

AI ASSISTED CODING LAB-6.4

TASK 01

```
class Student:

    def __init__(self, name, roll_number, marks):

        self.name = name

        self.roll_number = roll_number

        self.marks = marks


# Prompt GitHub Copilot to complete the following methods:

# Method to display student details

def display_details(self):

    print(f"Name: {self.name}")

    print(f"Roll Number: {self.roll_number}")

    print(f"Marks: {self.marks}")


# Method to check if marks are above average (you can define the average)

def is_passed(self, average_marks):

    return self.marks >= average_marks


# Create a Student object and demonstrate the methods

student1 = Student("Alice", "A101", 85)

student1.display_details()
```

```

average = 75

if student1.is_passed(average):

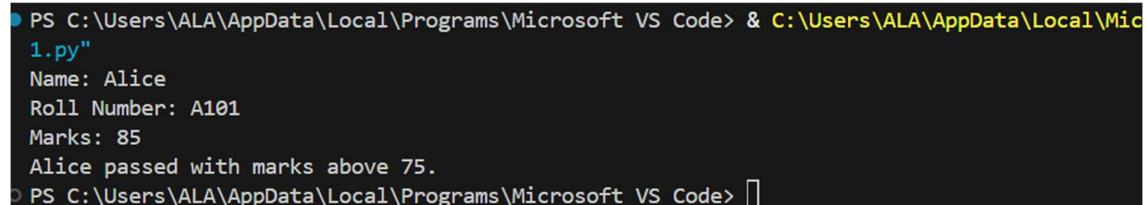
    print(f"{student1.name} passed with marks above {average}.")

else:

    print(f"{student1.name} did not pass with marks above {average}.")

```

OUTPUT :



```

PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ALA\AppData\Local\Mic
1.py"
Name: Alice
Roll Number: A101
Marks: 85
Alice passed with marks above 75.
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> 

```

EXPLANATION :

This code defines a Python class called Student and demonstrates its usage.

- **class Student::** This line declares a new class named Student. Classes are blueprints for creating objects.
- **__init__(self, name, roll_number, marks)::** This is the constructor method. It's called when you create a new Student object. It initializes the object's attributes: name, roll_number, and marks.
- **self.name = name, self.roll_number = roll_number, self.marks = marks:** These lines assign the values passed to the constructor to the object's corresponding attributes. self refers to the instance of the class being created.
- **display_details(self)::** This method prints the details of the student (name, roll number, and marks) in a formatted way.
- **is_passed(self, average_marks)::** This method checks if the student's marks are greater than or equal to a given average_marks. It returns True if the student passed and False otherwise.
- **student1 = Student("Alice", "A101", 85):** This line creates a new Student object named student1 with the name "Alice", roll number "A101", and marks 85.
- **student1.display_details():** This line calls the display_details method on the student1 object, printing its details.
- **average = 75:** This line sets a variable average to 75.

- **if student1.is_passed(average)::** This line calls the `is_passed` method on `student1` with the `average` value and checks if it returns `True`.
- **print(f'{student1.name} passed with marks above {average}.')** This line is executed if the student's marks are above or equal to the average, printing a success message.
- **else: print(f'{student1.name} did not pass with marks above {average}.')** This block is executed if the student's marks are below the average, printing a failure message.

0 / 2000

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

TASK 2

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

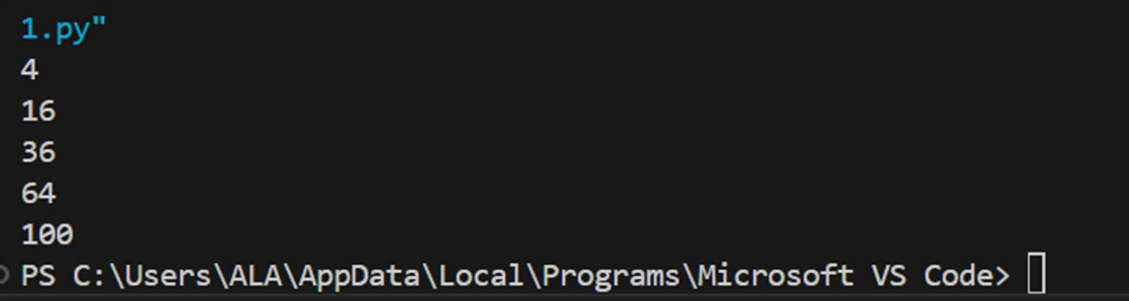
```
for number in numbers:
```

```
    # Calculate the square of the number, if it is even, and print the result.
```

```
    if number % 2 == 0:
```

```
        square = number ** 2
```

```
        print(square)
```



```
1.py
4
16
36
64
100
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code>
```

EXPLANATION :

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

- **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 starts a for loop that will go 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 a conditional statement that checks if the current number is even. The modulo operator (%) returns the remainder of a division. If number % 2 is 0, it means the number is perfectly divisible by 2 and therefore even.
- **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 (** 2) and assigns the result to the variable square.
- **print(square):** This line prints the value stored in the square variable to the console.

In summary, the code loops through the list, checks if each number is even, and if it is, it calculates and prints its square.

0 / 2000

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

TASK 3

```
class BankAccount:
```

```
    def __init__(self, account_holder, balance=0):
```

```
        self.account_holder = account_holder
```

```
        self.balance = balance
```

```
    def deposit(self, amount):
```

```
        # Add the amount to the balance
```

```
        self.balance += amount
```

```
        print(f"Deposited {amount}. New balance is {self.balance}")
```

```

def withdraw(self, amount):
    # Check for insufficient balance
    if self.balance - amount < 0:
        print("Insufficient balance.")
        return False
    else:
        # Deduct the amount from the balance
        self.balance -= amount
        print(f"Withdrew {amount}. New balance is {self.balance}")
        return True

def check_balance(self):
    # Print the current balance
    print(f"Current balance: {self.balance}")
    # Create a new bank account
my_account = BankAccount("Alice", 1000)

# Check the initial balance
my_account.check_balance()

# Make a deposit
my_account.deposit(500)

# Make a withdrawal
my_account.withdraw(200)

# Attempt to withdraw more than the balance
my_account.withdraw(2000)

```

Check the final balance

my_account.check_balance()

```
1.py"
Current balance: 1000
Deposited 500. New balance is 1500
Withdrew 200. New balance is 1300
Insufficient balance.
Current balance: 1300
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> |
```

EXPLANATION :

Cell 1 (fdd26afc):

This cell iterates through a list of numbers from 1 to 10. For each number, it checks if the number is even. If it is, it calculates the square of that number and prints the result.

Cell 2 (1ffe4020):

This cell defines a Python class called **BankAccount**. This class represents a bank account and has the following methods:

- **__init__(self, account_holder, balance=0):** This is the constructor. It initializes a new **BankAccount** object with an **account_holder** name and an optional starting balance (defaulting to 0).
- **deposit(self, amount):** This method allows you to deposit an amount into the account, increasing the balance.
- **withdraw(self, amount):** This method allows you to withdraw an amount from the account. It first checks if there is enough balance. If not, it prints an "Insufficient balance" message.
- **check_balance(self):** This method prints the current balance of the account.

Cell 3 (5db74382):

This cell demonstrates how to use the **BankAccount** class:

- It creates a new **BankAccount** instance named **my_account** with the account holder "Alice" and an initial balance of 1000.
- It then calls the **check_balance()** method to print the initial balance.
- It calls the **deposit()** method to deposit 500.

- It calls the `withdraw()` method to withdraw 200.
- It attempts to withdraw 2000, which will fail due to insufficient balance.
- Finally, it calls `check_balance()` again to show the final balance after the operations.

TASK 4

```
students = [  
    {"name": "Bob", "score": 90},  
    {"name": "Charlie", "score": 70},  
    {"name": "David", "score": 80},  
    {"name": "Eve", "score": 65},  
    {"name": "Alice", "score": 85}  
]
```

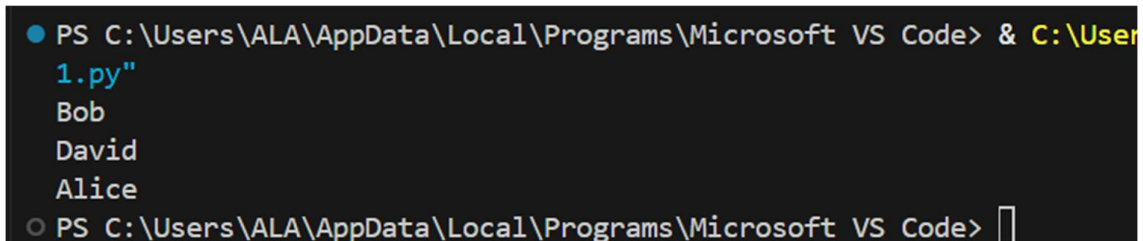
```
i = 0
```

```
while i < len(students):
```

```
    if students[i]["score"] > 75:
```

```
        print(students[i]["name"])
```

```
    i += 1
```



```
● PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\User  
1.py  
Bob  
David  
Alice  
○ PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> █
```

EXPLANATION :

This code snippet works with a list of student records to identify high-scoring students:

- `students = [...]`: This line creates a list named `students`. Each item in the list is a dictionary representing a student with keys for "name" and "score".
- `i = 0`: This initializes a counter variable `i` to 0. This variable will be used to iterate through the `students` list.
- `while i < len(students)::` This starts a while loop that will continue as long as the value of `i` is less than the number of elements in the `students` list.
- `if students[i]["score"] > 75::` Inside the loop, this line checks if the "score" of the current student (accessed using `students[i]`) 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 prints the "name" of the current student.
- `i += 1`: This line increments the counter `i` by 1 after each iteration. This moves the loop to the next student in the list.

In essence, the code iterates through the list of students and prints the names of those who have a score greater than 75.

0 / 2000

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

TASK 5

```
class ShoppingCart:
```

```
    def __init__(self):
```

```
        self.items = []
```

```
    def add_item(self, item):
```

```
        self.items.append(item)
```

```
    def remove_item(self, item):
```

```
        try:
```



```

        self.items.remove(item)

    except ValueError:

        print(f"{item} not found in the shopping cart.")


def calculate_total(self):

    total_bill = 0

    for item in self.items:

        total_bill += item.get('price', 0) # Using .get() for safer access


    # Apply conditional discount

    if total_bill > 100:

        total_bill *= 0.90 # 10% discount


    return total_bill


# 1. Create an instance of the ShoppingCart class.
cart = ShoppingCart()


# 2. Define a list of items.
items_list = [

    {"name": "Laptop", "price": 1200},

    {"name": "Mouse", "price": 25},

    {"name": "Keyboard", "price": 75},

    {"name": "Monitor", "price": 300},

    {"name": "Webcam", "price": 50}

]


# 3. Add several items from the list to the shopping cart.
cart.add_item(items_list[0]) # Laptop

```

```
cart.add_item(items_list[1]) # Mouse
cart.add_item(items_list[2]) # Keyboard
cart.add_item(items_list[3]) # Monitor
cart.add_item(items_list[4]) # Webcam
cart.add_item(items_list[1]) # Another Mouse
```

4. Remove one or more items from the shopping cart.

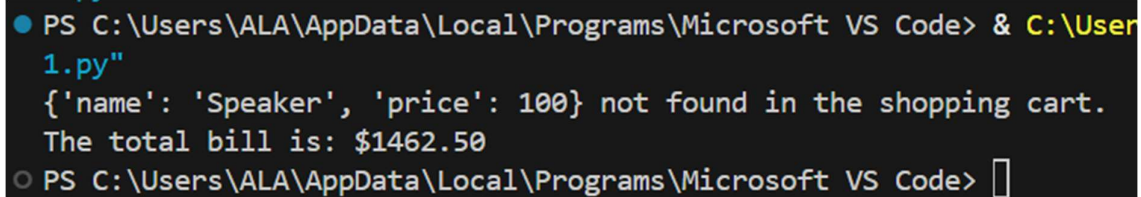
```
cart.remove_item(items_list[4]) # Remove Webcam
cart.remove_item({"name": "Speaker", "price": 100}) # Try removing an item not in the
cart
```

5. Call the calculate_total method to get the final bill amount.

```
total_bill = cart.calculate_total()
```

6. Print the total bill amount to the console.

```
print(f"The total bill is: ${total_bill:.2f}")
```



```
● PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\User
1.py"
{'name': 'Speaker', 'price': 100} not found in the shopping cart.
The total bill is: $1462.50
○ PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> █
```

EXPLANATION :

Data Analysis Key Findings

- The ShoppingCart class was successfully created with methods to add and remove items and calculate the total bill.
- The add_item method correctly appends items to the cart's item list.
- The remove_item method removes a specified item and includes error handling for items not found.
- The calculate_total method sums item prices and applies a 10% discount if the total exceeds \$100.

- The demonstration successfully showed adding multiple items, removing an existing item, attempting to remove a non-existent item (triggering the error message), and calculating the final discounted total bill.

Insights or Next Steps

- The current implementation assumes items are dictionaries with a 'price' key. Future development could involve creating an Item class to encapsulate item properties and potentially add quantity tracking.
- The discount logic is hardcoded. A next step could be to make discounts more flexible, perhaps based on item categories, quantities, or promotional codes.