

Assignment-6.4

2403a52132

Task 1:

>Give me a python code for displaying details and check marks are above average ,Students with attributes name, roll-number, marks, pass or fail.

>User should enter the details.

Code and Output:

The screenshot displays a Jupyter Notebook interface with a code editor on the left and a Gemini chat window on the right.

Code Editor:

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

def display_student_status(student, average_marks):
    student.display_details()
    if student.marks > average_marks:
        print("Status: Pass")
    else:
        print("Status: Fail")

# Example usage:
# Get student details from user input
name = input("Enter student name: ")
roll_number = input("Enter roll number: ")
marks = int(input("Enter marks: ")) # Convert marks to integer

# Get average marks from user input
average = int(input("Enter average marks: ")) # Convert average to integer

# Create a student object
student1 = Student(name, roll_number, marks)

# Display student status using the user-defined function
display_student_status(student1, average)
```

Gemini Chat:

give me a python code for displaying details and check marks are above average ,Students with attributes name, roll_number, marks, pass or fail by a user defined function

Okay, I understand. Here is the Python code that includes the `Student` class and a user-defined function to display the details and check if the marks are above average, indicating pass or fail.

The code for the `Student` class and the `display_student_status` function has been provided. Would you like to create a student object and test the function with different average marks?

user should enter the details

Okay, I will modify the code so that the user can enter the student's name, roll number, and marks, as well as the average marks to check against.

The code now allows you to input student details and the average marks. Would you like to try running the code with different inputs?

What can I help you build?

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

Terminal Output:

```
Enter student name: Raj Kumar
Enter roll number: 22
Enter marks: 98
Enter average marks: 40
Name: Raj Kumar
Roll Number: 22
Marks: 98
Status: Pass
```

Explanation:

Gemini X

K explanation

Certainly, I can explain the content of the cells you have visible.

Cell 1 (id: d38aac2b): This is a Markdown cell that provides a summary of the "ShoppingCart" task, key findings, and insights or next steps.

- It summarizes that the ShoppingCart class was successfully created with methods for adding, removing, and calculating the total with discounts.
- It lists key findings, such as the successful implementation of each method and the details of the conditional discounts applied (10% for quantity > 5 and 5% for price > \$50).
- It suggests insights or next steps, including adding validation to the add_item method and considering more complex discount rules.

Cell 2 (id: dbe3c467): This cell defines a Python class named Student.

- The __init__ method is the constructor of the class. It initializes a student object with their name, roll_number, and marks. It also takes average_marks as input during initialization and calculates whether the student passed by calling the is_above_average method. The result of this check is stored in the self.passed attribute.
- The display_details method prints the student's name, roll number, marks, and whether they passed (Yes or No) based on the value of the self.passed attribute.
- The is_above_average method takes average_marks as input and returns True if the student's marks are greater than average_marks, and False otherwise.
- The commented-out lines at the end of the cell show example usage of how to create a Student object and call the display_details method.

👍 🗨

◆ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

Task 2:

>Write a python program to creating the list of two numbers.

>Write the first two lines of a for loop to iterate through the list.

- >Calculates and print the square of even numbers only

Code and Output:

The screenshot shows the Google Colab environment. The top bar includes the Colab logo, the file name 'Al(6).ipynb', and various menu options like File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with icons for Commands, Code, Text, and Run all. The main workspace is divided into two panes. The left pane shows a Python script with two code cells. The first cell creates a list 'my_list' with two numbers and prints it. The second cell uses a for loop to iterate through the list, checking if each number is even, calculating its square, and printing the square. The right pane shows the Gemini chat interface with two messages. The first message asks for a Python program to create a list of two numbers. The second message asks for the first two lines of a for loop to iterate through the list. The Gemini response explains that a 'SyntaxError' was encountered because the 'for' loop was incomplete and provides a corrected version of the code.

Explanation:

Gemini

✕

📌

explanation

+

Certainly, I can explain the code in the cells you have visible.

Cell 1 (id: 102e44c1):

This cell defines a `Student` class and a function `display_student_status`.

- The `Student` class has an `__init__` method to initialize a student with a name, roll number, and marks.
- It also has a `display_details` method to print the student's name, roll number, and marks.
- The `display_student_status` function takes a `Student` object and an `average_marks` value as input. It first displays the student's details using the `display_details` method and then prints whether the student "Pass" or "Fail" based on whether their marks are greater than the provided `average_marks`.
- The rest of the code in this cell is an example of how to use the class and function, including getting input from the user for student details and average marks, creating a `Student` object, and then calling `display_student_status`.

Cell 2 (id: f2de40b8):

This cell creates a Python list named `my_list` with four integer values: 10, 22, 3, and 20. Then, it prints the entire list to the console.

Cell 3 (id: 160679fa):

This cell contains a `for` loop that iterates through each `number` in the `my_list`.

- Inside the loop, it checks if the current `number` is even using the modulo operator (`%`). If `number % 2 == 0` is true, it means the number is even.
- If the number is even, it calculates the square of the number (`number ** 2`) and stores it in the `square` variable.
- Finally, it prints the calculated `square` value to the console. This means only the squares of the even numbers from `my_list` will be printed.

👍

👎

🌟

What can I help you build?

🗨

➡

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

👍

👎

🔒

3:39 PM

Python 3

Task 3:

>Write the skeleton with attributes to generate methods `def deposit(self.amount):`
`def withdraw(self.amount)` and a method to check the balance.

>Test the class by creating objects.

Code and Output:

The screenshot displays a Jupyter Notebook environment with a code editor on the left and a Gemini chat interface on the right. The code in the notebook defines a `Classname` class with attributes `attribute1`, `attribute2`, and `initial_balance`. It includes methods for `deposit`, `withdraw`, and `check_balance`. The `deposit` method checks if the amount is positive and updates the balance. The `withdraw` method checks if the balance is sufficient and updates it. The `check_balance` method prints the current balance. The notebook also includes a test section where an object of the class is created and the methods are tested with various inputs.

```
[21] class Classname:
    def __init__(self, attribute1, attribute2, initial_balance=0):
        self.attribute1 = attribute1
        self.attribute2 = attribute2
        self.balance = initial_balance # Added balance attribute

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f'Deposit of {amount} successful. New balance: {self.balance}')
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f'Withdrawal of {amount} successful. New balance: {self.balance}')
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        print(f'Current balance: {self.balance}')

    # Add other methods here
```

Variables: `my_object`

```
[22] # Create an object of the Classname
# Replace 'value1' and 'value2' with actual values for attribute1 and attribute2
my_object = Classname('value1', 'value2', initial_balance=100)

# Test the check_balance method
my_object.check_balance()

# Test the deposit method
my_object.deposit(50)

# Test the withdraw method
my_object.withdraw(30)

# Test withdrawal with insufficient funds
my_object.withdraw(200)

# Check the balance again
my_object.check_balance()
```

Output:

```
Current balance: 100
Deposit of 50 successful. New balance: 150
Withdrawal of 30 successful. New balance: 120
Insufficient funds.
Current balance: 120
```

Gemini chat history:

- write the skeleton with attributes
 - Okay, here is a skeleton of a Python class with an `__init__` method to define and initialize attributes. You can replace `Classname` and the attribute names with your desired names. I have provided a skeleton for a Python class with attributes. Would you like to add more methods to this class or create an object from this class?
- to generate methods `def deposit(self.amount):` `def withdraw(self.amount)` and a method to check the balance
 - Okay, I will add the `deposit`, `withdraw`, and `check_balance` methods to the class skeleton. I'll also include a `balance` attribute to keep track of the account balance. I have added the `deposit`, `withdraw`, and `check_balance` methods to the class. Would you like to create an instance of this class and test these methods?
- test the class by creating objects
 - Great! Let's test the `Classname` by creating an object and using the methods. I have created an object of the `Classname` and tested the `deposit`, `withdraw`, and

What can I help you build?

Explanation:

Gemini X

K explanation

◆ Certainly, I can explain the code in the cells you have visible.

Cell 1 (id: 160679fa): This cell contains a `for` loop that iterates through each `number` in the list `my_list` (which was defined in a previous cell).

- Inside the loop, it checks if the current `number` is even using the modulo operator (`%`). If `number % 2 == 0` is true, it means the number is even.
- If the number is even, it calculates the square of the number (`number ** 2`) and stores it in the `square` variable.
- Finally, it prints the calculated `square` value to the console. This means only the squares of the even numbers from `my_list` will be printed.

Cell 2 (id: ecf83fa1): This cell defines a Python class named `ClassName`.

- The `__init__` method is the constructor of the class. It initializes the object with three attributes: `attribute1`, `attribute2`, and `balance` (which has a default value of 0 if not provided).
- The `deposit` method takes an `amount` as input and adds it to the `balance` if the amount is positive. It then prints a message indicating the successful deposit and the new balance.
- The `withdraw` method takes an `amount` as input. If the amount is positive and there are sufficient funds (`self.balance >= amount`), it subtracts the amount from the `balance` and prints a success message with the new balance. Otherwise, it prints an "Insufficient funds" message.
- The `check_balance` method simply prints the current value of the `balance` attribute.

Cell 3 (id: 6ea779a8): This cell demonstrates how to use the `ClassName` defined in the previous cell.

- It creates an object named `my_object` from the `ClassName`, providing values for `attribute1`, `attribute2`, and an `initial_balance` of 100.
- It then calls the `check_balance()` method to display the initial balance.
- It calls the `deposit(50)` method to add 50 to the balance and shows the updated balance.
- It calls the `withdraw(30)` method to subtract 30 from the balance and shows the updated balance.
- It attempts to call `withdraw(200)` to demonstrate the "Insufficient funds" message because the current balance is less than 200.
- Finally, it calls `check_balance()` again to show the final balance after the operations.

◆ What can I help you build?

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

Task 4:

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

Code and Output:

The screenshot displays a Jupyter Notebook environment with a code editor on the left and a Gemini chat interface on the right.

Code Editor:

```
# Define a list of student dictionaries
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 70},
    {"name": "Charlie", "score": 92},
    {"name": "David", "score": 65},
    {"name": "Eve", "score": 78},
]

# Initialize an index for the while loop
i = 0

# Use a while loop to iterate through the list
while i < len(students):
    student = students[i]
    if student["score"] > 75:
        print(student["name"])
    i += 1
```

Output:

```
Alice
Charlie
Eve
```

Gemini Chat:

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

◆ Okay, I can help with that. Here is the Python code to define a list of student dictionaries and use a `while` loop to print the names of students who scored more than 75.

I have created the list of student dictionaries and added the `while` loop to print the names of students who scored more than 75. Would you like to modify the list, change the score threshold, or try a different type of loop like a `for` loop?

◆ What can I help you build?

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

Explanation:

Gemini

X

K

explanation

◆

Certainly, I can explain the code in the cells you have visible.

Cell 1 (id: 6ea779a8):

This cell demonstrates how to use the `ClassName` defined in a previous cell.

- It creates an object named `my_object` from the `ClassName`, providing values for `attribute1`, `attribute2`, and an `initial_balance` of 100.
- It then calls the `check_balance()` method to display the initial balance.
- It calls the `deposit(50)` method to add 50 to the balance and shows the updated balance.
- It calls the `withdraw(30)` method to subtract 30 from the balance and shows the updated balance.
- It attempts to call `withdraw(200)` to demonstrate the "Insufficient funds" message because the current balance is less than 200.
- Finally, it calls `check_balance()` again to show the final balance after the operations.

Cell 2 (id: 1780c7f1):

This cell defines a list of dictionaries, where each dictionary represents a student with a "name" and a "score".

- It then initializes a variable `i` to 0, which will be used as an index for the `while` loop.
- The `while i < len(students):` line starts a loop that will continue as long as the value of `i` is less than the total number of students in the list.
- Inside the loop, `student = students[i]` accesses the dictionary at the current index `i` and assigns it to the `student` variable.
- The `if student["score"] > 75:` line checks if the "score" of the current `student` is greater than 75.
- If the score is greater than 75, `print(student["name"])` prints the "name" of that student.
- Finally, `i += 1` increments the index `i` so that the loop moves to the next student in the list.

👍

🔗

◆

What can I help you build?

+

➤

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

◆

✓ 4:06 PM

Python 3

Task 5:

Begin writing a class `ShoppingCart` with an empty items list to generate methods to `add_item`, `remove_item`, and use a loop to calculate the total bill using conditional discounts.

Code and Output:

Al(6).ipynb

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

✓ [27]

```
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)
```

Implement remove item method

Subtask:

Add a method to the `ShoppingCart` class that takes an item's name and removes it from the items list. Handle cases where the item is not found.

Reasoning: Add the `remove_item` method to the `ShoppingCart` class as instructed.

✓ [29]

```
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)

    def remove_item(self, item_name):
        found = False
        for item in self.items:
            if item['name'] == item_name:
                self.items.remove(item)
                print(f'"{item_name}" removed from the shopping cart.')
                found = True
                break
```

Gemini

X

K

Begin writing a class `ShoppingCart` with an empty items list to generate methods to `add_item`, `remove_item`, and use a loop to calculate the total bill using conditional discounts.

◆

Happy to help! Here is a plan to create a `ShoppingCart` class with methods to manage items and calculate the total bill with conditional discounts.

- Define the `shoppingcart` class:** Create the class with an `__init__` method to initialize an empty list for items.
- Implement `add item` method:** Add a method to the `ShoppingCart` class that takes an item (e.g., a dictionary with name, price, and quantity) and adds it to the items list.
- Implement `remove item` method:** Add a method to the `ShoppingCart` class that takes an item's name and removes it from the items list. Handle cases where the item is not found.
- Implement `calculate total` method:** Add a method that iterates through the items list, calculates the subtotal for each item, applies conditional discounts based on criteria (which we'll need to define), and sums up the total bill.
- Add example usage:** Create an instance of the `ShoppingCart` class, add and remove some items, and call the `calculate_total` method to demonstrate its functionality.
- Finish task:** Review the code and ensure it meets all the requirements.

Generating...

🔄

Preparing

🔄

Define the 'shoppingcart' class

🔄

Implement 'add item' method

🔄

Implement 'remove item' method

🔄

Implement 'calculate total' method

◆

What can I help you build?

+

➤

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

Variables

Terminal

◆

✓ 4:19 PM

Python 3

AI(6).ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[29]

```
        round = True
        break
    if not found:
        print(f'{item_name} not found in the shopping cart.')
```

Implement calculate total method

Subtask:

Add a method that iterates through the items list, calculates the subtotal for each item, applies conditional discounts based on criteria (which we'll need to define), and sums up the total bill.

Reasoning: Implement the `calculate_total` method in the `ShoppingCart` class as described in the instructions, including iterating through items, calculating subtotals, applying conditional discounts, and summing up the total.

[31]

```
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)

    def remove_item(self, item_name):
        found = False
        for item in self.items:
            if item['name'] == item_name:
```

Variables Terminal

4:19 PM Python 3

Gemini

Begin writing a class `ShoppingCart` with an empty items list to generate methods to `add_item`, `remove_item`, and use a loop to calculate the total bill using conditional discounts.

Happy to help! Here is a plan to create a `ShoppingCart` class with methods to manage items and calculate the total bill with conditional discounts.

- Define the `shoppingcart` class:** Create the class with an `__init__` method to initialize an empty list for items.
- Implement `add item` method:** Add a method to the `ShoppingCart` class that takes an item (e.g., a dictionary with name, price, and quantity) and adds it to the items list.
- Implement `remove item` method:** Add a method to the `ShoppingCart` class that takes an item's name and removes it from the items list. Handle cases where the item is not found.
- Implement `calculate total` method:** Add a method that iterates through the items list, calculates the subtotal for each item, applies conditional discounts based on criteria (which we'll need to define), and sums up the total bill.
- Add example usage:** Create an instance of the `ShoppingCart` class, add and remove some items, and call the `calculate_total` method to demonstrate its functionality.
- Finish task:** Review the code and ensure it meets all the requirements.

What can I help you build?

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

AI(6).ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[31]

```
        if item['name'] == item_name:
            self.items.remove(item)
            print(f'{item_name} removed from the shopping cart.')
            found = True
            break
        if not found:
            print(f'{item_name} not found in the shopping cart.')
```

def calculate_total(self):
total = 0
for item in self.items:
 subtotal = item['price'] * item['quantity']
 # Apply conditional discounts
 if item['quantity'] > 5: # Example: 10% discount for quantity > 5
 discount = subtotal * 0.10
 subtotal -= discount
 print(f'Applied 10% discount to {item['name']} for quantity over 5.')
 elif item['price'] > 50: # Example: 5% discount for price > 50
 discount = subtotal * 0.05
 subtotal -= discount
 print(f'Applied 5% discount to {item['name']} for price over 50.')
 total += subtotal
return total

Add example usage

Subtask:

Variables Terminal

4:19 PM Python 3

Gemini

that takes an item's name and removes it from the items list. Handle cases where the item is not found.

- Implement `calculate total` method:** Add a method that iterates through the items list, calculates the subtotal for each item, applies conditional discounts based on criteria (which we'll need to define), and sums up the total bill.
- Add example usage:** Create an instance of the `ShoppingCart` class, add and remove some items, and call the `calculate_total` method to demonstrate its functionality.
- Finish task:** Review the code and ensure it meets all the requirements.

Generating...

- Preparing
- Define the `'shoppingcart'` class
- Implement `'add item'` method
- Implement `'remove item'` method
- Implement `'calculate total'` method
- Add example usage
- Finish task

What can I help you build?

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

AI(6).ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[31]

```
# Create an instance of the ShoppingCart class
my_cart = ShoppingCart()

# Create item dictionaries
item1 = {'name': 'Laptop', 'price': 1000, 'quantity': 1}
item2 = {'name': 'Mouse', 'price': 25, 'quantity': 10}
item3 = {'name': 'Keyboard', 'price': 75, 'quantity': 2}
item4 = {'name': 'Monitor', 'price': 300, 'quantity': 1}

# Add items to the shopping cart
my_cart.add_item(item1)
my_cart.add_item(item2)
my_cart.add_item(item3)
my_cart.add_item(item4)

# Remove an item
my_cart.remove_item('Mouse')

# Calculate the total
total_bill = my_cart.calculate_total()

# Print the total
print(f'The total bill is: ${total_bill:.2f}')
```

Mouse removed from the shopping cart.
Applied 5% discount to Laptop for price over 50.
Applied 5% discount to Keyboard for price over 50.
Applied 5% discount to Monitor for price over 50.
The total bill is: \$1377.50

Variables Terminal

4:19 PM Python 3

Gemini

that takes an item's name and removes it from the items list. Handle cases where the item is not found.

- Implement `calculate total` method:** Add a method that iterates through the items list, calculates the subtotal for each item, applies conditional discounts based on criteria (which we'll need to define), and sums up the total bill.
- Add example usage:** Create an instance of the `ShoppingCart` class, add and remove some items, and call the `calculate_total` method to demonstrate its functionality.
- Finish task:** Review the code and ensure it meets all the requirements.

Generating...

- Preparing
- Define the `'shoppingcart'` class
- Implement `'add item'` method
- Implement `'remove item'` method
- Implement `'calculate total'` method
- Add example usage
- Finish task

What can I help you build?

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

Explanation:

Gemini

✕

...

K

Explanation

✦

Certainly, I can explain the content of the cells you have visible.

Cell 1 (id: ccb80eed):

This is a Markdown cell that introduces the subtask of adding example usage to the `ShoppingCart` class. It explains that the goal is to create an instance of the class, add and remove items, and call the `calculate_total` method to show how it works.

Cell 2 (id: e4ec64e6):

This is a Markdown cell providing the reasoning for the next code cell. It states that the next step is to create an instance of the `ShoppingCart` class, define item dictionaries, add and remove items, calculate the total, and print the result.

Cell 3 (id: 2d979ea4):

This is a code cell that demonstrates the usage of the `ShoppingCart` class.

- It creates an instance of the `ShoppingCart` class called `my_cart`.
- It defines four dictionaries (`item1`, `item2`, `item3`, `item4`) representing different items with their names, prices, and quantities.
- It adds all four items to the `my_cart` using the `add_item` method.
- It then removes the item named 'Mouse' from the cart using the `remove_item` method.
- It calls the `calculate_total()` method on `my_cart` to calculate the total bill after adding and removing items, and stores the result in `total_bill`.
- Finally, it prints the `total_bill` formatted as a currency value with two decimal places.

Cell 4 (id: d38aac2b):

This is a Markdown cell that provides a summary of the task, key findings, and insights or next steps.

- It summarizes that the `ShoppingCart` class was successfully created with methods for adding, removing, and calculating the total with discounts.
- It lists key findings, such as the successful implementation of each method and the details of the conditional discounts applied (10% for quantity > 5 and 5% for price > \$50).
- It suggests insights or next steps, including adding validation to the `add_item` method and considering more complex discount rules.

◆

What can I help you build?

⊕ ▶

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