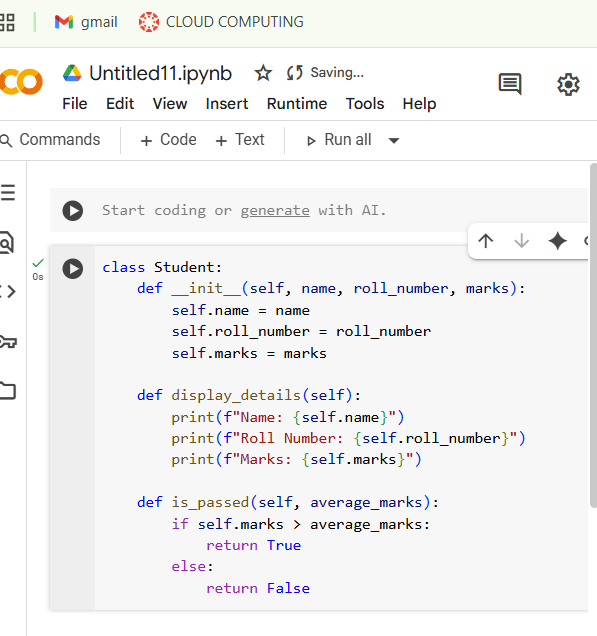
Assingment-6

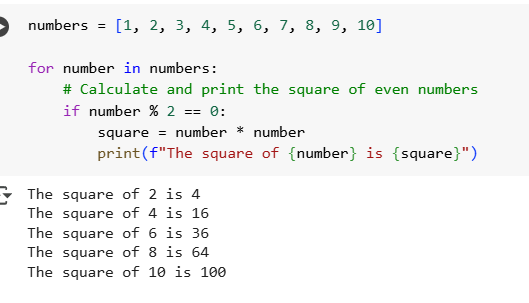
Task-1: Start a Python class named Student with attributes name, roll\_number, and marks. Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average.

Explaintion:

Python class called Student.

* The  in it method is the constructor, which is used to create new Student objects. It takes the student's name, roll\_number, and marks as input and stores them as attributes of the object.
* The display\_details method simply prints out the student's name, roll number, and marks.
* The is\_passed method takes an average\_marks value as input and checks if the student's marks are greater than the average. It returns True if they are, and False otherwise.

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



Explaination:

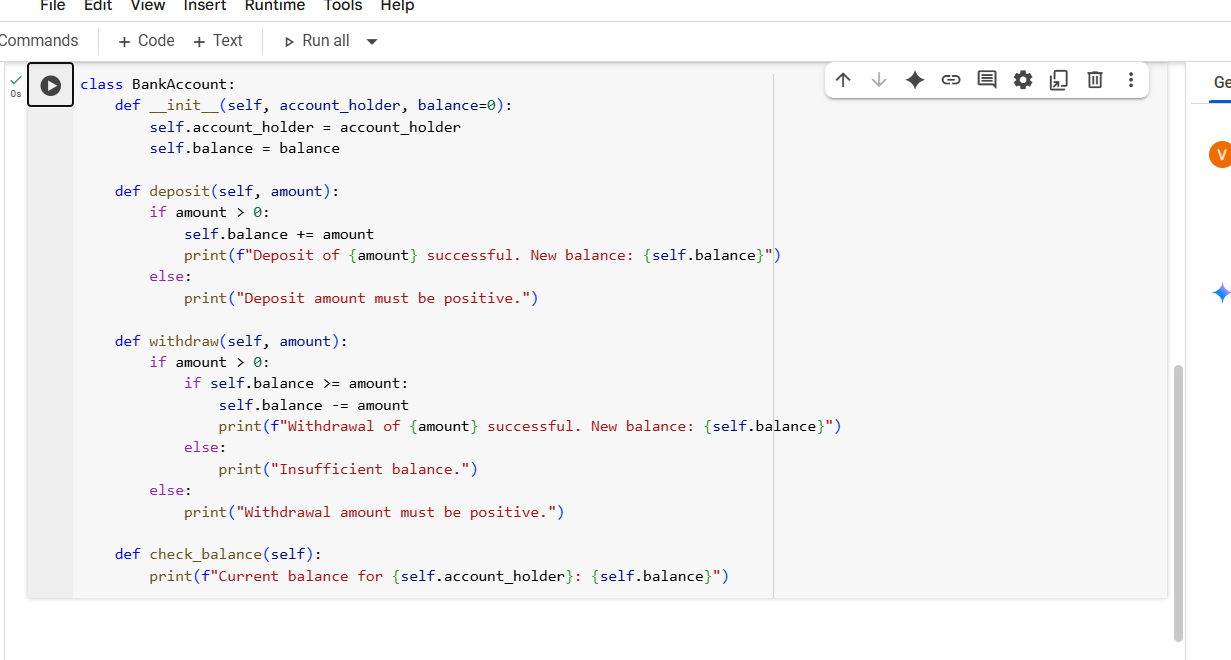
Certainly! This code iterates through a list of numbers and prints the square of the even numbers.

* numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] creates a list named numbers containing integers from 1 to 10.
* for number in numbers: starts a for loop that will go through each element in the numbers list, assigning the current element to the variable number in each iteration.
* if number % 2 == 0: checks if the current number is even using the modulo operator (%). If the remainder when divided by 2 is 0, the number is even.
* square = number \* number calculates the square of the even number and stores it in the square variable.
* print(f"The square of {number} is {square}") prints a formatted string showing the original even number and its calculated square.

So, for each even number in the list, the code calculates its square and prints the result.

Task-3:

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



Explaination:

Absolutely! This code defines a Python class called BankAccount to simulate a simple bank account.

* The \_\_init\_\_ method is the constructor. It's called when you create a new BankAccount object. It takes the account\_holder's name and an optional initial balance (defaulting to 0) and stores them as attributes of the object.
* The deposit method takes an amount as input. If the amount is positive, it adds it to the balance and prints a success message. Otherwise, it prints an error message.
* The withdraw method takes an amount as input. If the amount is positive, it then checks if the current balance is greater than or equal to the amount. If it is, the amount is subtracted from the balance, and a success message is printed. If there's insufficient balance, it prints an "Insufficient balance" message. If the withdrawal amount is not positive, it prints an error.
* The check\_balance method simply prints the current balance for the account\_holder.

In essence, this class provides basic functionality for depositing, withdrawing, and checking the balance of a bank account, with safeguards against depositing or withdrawing non-positive amounts and overdrawing

Task-4:

Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.



Explaination:

Certainly! This code iterates through a list of student records and prints the names of students whose scores are above 75.

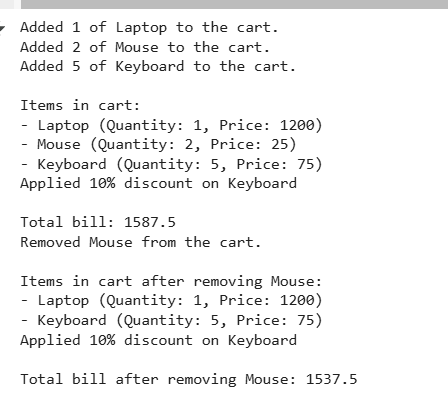
* students = [...] defines a list named students. Each element in the list is a dictionary representing a student, with keys for name and score.
* i = 0 initializes a counter variable i to 0. This variable will be used to access elements in the students list by their index.
* while i < len(students): starts a while loop. The loop will continue 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 all the student records.
* if students[i]["score"] > 75: inside the loop, this line accesses the dictionary at the current index i in the students list (students[i]) and then accesses the value associated with the key "score" (students[i]["score"]). It then checks if this score is greater than 75.
* print(students[i]["name"]) if the condition in the if statement is true (the score is greater than 75), this line accesses the name of the student at the current index i (students[i]["name"]) and prints it to the console.
* i += 1 this line increments the counter i by 1 after each iteration of the loop. This is crucial for the loop to eventually terminate and to move to the next student in the list.

In summary, the code goes through each student in the list one by one. For each student, it checks their score. If the score is higher than 75, it prints the student's name.

Task-5:

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





Explaination:

Certainly! This code demonstrates how to use the ShoppingCart class we defined earlier by creating an instance of the class and calling its methods.

Here's a breakdown:

1. **my\_cart = ShoppingCart()**: This line creates a new object (an instance) of the ShoppingCart class and assigns it to the variable my\_cart. This initializes an empty list called items within this my\_cart object.
2. **my\_cart.add\_item("Laptop", 1200, 1)**, **my\_cart.add\_item("Mouse", 25, 2)**, **my\_cart.add\_item("Keyboard", 75, 5)**: These lines call the add\_item method of the my\_cart object to add different items to the shopping cart. Each call specifies the item name, price, and quantity. Notice that the "Keyboard" is added with a quantity of 5, which will trigger the discount later.
3. **print("\nItems in cart:")** and the following for loop: This section is for demonstration. It prints a heading and then iterates through the items list within the my\_cart object, printing the details of each item currently in the cart.
4. **total = my\_cart.calculate\_total()**: This line calls the calculate\_total method of the my\_cart object. This method iterates through the items, calculates the cost for each (applying a 10% discount if the quantity is 5 or more), sums up the costs, and returns the final total. The returned total is stored in the total variable.
5. **print(f"\nTotal bill: {total}")**: This line prints the calculated total bill.
6. **my\_cart.remove\_item("Mouse")**: This line calls the remove\_item method to remove the item named "Mouse" from the my\_cart.
7. **print("\nItems in cart after removing Mouse:")** and the following for loop: Similar to the earlier display, this section shows the items remaining in the cart after "Mouse" has been removed.
8. **total\_after\_removal = my\_cart.calculate\_total()**: This line calls the calculate\_total method again to calculate the new total bill after "Mouse" has been removed.
9. **print(f"\nTotal bill after removing Mouse: {total\_after\_removal}")**: This line prints the new total bill after the removal.

In summary, the code creates a shopping cart, adds items to it, calculates and prints the initial total, removes an item, and then calculates and prints the total again to show the effect of the removal. It demonstrates the basic usage of the ShoppingCart class methods.