It first checks if the item is already in the self.items list. If it is, it increments the quantity of the existing item. If not, it appends a new dictionary representing the item (with its name, price, and quantity) to the self.items list.
* remove\_item(self, item\_name, quantity=1): This method removes a specified quantity of an item from the cart. It iterates through the self.items list to find the item. If found, it checks if the quantity to remove is less than or equal to the quantity in the cart. It updates the quantity or removes the item entirely if the quantities match.
* calculate\_total(self): This method calculates the total bill for the items in the cart. It iterates through the self.items list, calculates the total price for each item (price \* quantity), and applies a conditional discount (10% off if the item's total is over $100). It then adds the item's (potentially discounted) total to the total\_bill.
* display\_cart(self): This method prints the current contents of the shopping cart, showing the name, price, and quantity of each item.

The code then shows an example of how to use the ShoppingCart class by creating an instance, adding and removing items, displaying the cart contents, and calculating the final total bill.

\*\*\*\*\*