

AI ASSISTED CODING

LAB-6

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.

colab.research.google.com/drive/1FJMLJ7zhNogB2-9Gw9D0qiTz151FNWaz#scrollTo=5ba71d79

Untitled10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
student_obj = student(name, roll_number, marks)

# Display the details of the student
print("\nStudent Details:")
student_obj.display_details()

# You would need to calculate the average_marks based on other students
# for a meaningful comparison. For now, let's use a placeholder average.
# In a real scenario, you would have a list of students and calculate the average from there
placeholder_average_marks = 75.0

# Check if the student's marks are above average
if student_obj.is_above_average(placeholder_average_marks):
    print(f"\n{student_obj.name}'s marks are above the placeholder average.")
else:
    print(f"\n{student_obj.name}'s marks are not above the placeholder average.")
```

Enter student name: sweety
Enter roll number: 25
Enter marks: 93

Student Details:
Student Name: sweety
Roll Number: 25
Marks: 93.0

sweety's marks are above the placeholder average.

Variables Terminal

Gemini

1. Class Definition:

- `class student`: This line defines a new class named `student`. Classes are blueprints for creating objects (instances).

2. Constructor (`__init__`):

- `def __init__(self, name, roll_number, marks)`: This is the constructor method. It's automatically called when you create a new `student` object.
- `self`: Refers to the instance of the class being created.
- `name`, `roll_number`, `marks`: These are the parameters passed when creating a `student` object.
- `self.name = name`, `self.roll_number = roll_number`, `self.marks = marks`: These lines assign the values passed as arguments to the corresponding attributes of the `student` object.

3. `display_details` Method:

- `def display_details(self)`: This method is defined within the `student` class.
- `self`: Refers to the instance of the class on which the method is called.
- `print(f"Student Name: {self.name}")`: This line prints the name of the student using the `name` attribute of the object. The `f` before the string indicates a formatted string literal, allowing you to embed variable values within curly braces `{}`.
- Similarly, it prints the `roll_number` and `marks` of the student.

4. `is_above_average` Method:

What can I help you build?

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

Rainy days ahead
29°C

Search

15:28
01-09-2025

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.

Untitled10.ipynb - Colab

colab.research.google.com/drive/1FJMLJ7zhNogB2-9Gw9D0qiTz151FNWAZ#scrollTo=4a82a0e7

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

a's marks are not above the placeholder average.

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

for number in my_list:
if number % 2 == 0: # Check if the number is even
square = number ** 2
print(f"The square of {number} is {square}")

The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100

Gemini

Certainly! Here's an explanation of the code that allows you to input a list of numbers and then calculates the square of the even numbers:

1. Get User Input:

input_list_str = input("Enter a list of numbers separated by spaces: ")

This line prompts you to enter a list of numbers. The numbers should be separated by spaces. The input you provide is stored as a single string in the input_list_str variable.

2. Split the Input String:

my_list_str = input_list_str.split()

The split() method is used to split the input string (input_list_str) into a list of substrings. By default, split() splits the string at whitespace (spaces, tabs, newlines). So, if you enter "1 2 3 4", my_list_str will become ['1', '2', '3', '4'].

3. Convert Strings to Integers:

my_list = [int(num) for num in my_list_str]

This is a list comprehension. It iterates through each string element (num) in the my_list_str list and converts each string to an integer using int(). The resulting integers are collected into a new list called my_list.

4. Print the Entered List:

What can I help you build?

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

Variables Terminal

29°C Mostly cloudy

Search

15:48 01-09-2025

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.

The screenshot displays a Google Colab notebook titled 'Untitled10.ipynb' with a single code cell. The code implements a simple bank account system. It starts by creating a `bank_account` class with an `__init__` method that takes `account_holder_name` and `initial_balance` as arguments. Below the class definition, there's a loop that prompts the user to choose an action (1. Deposit, 2. Withdraw, 3. Check Balance, 4. Exit). Depending on the choice, it calls the corresponding methods on the `my_account` instance. If the user enters an invalid choice, it prints an error message.

```
# Create a bank account
class bank_account:
    def __init__(self, account_holder_name, initial_balance):
        self.account_holder_name = account_holder_name
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

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

# Interact with the account
while True:
    print("\nChoose an action:")
    print("1. Deposit")
    print("2. Withdraw")
    print("3. Check Balance")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == '1':
        amount = float(input("Enter deposit amount: "))
        my_account.deposit(amount)
    elif choice == '2':
        amount = float(input("Enter withdrawal amount: "))
        my_account.withdraw(amount)
    elif choice == '3':
        my_account.check_balance()
    elif choice == '4':
        print("Exiting.")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 4.")
```

The Gemini chat interface on the right explains the code in two parts:

- Create a Bank Account:**
 - `account_holder_name = input("Enter account holder name: ")`: This line prompts you to enter the name of the account holder and stores it in the `account_holder_name` variable.
 - `initial_balance = float(input("Enter initial balance: "))`: This line prompts you to enter the initial balance for the account. `float()` is used to convert your input into a floating-point number (allowing for decimal values). The value is stored in `initial_balance`.
 - `my_account = bank_account(account_holder_name, initial_balance)`: This line creates an instance (an object) of the `bank_account` class. It passes the `account_holder_name` and `initial_balance` as arguments to the `__init__` constructor of the class, which initializes the new bank account object.
- Interact with the Account (Loop):**
 - `while True:`: This starts an infinite loop. The code inside this loop will continue to run until explicitly broken out of. This allows you to perform multiple operations on the

At the bottom of the Gemini chat, there's a prompt: "What can I help you build?" and a note: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

The bottom of the screen shows the Windows taskbar with the time 15:59 on 01-09-2025, and the system tray indicating 29°C and mostly cloudy weather.

Untitled10.ipynb - Colab

colab.research.google.com/drive/1FJMLJ7zhNogB2-9Gw9D0qiTz151FNWAZ#scrollTo=d31aa5df

Untitled10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

Share Gemini

Enter account holder name: `sweety`
Enter initial balance: `125000`

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): `3`
Account holder: `sweety`, Current balance: `125000.0`

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): `2`
Enter withdrawal amount: `4`
Withdrew `4.0`. New balance: `124996.0`

Choose an action:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice (1-4): `4`
Exiting.

3. Handle User Choice:

- `if choice == '1':` : If your choice is '1', this block of code is executed for depositing.
 - `amount = float(input("Enter deposit amount: "))` : Prompts you to enter the deposit amount and converts it to a float.
 - `my_account.deposit(amount)` : Calls the `deposit()` method of the `my_account` object, passing the entered amount.
- `elif choice == '2':` : If your choice is '2', this block of code is executed for withdrawing.
 - `amount = float(input("Enter withdrawal amount: "))` : Prompts you to enter the withdrawal amount and converts it to a float.
 - `my_account.withdraw(amount)` : Calls the `withdraw()` method of the `my_account` object, passing the entered amount.
- `elif choice == '3':` : If your choice is '3', this block of code is executed for checking the balance.
 - `my_account.check_balance()` : Calls the `check_balance()` method of the `my_account` object to display the current balance.
- `elif choice == '4':` : If your choice is '4', this block of code is executed to exit the loop.

What can I help you build?

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

Variables Terminal

29°C Mostly cloudy 16:00 01-09-2025

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.

The screenshot displays a Google Colab notebook environment. The notebook is titled "Untitled10.ipynb" and is open in a web browser. The code cell contains a Python script that prompts the user to enter the number of students, then enters a loop to collect student names and scores. After the loop, it prints the names of students who scored more than 75. The output of the code shows that two students, "hani" and "siri", scored more than 75.

```
students = []
num_students = int(input("Enter the number of students: "))

i = 0
while i < num_students:
    name = input(f"Enter name for student {i+1}: ")
    score = int(input(f"Enter score for student {i+1}: "))
    students.append({"name": name, "score": score})
    i += 1

print("\nStudents who scored more than 75:")
index = 0
while index < len(students):
    student = students[index]
    if student["score"] > 75:
        print(student["name"])
    index += 1
```

Enter the number of students: 2
Enter name for student 1: hani
Enter score for student 1: 78
Enter name for student 2: siri
Enter score for student 2: 86

Students who scored more than 75:
hani
siri

The Gemini chat interface on the right provides an explanation of the code. It starts with a prompt "explain the code" and responds with a detailed breakdown of the code's logic, including the initialization of the list, the input for the number of students, and the while loop for collecting student details. It also includes a prompt "What can I help you build?" and a disclaimer that Gemini can make mistakes.

explain the code

Certainly! Here's an explanation of the code in the cell above that allows you to input student details and then prints the names of students who scored more than 75:

- 1. Initialize an Empty List:**
 - `students = []`: This line creates an empty list called `students`. This list will store the dictionaries, where each dictionary represents a student.
- 2. Get the Number of Students:**
 - `num_students = int(input("Enter the number of students: "))`: This line prompts you to enter the total number of students you want to add to the list. `int()` converts your input into an integer, which is stored in the `num_students` variable.
- 3. Input Student Details (First While Loop):**
 - `i = 0`: Initializes a counter variable `i` to 0. This variable will be used to keep track of the current student number.
 - `while i < num_students:`: This starts a `while` loop that will continue to execute as long as the value of `i` is less than the `num_students` you entered. This loop is used to get details for each student.
 - `name = input(f"Enter name for student {i+1}: ")`: Inside the loop, this line prompts you to enter the name for the current student. The `f-string` `f"Enter name for student {i+1}: "` is used to format the prompt.

What can I help you build?

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

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.

```
# Create a shopping cart
my_cart = shopping_cart()

# Add items to the cart
while True:
    item_name = input("Enter item name to add (or 'done' to finish): ")
    if item_name.lower() == 'done':
        break
    try:
        price = float(input(f"Enter price for {item_name}: "))
        my_cart.add_item(item_name, price)
    except ValueError:
        print("Invalid price. Please enter a number.")

print("\nItems in your cart:")
for item in my_cart.items:
    print(f"- {item['name']}: ${item['price']:.2f}")
```

```
Enter item name to add (or 'done' to finish): eraser
Enter price for eraser: 10
Added eraser to the cart.
Enter item name to add (or 'done' to finish): pen
Enter price for pen: 30
Added pen to the cart.
Enter item name to add (or 'done' to finish): sharpener
Enter price for sharpener: 5
Added sharpener to the cart.
Enter item name to add (or 'done' to finish): done

Items in your cart:
- eraser: $10.00
- pen: $30.00
- sharpener: $5.00
```


P explain the code

◆ Certainly! Here's an explanation of the code in the cell above that allows you to add items to your shopping cart using your input:

1. Create a Shopping Cart:

- `my_cart = shopping_cart()`: This line creates an instance (an object) of the `shopping_cart` class. This object will represent your shopping cart.

2. Add Items to the Cart (Loop):

- `while True:`: This starts an infinite loop, allowing you to add multiple items to the cart until you decide to stop.
- `item_name = input("Enter item name to add (or 'done' to finish): ")`: This line prompts you to enter the name of the item you want to add to the cart. It also tells you to type 'done' to finish adding items. Your input is stored in the `item_name` variable.
- `if item_name.lower() == 'done':`: This checks if you entered 'done' (case-insensitive). If you did, the `break` statement is executed, which exits the `while` loop.
- `try...except ValueError:`: This is a `try-except` block used for error handling. It attempts to execute the code within the `try` block. If a `ValueError` occurs (which would happen if you enter something that cannot be converted to a float for the price), the code within the `except` block is executed.
 - `price = float(input(f"Enter price for {item_name}: "))`: Inside the `try` block, this line prompts you to enter the price for the item you just named. `float()` converts your input to a floating-point number.
 - `my_cart.add_item(item_name, price)`: This line calls the `add_item()` method of the `my_cart` object, passing the `item_name` and `price` you entered as arguments. This adds the item to the cart's internal list.
 - `print("Invalid price. Please enter a number.")`: Inside the `except` block, this line prints an error message if the price you entered was not a valid number.

3. Print Items in the Cart:

- `print("\nItems in your cart:")`: After the loop finishes (when you enter 'done'), this line prints a header indicating the items that are currently in your cart.
- `for item in my_cart.items:`: This starts a `for` loop that iterates through each dictionary (representing an item) in the `items` list of the `my_cart` object.
- `print(f"- {item['name']}: ${item['price']:.2f}")`: Inside this loop, this line prints the name and price of each item in the cart. `item['name']` accesses the name from the dictionary, and `item['price']` accesses the price. `:.2f` formats the price to two decimal places.

In summary, this code allows you to interactively build a shopping cart by repeatedly entering item names and prices until you type 'done'. It then prints a list of the items you added to the cart.