**LAB ASSIGNMENT – 6.4**

**NAME : Bhavaneesh.A**

**HALL.NO :**  **2403A52350**

**BATCH.NO : AI 13**

**ASSIGNMENT : 6.4**

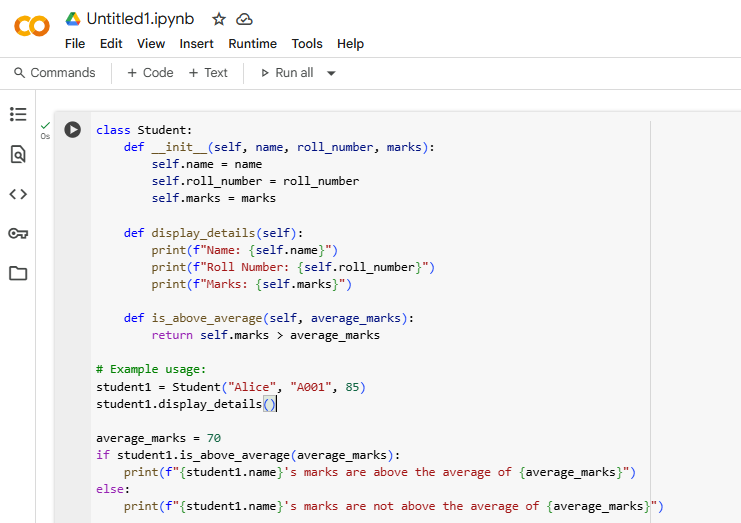
**Prompt – 1:**

Start a Python class named Student with attributes name, roll\_number, and marks.

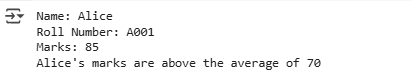
Add two methods:

1. display\_details() → prints the student's name, roll number, and marks
2. is\_above\_average(average=50) → returns True if marks are above average, otherwise False.

**Code:**



**Output:**

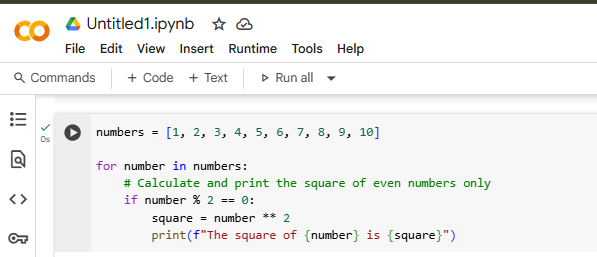


**Explanation:**

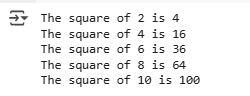
* **class Student:**: This line defines a new class named Student. Classes are blueprints for creating objects (instances).
* **\_\_init\_\_(self, name, roll\_number, marks)**: This is the constructor method. It's called when you create a new Student object.
  + self refers to the instance of the class being created.
  + name, roll\_number, and marks are the attributes (data) that each Student object will have.
  + Inside the constructor, self.name = name, self.roll\_number = roll\_number, and self.marks = marks assign the values passed when creating the object to the object's attributes.
* **display\_details(self)**: This method is defined within the Student class. It allows you to print the details (name, roll number, and marks) of a specific Student object.
  + self is used to access the attributes of the object calling the method.
* **is\_above\_average(self, average\_marks)**: This method checks if a student's marks are above a given average.
  + It takes self (the student object) and average\_marks as input.
  + It returns True if self.marks is greater than average\_marks, and False otherwise.
* **student1 = Student("Alice", "A001", 85)**: This line creates an instance (object) of the Student class named student1. It passes "Alice", "A001", and 85 as the values for name, roll\_number, and marks respectively.
* **student1.display\_details()**: This calls the display\_details method on the student1 object, which prints its details.
* **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, passing average\_marks (70) as the argument. The if statement checks if the method returns True.
* **print(...)**: These lines print messages based on whether student1's marks are above average.

**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.

**Code:**



**Output:**



**Explanation:**

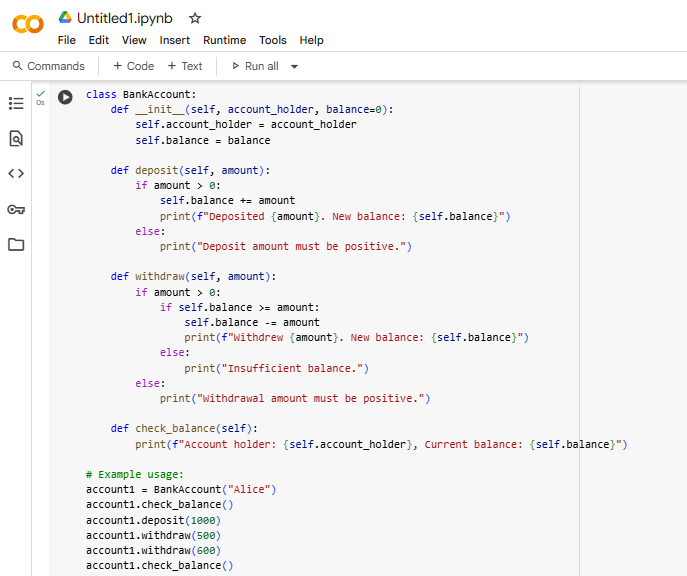
Certainly! The code in the selected cell iterates through a list of numbers and prints the square of only the even numbers. Here's a breakdown:

* 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 number % 2 is equal to 0, it means the number is perfectly divisible by 2, and thus, 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 using the exponentiation operator (\*\*) and stores it in the variable square.
* print(f"The square of {number} is {square}"): This line prints the result using an f-string. It displays the original even number and its calculated square.

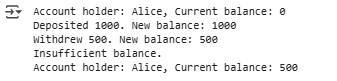
**Prompt – 3:**

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

**Code:**



**Output:**



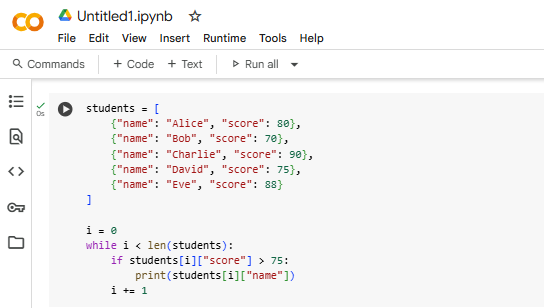
**Explanation:**

Certainly! The code in the selected cell defines a Python class called BankAccount, which simulates a basic bank account. Here's a breakdown:

* **class BankAccount:**: This line defines 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.
  + self refers to the instance of the class being created.
  + account\_holder is an attribute to store the name of the account holder.
  + balance is an attribute to store the account balance, with a default value of 0 if not specified.
  + Inside the constructor, self.account\_holder = account\_holder and self.balance = balance assign the provided values to the object's attributes.
* **deposit(self, amount)**: This method allows you to deposit money into the account.
  + It checks if the amount is greater than 0.
  + If it is, the amount is added to the self.balance, and a confirmation message is printed.
  + If the amount is not positive, an error message is printed.
* **withdraw(self, amount)**: This method allows you to withdraw money from the account.
  + It checks if the amount is greater than 0.
  + If it is, it then checks if the self.balance is greater than or equal to the amount (to prevent overdrafts).
  + If there is sufficient balance, the amount is subtracted from self.balance, and a confirmation message is printed.
  + If there is insufficient balance, an "Insufficient balance" message is printed.
  + If the amount is not positive, an error message is printed.
* **check\_balance(self)**: This method simply prints the current account holder and their balance.
* **account1 = BankAccount("Alice")**: This line creates an instance (object) of the BankAccount class named account1 with the account holder "Alice". The balance is initialized to 0 by default.
* **account1.check\_balance()**: Calls the check\_balance method to print the initial balance.
* **account1.deposit(1000)**: Calls the deposit method to add 1000 to the balance.
* **account1.withdraw(500)**: Calls the withdraw method to subtract 500 from the balance.
* **account1.withdraw(600)**: Calls the withdraw method to attempt to subtract 600 from the balance. Since the balance is only 500, the "Insufficient balance" message is printed.
* **account1.check\_balance()**: Calls the check\_balance method again to print the final balance.

**Prompt – 4 :** Define a list of student dictionaries with keys name and score. to write a while loop to print the names of students who scored more than 75.

**Code:**



**Output:**



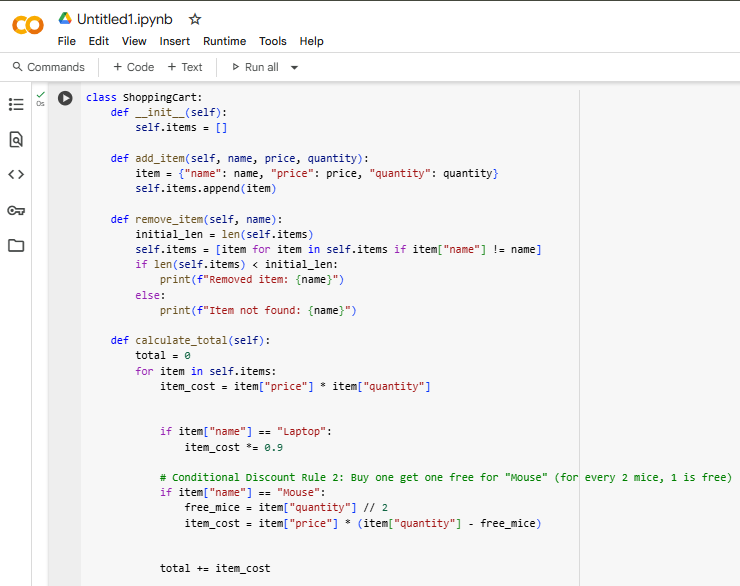
**Explanation:**

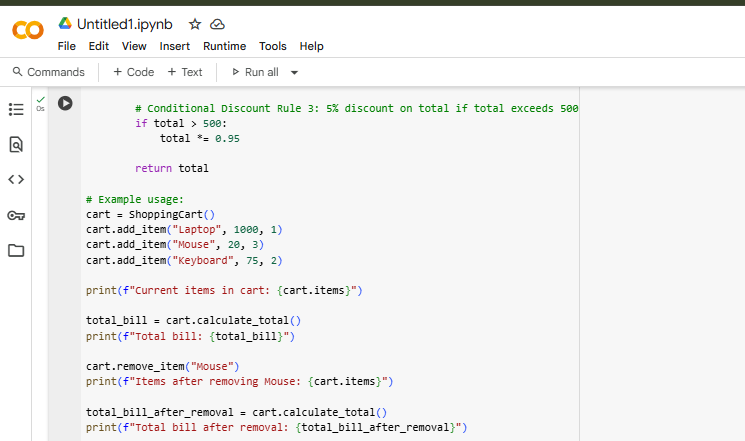
Certainly! The code in the selected cell iterates through a list of student dictionaries and prints the names of students whose score is greater than 75. Here's a breakdown:

* **students = [...]**: This line defines 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 number of elements in the students list (which is obtained using len(students)). This ensures that the loop iterates through all the elements in the list.
* **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" in that dictionary (students[i]["score"]). The if statement checks if this score is greater than 75.
* **print(students[i]["name"])**: If the if condition 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.
* **i += 1**: This line increments the value of i by 1. This is crucial for the while loop to eventually terminate. It moves the index to the next element in the list for the next iteration.

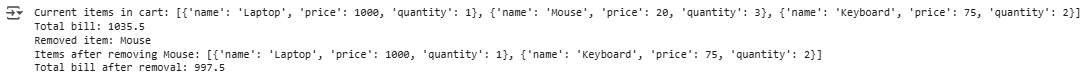
**Prompt – 5:** Begin writing a class ShoppingCart with an empty items list. generate methods to add\_item, remove\_item, and use a loop to calculate the total bill using conditional discounts. all in one

**Code:**





**Output:**



**Explanation:**

Certainly! The code you've provided defines a ShoppingCart class with methods to manage items and calculate the total with specific discount rules. Here's a breakdown:

* **class ShoppingCart:**: This defines the ShoppingCart class.
* **\_\_init\_\_(self):**: The constructor initializes an empty list called self.items to store the items in the cart.
* **add\_item(self, name, price, quantity):**: This method adds an item to the cart.
  + It creates a dictionary for the item with 'name', 'price', and 'quantity'.
  + It appends this item dictionary to the self.items list.
* **remove\_item(self, name):**: This method removes an item from the cart by its name.
  + It creates a new list self.items that includes all items except the one with the matching name.
  + It prints a message indicating whether the item was removed or not found.
* **calculate\_total(self):**: This is the core method for calculating the total bill with discounts.
  + It initializes total to 0.
  + It iterates through each item in the self.items list.
  + item\_cost = item["price"] \* item["quantity"] calculates the cost for the current item before discounts.
  + **Conditional Discount Rule 1:** if item["name"] == "Laptop": item\_cost \*= 0.9 applies a 10% discount if the item is a "Laptop".
  + **Conditional Discount Rule 2:** if item["name"] == "Mouse": free\_mice = item["quantity"] // 2; item\_cost = item["price"] \* (item["quantity"] - free\_mice) implements a "buy one get one free" logic for "Mouse" items. For every two mice, one is free, so the cost is calculated based on the quantity minus the free mice.
  + total += item\_cost adds the calculated cost of the current item to the total.
  + **Conditional Discount Rule 3:** if total > 500: total \*= 0.95 applies a 5% discount to the *entire* total if the total exceeds 500 after individual item discounts.
  + return total returns the final calculated total bill.
* **Example usage:** The code then demonstrates how to create a ShoppingCart object, add items, print the initial items and total, remove an item, and then print the items and total after removal.