ASSIGNMENT 6

*Name -Sriram Sushanth*

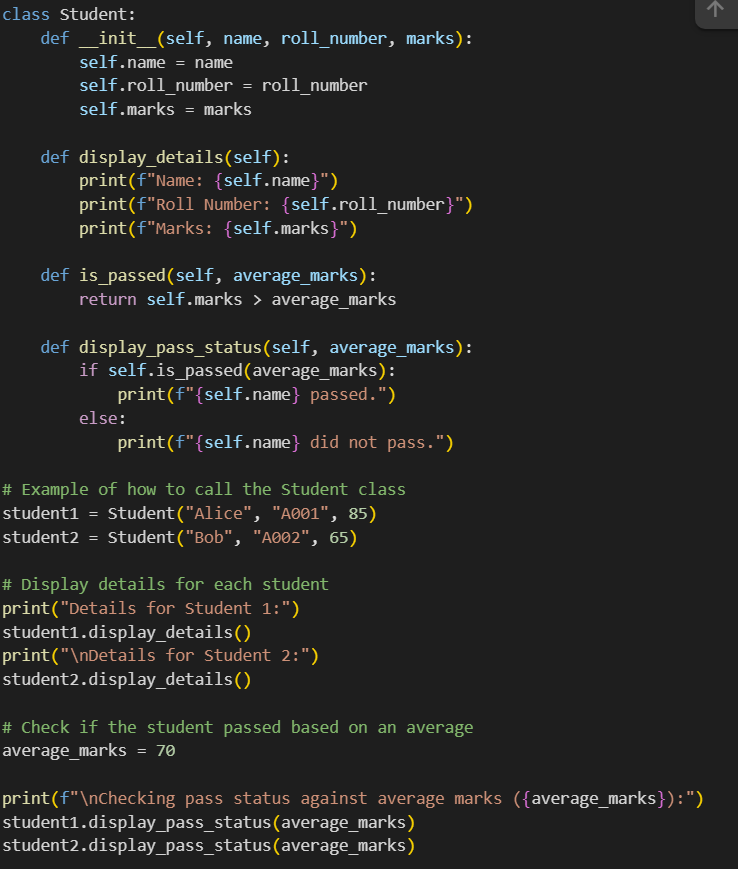
*Roll – 2403a54079*

*Batch – 03 (DS)*

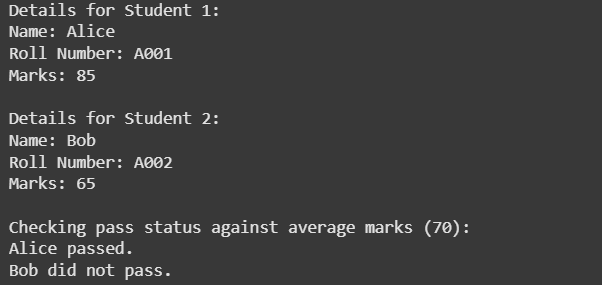
Task Description #1:  
• Start a Python class named Student with attributes name, roll\_number, and marks. Prompt  
GitHub Copilot to complete methods for displaying details and checking if marks are above  
average

PROMPT: Create a Student class having attributes name, rollnumber, marks and display the results using a function named display\_details(), and create another function named is\_passed() to check whether the student is passed using their average and call the class

CODE:



OUTPUT:



EXPLANATION:

This code defines a Python class called Student.

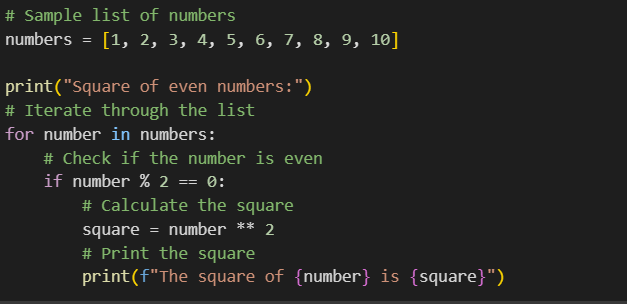
Here's a breakdown of the code:

* **class Student:**: This line starts the definition of the Student class.
* **\_\_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, and name, roll\_number, and marks are the attributes you provide when creating a student. Inside the constructor, these attributes are assigned to the object using self.attribute\_name.
* **display\_details(self):**: This method prints the name, roll\_number, and marks of the student object.
* **is\_passed(self, average\_marks):**: This method checks if the student's marks are greater than the average\_marks provided as an argument. It returns True if the student's marks are higher and False otherwise.
* **display\_pass\_status(self, average\_marks):**: This method uses the is\_passed() method to determine if the student passed and then prints a message indicating whether the student passed or did not pass.
* **student1 = Student("Alice", "A001", 85) and student2 = Student("Bob", "A002", 65)**: These lines create two instances (objects) of the Student class, named student1 and student2, with different details.
* **The print statements and method calls**: These lines demonstrate how to use the methods of the Student objects to display their details and check their pass status based on an average\_marks of 70.

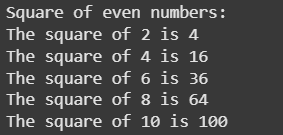
Task Description #2:  
• Write the first two lines of a for loop to iterate through a list of numbers. Use a comment  
prompt to let Copilot suggest how to calculate and print the square of even numbers only.

PROMPT: write a python program to calculate and print the square of even numbers from a list of numbers, write program using for loop to iterate

CODE:



OUTPUT:



EXPLANATION:

This code snippet calculates and prints the square of even numbers from a given list.

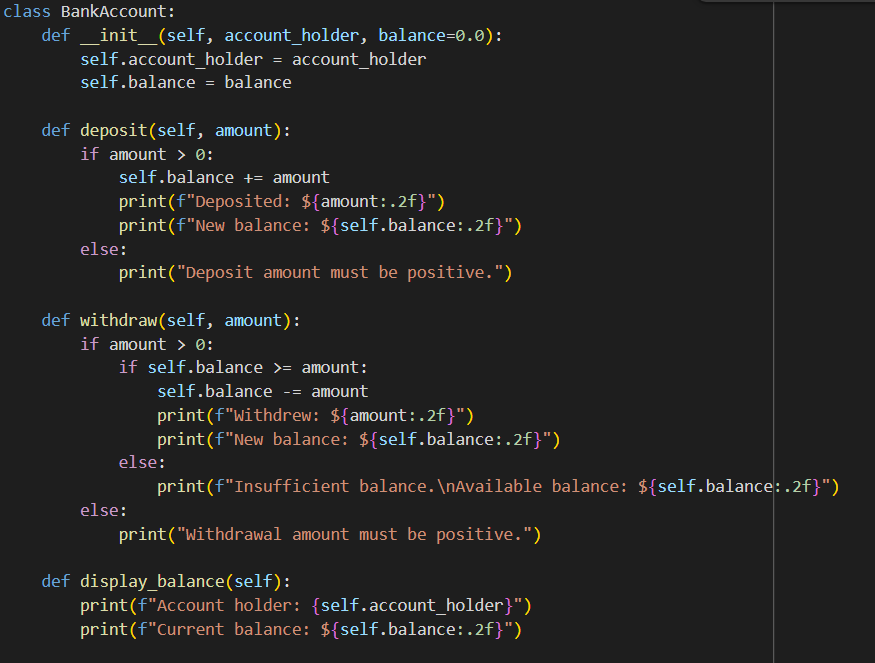
Here's a breakdown of the code:

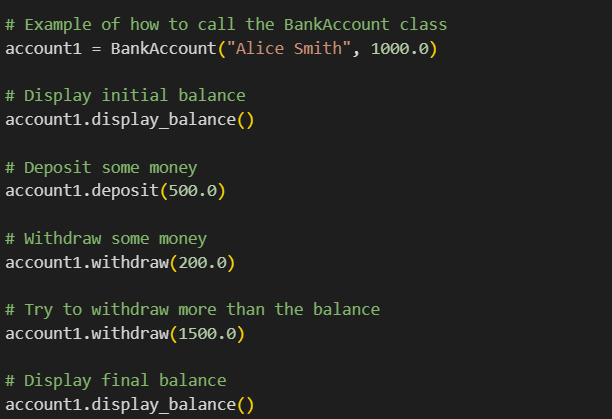
* **numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**: This line creates a list named numbers containing integers from 1 to 10.
* **print("Square of even numbers:")**: This line simply prints a descriptive header to the console.
* **for number in numbers:**: This is a for loop that iterates through each element 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's 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 the result in the square variable.
* **print(f"The square of {number} is {square}")**: This line prints the original even number and its calculated square in a formatted string. This line only executes for even numbers.

Task Description #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.

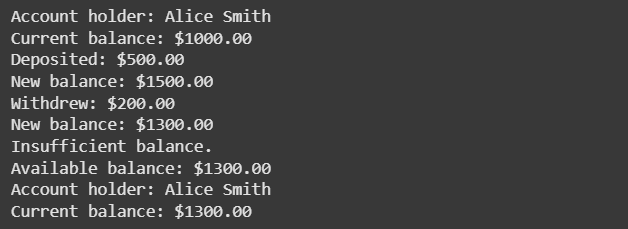
PROMPT: Create a class named BanckAccount with attributes account\_holder and balance and self attributes. Compete methods for deposit() using if conditions , withdraw(), and check for insufficient balance. And,call the class

CODE:





OUTPUT:



EXPLANATION:

This code defines a Python class called BankAccount and provides an example of how to use it.

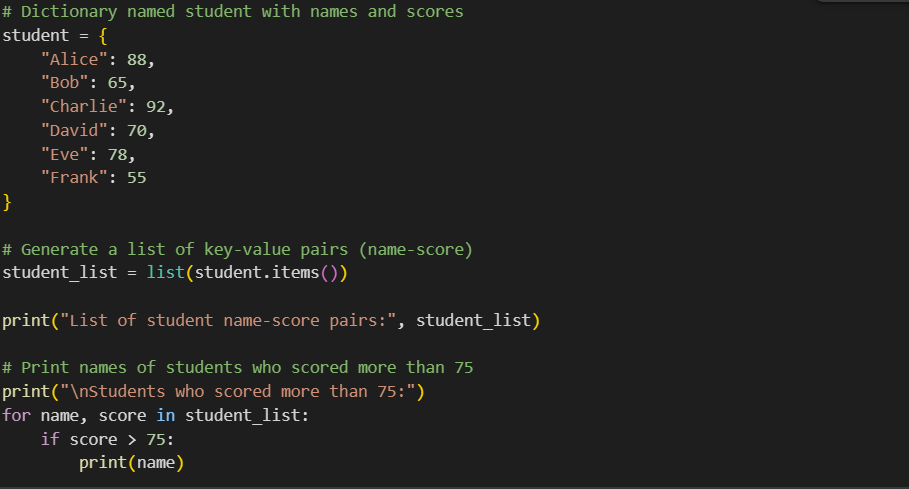
Here's a breakdown:

* **class BankAccount:**: This line starts the definition of the BankAccount class.
* **\_\_init\_\_(self, account\_holder, balance=0.0):**: This is the constructor. It's called when you create a new BankAccount object. self refers to the object itself. account\_holder is the name of the account holder, and balance is the initial balance (defaulting to 0.0 if not provided). These values are stored as attributes of the object.
* **deposit(self, amount):**: This method handles deposits. It checks if the amount is positive, adds it to the balance, and prints a confirmation message with the new balance.
* **withdraw(self, amount):**: This method handles withdrawals. 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 for insufficient balance or a non-positive withdrawal amount.
* **display\_balance(self):**: This method simply prints the account\_holder and the current balance of the account.
* **The code after the class definition**: This part demonstrates how to create an instance of the BankAccount class (account1), display its initial balance, perform a deposit, perform a withdrawal, attempt an invalid withdrawal (more than the balance), and finally display the balance after these operations.

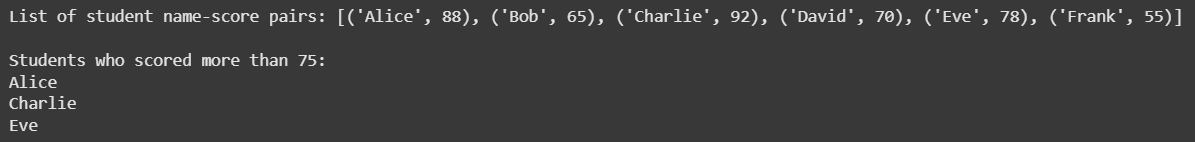
Task Description #4:  
• Define a list of student dictionaries with keys name and score. Ask Copilot to write a while  
loop to print the names of students who scored more than 75

PROMPT: write a python program which has dictionary named student and generate a list of key(name)-value(score) pairs and should print the names of student who scores more than75

CODE:



OUTPUT:



EXPLANATION:

This code snippet works with a dictionary of student names and their scores to identify and print the names of students who scored more than 75.

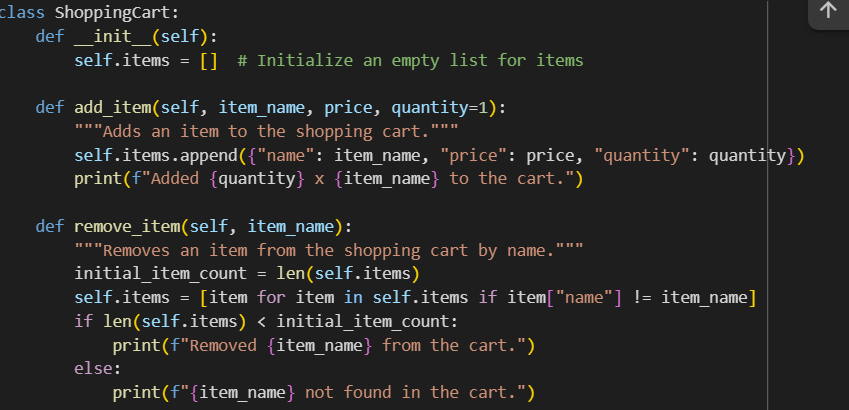
Here's a breakdown:

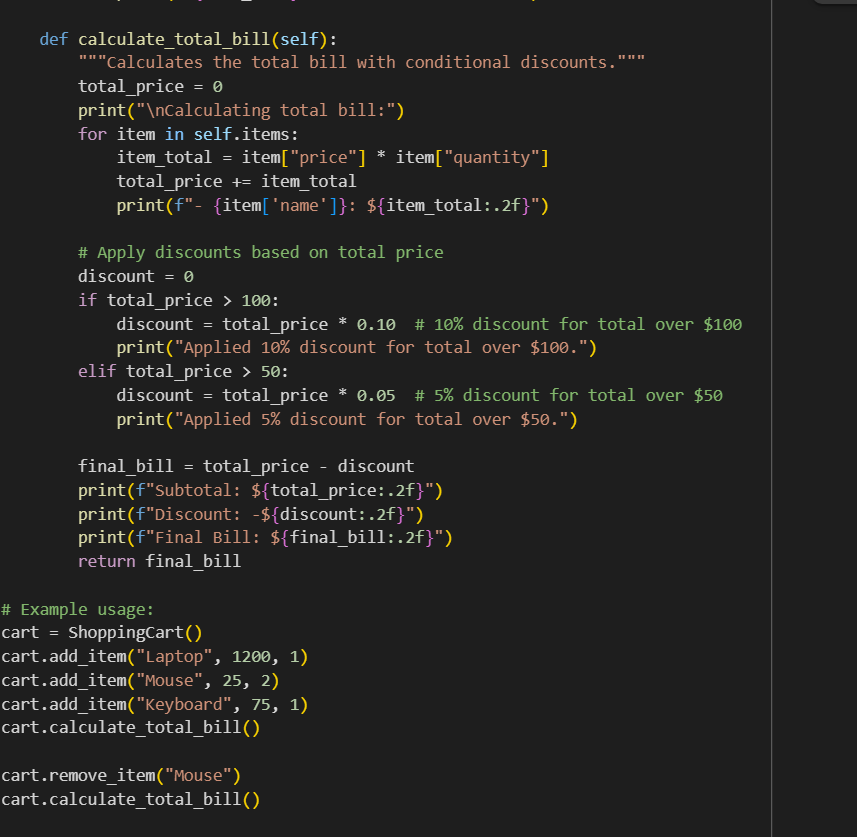
* **student = { ... }**: This line creates a dictionary named student. The keys of the dictionary are student names (strings), and the values are their corresponding scores (integers).
* **student\_list = list(student.items())**: The .items() method of a dictionary returns a view object that displays a list of a dictionary's key-value tuple pairs. This line converts that view object into a list of tuples, where each tuple contains a student's name and their score.
* **print("List of student name-score pairs:", student\_list)**: This line prints the generated list of name-score pairs.
* **print("\nStudents who scored more than 75:")**: This line prints a header before listing the names of students who meet the criteria.
* **for name, score in student\_list:**: This is a for loop that iterates through each tuple (name, score) in the student\_list. In each iteration, the first element of the tuple is assigned to the variable name, and the second element is assigned to the variable score.
* **if score > 75:**: This is an if statement that checks if the current score is greater than 75.
* **print(name)**: If the if condition is true (the score is greater than 75), this line prints the name of the student. This line only executes for students who scored more than 75.

Task Description #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

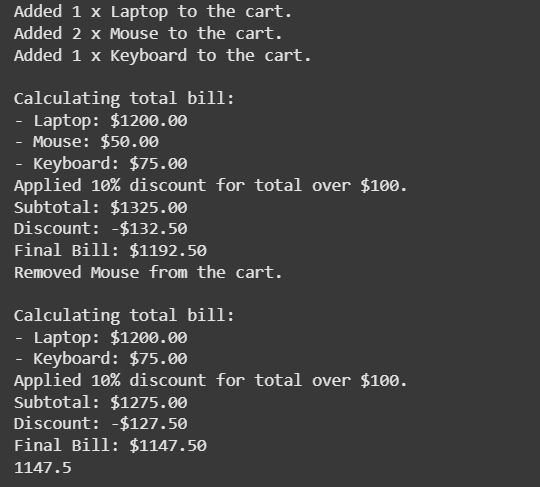
PROMPT: write a program in python. Create a class named ShopingCart having empty list items. Add methods like add\_item, remove\_item and uses a loop to calculate the total bill using conditional(if-else) discounts. And call the class.

CODE:





OUTPUT:



EXPLANATION:

This code defines a Python class named ShoppingCart to simulate a basic shopping cart functionality and provides an example of how to use it.

Here's a breakdown:

* **class ShoppingCart:**: This line begins the definition of the ShoppingCart class.
* **\_\_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\_name, price, quantity=1):**: This method adds an item to the items list. It takes the item\_name, price, and optional quantity (defaulting to 1) as arguments. Each item is stored as a dictionary within the items list.
* **remove\_item(self, item\_name):**: This method removes an item from the cart based on its item\_name. It uses a list comprehension to create a new list containing only the items whose names do not match the item\_name to be removed. It also provides feedback on whether the item was found and removed.
* **calculate\_total\_bill(self):**: This method calculates the total cost of the items in the cart and applies conditional discounts.
  + It initializes total\_price to 0.
  + It iterates through each item in the self.items list.
  + For each item, it calculates the item\_total (price multiplied by quantity) and adds it to the total\_price.
  + It then checks the total\_price to apply discounts: 10% for totals over $100, and 5% for totals over $$100, and 5% for totals over $50.
  + Finally, it calculates the final\_bill by subtracting the discount from the total\_price and prints a summary of the calculation, including the subtotal, discount, and final bill.
* **The code after the class definition**: This section demonstrates how to create a ShoppingCart object, add several items, calculate the initial bill, remove an item, and then calculate the bill again to show the effect of the removal and potential change in discount.