

# LAB ASSIGNMENT – 6

NAME : P.HANSINI REDDY

ROLL NO : 2403A510D5

COURSE : AI ASSISTED CODING

BATCH :01

## QUESTIONS :

Duration		2 Hours	Applicable to Batches	
Assignment Number: 6.3(Present assignment number)/24(Total number of assignments)				
Q.No.	Question			Expected Time to complete
1	<p>Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals</p> <p><b>Lab Objectives:</b></p> <ul style="list-style-type: none"><li>To explore AI-powered auto-completion features for core Python constructs.</li><li>To analyze how AI suggests logic for class definitions, loops, and conditionals.</li><li>To evaluate the completeness and correctness of code generated by AI assistants.</li></ul> <p><b>Lab Outcomes (LOs):</b></p>			Week3 - Wednesday

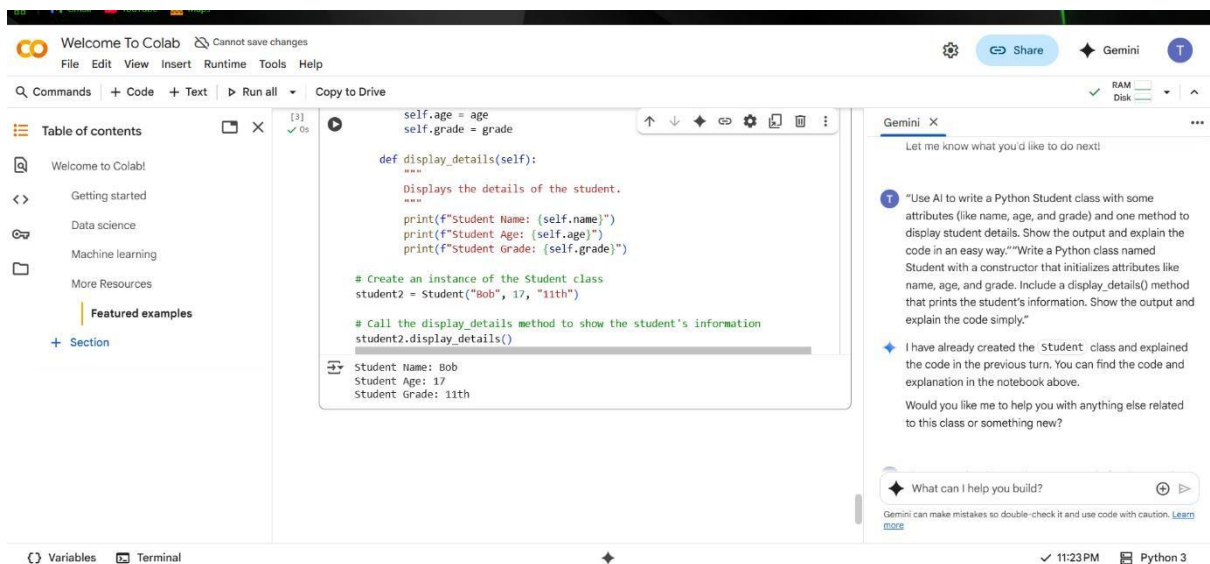
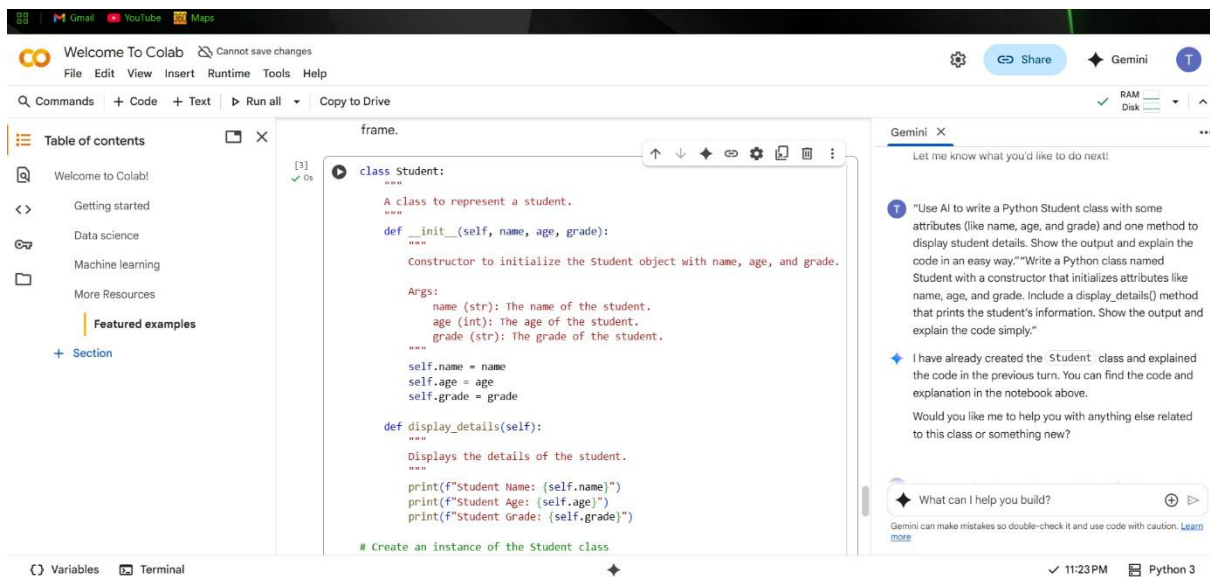
	<p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>• Use AI tools to generate and complete class definitions and methods.</li> <li>• Understand and assess AI-suggested loops for iterative tasks.</li> <li>• Generate conditional statements through prompt-driven suggestions.</li> <li>• Critically evaluate AI-assisted code for correctness and clarity.</li> </ul> <p><b>Task Description#1 (Classes)</b></p> <ul style="list-style-type: none"> <li>• Use AI to complete a Student class with attributes and a method.</li> <li>• Check output</li> <li>• Analyze the code generated by AI tool</li> </ul> <p><b>Expected Output#1</b></p> <ul style="list-style-type: none"> <li>• Class with constructor and display_details() method</li> </ul> <p><b>Task Description#2 (Loops)</b></p> <ul style="list-style-type: none"> <li>• Prompt AI to complete a function that prints the first 10 multiples of a number using a loop.</li> <li>• Analyze the generated code</li> <li>• Ask AI to generate code using other controlled looping</li> </ul> <p><b>Expected Output#2</b></p> <ul style="list-style-type: none"> <li>• Correct loop-based implementation</li> </ul> <p><b>Task Description#3 (Conditional Statements)</b></p> <ul style="list-style-type: none"> <li>• Ask AI to write nested if-elif-else conditionals to classify age groups.</li> <li>• Analyze the generated code</li> <li>• Ask AI to generate code using other conditional statements</li> </ul> <p><b>Expected Output#3</b></p> <ul style="list-style-type: none"> <li>• Age classification function with appropriate conditions and with explanation</li> </ul> <p><b>Task Description#4 (For and While loops)</b></p> <ul style="list-style-type: none"> <li>• Generate a sum_to_n() function to calculate sum of first n numbers</li> <li>• Analyze the generated code</li> <li>• Get suggestions from AI with other controlled looping</li> </ul>	
--	---	--

<ul style="list-style-type: none"> <li>Analyze the generated code</li> <li>Ask AI to generate code using other conditional statements</li> </ul>					
<b>Expected Output#3</b>					
<ul style="list-style-type: none"> <li>Age classification function with appropriate conditions and with explanation</li> </ul>					
<b>Task Description#4 (For and While loops)</b>					
<ul style="list-style-type: none"> <li>Generate a sum_to_n() function to calculate sum of first n numbers</li> <li>Analyze the generated code</li> <li>Get suggestions from AI with other controlled looping</li> </ul>					
<b>Expected Output#4</b>					
<ul style="list-style-type: none"> <li>Python code with explanation</li> </ul>					
<b>Task Description#5 (Class)</b>					
<ul style="list-style-type: none"> <li>Use AI to build a BankAccount class with deposit, withdraw, and balance methods.</li> <li>Analyze the generated code</li> <li>Add comments and explain code</li> </ul>					
<b>Expected Output#5</b>					
<ul style="list-style-type: none"> <li>Python code with explanation</li> </ul>					
<p><b>Note:</b> Report should be submitted a word document for all tasks in a single document with prompts, comments &amp; code explanation, and output and if required, screenshots</p>					
<b>Evaluation Criteria:</b>					
<table border="1"> <thead> <tr> <th>Criteria</th><th>Max Marks</th></tr> </thead> <tbody> <tr> <td>Class</td><td>1.0</td></tr> </tbody> </table>		Criteria	Max Marks	Class	1.0
Criteria	Max Marks				
Class	1.0				

## TASK 1

### PROMPT :

Use AI to write a Python Student class with some attributes (like name, age, and grade) and one method to display student details. Show the output and explain the code in an easy way.”“Write a Python class named Student with a constructor that initializes attributes like name, age, and grade. Include a display\_details() method that prints the student’s information. Show the output and explain the code simply

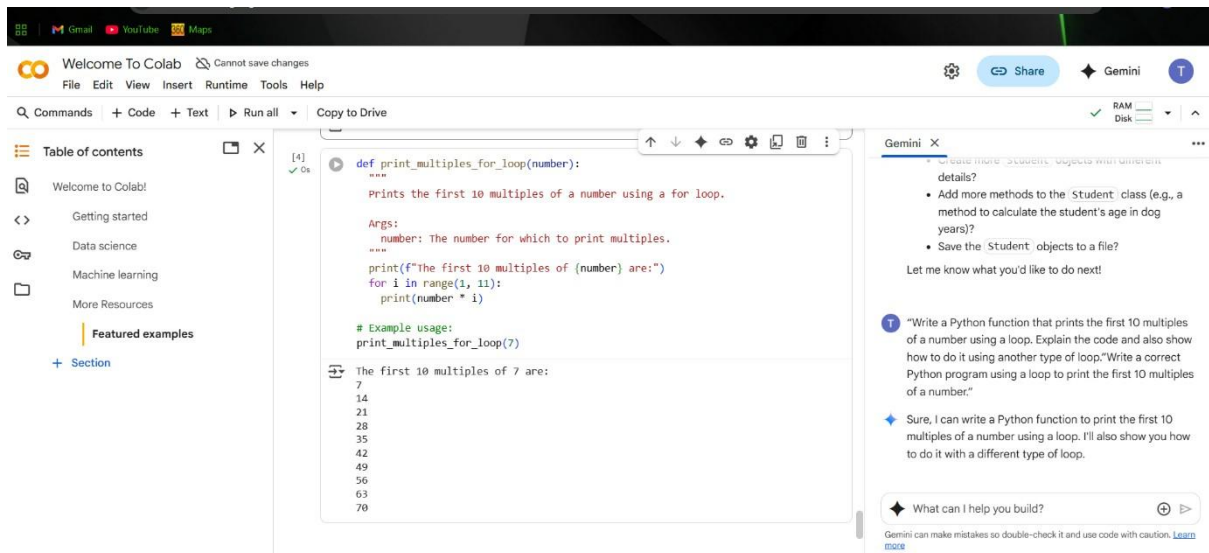


## TASK 2

### Prompt

Write a Python function that prints the first 10 multiples of a number using a loop. Explain the code and also show how to do it using another type of loop." "Write a correct Python program using a loop to print the first 10 multiples of a number

### Output



## TASK- 3

### PROMPT :

Write a Python program using nested if-elif-else statements to classify age groups. Explain the code and also show how to do it using other types of conditionals.”“Write a Python function to classify age groups with proper conditions and explain how the code works in a simple way.

### Explanation :

- **Define the function:** Create a Python function that takes age as input.
- **Implement nested if-elif-else:** Use nested if-elif-else statements within the function to classify age groups (e.g., Child, Teenager, Adult, Senior).
- **Add explanation:** Provide a simple explanation of the code and how the nested if-elif-else statements work.

- **Implement alternative conditional:** Show an alternative way to classify age groups using a different conditional structure (e.g., a dictionary or a series of simple if-elifelse statements without nesting).

## OUTPUT:

The screenshot shows a Google Colab notebook with a table of contents on the left. The main code cell contains a Python function `classify_age_nested(age)` that uses nested if-elif-else statements to classify age groups. The function is called with ages 10, 15, 30, and 70, resulting in the output: Child, Teenager, Adult, Senior. To the right, the Gemini chat interface is open, showing a prompt to write a Python program using nested if-elif-else statements to classify age groups. The Gemini response provides a plan to generate a Python program that classifies age groups using conditional statements, along with an explanation and an alternative approach. The plan includes: Define the function: Create a Python function that takes age as input. Implement nested if-elif-else: Use nested if-elif-else statements within the function to classify age groups (e.g., Child, Teenager, Adult, Senior). Add explanation: Provide a simple explanation of the code and how the nested if-elif-else statements work.

```
def classify_age_nested(age):
    """
    Classifies age into groups using nested if-elif-else statements.

    Args:
        age: The age of the person.
    """
    if age < 13:
        print("Child")
    elif age < 20:
        print("Teenager")
    elif age < 65:
        print("Adult")
    else:
        print("Senior")

classify_age_nested(10)
classify_age_nested(15)
classify_age_nested(30)
classify_age_nested(70)
```

Child  
Teenager  
Adult  
Senior

Gemini: "Write a Python program using nested if-elif-else statements to classify age groups. Explain the code and also show how to do it using other types of conditionals." "Write a Python function to classify age groups with proper conditions and explain how the code works in a simple way."

Happy to help! Here is a plan to generate a Python program that classifies age groups using conditional statements, along with an explanation and an alternative approach.

- **Define the function:** Create a Python function that takes age as input.
- **Implement nested if-elif-else:** Use nested if-elif-else statements within the function to classify age groups (e.g., Child, Teenager, Adult, Senior).
- **Add explanation:** Provide a simple explanation of the code and how the nested if-elif-else statements work.

What can I help you build?

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

## TASK 4

### PROMPT :

Write a Python function `sum_to_n()` to calculate the sum of the first n numbers. Explain the code and show how to do it using other loops. Python code with explanation

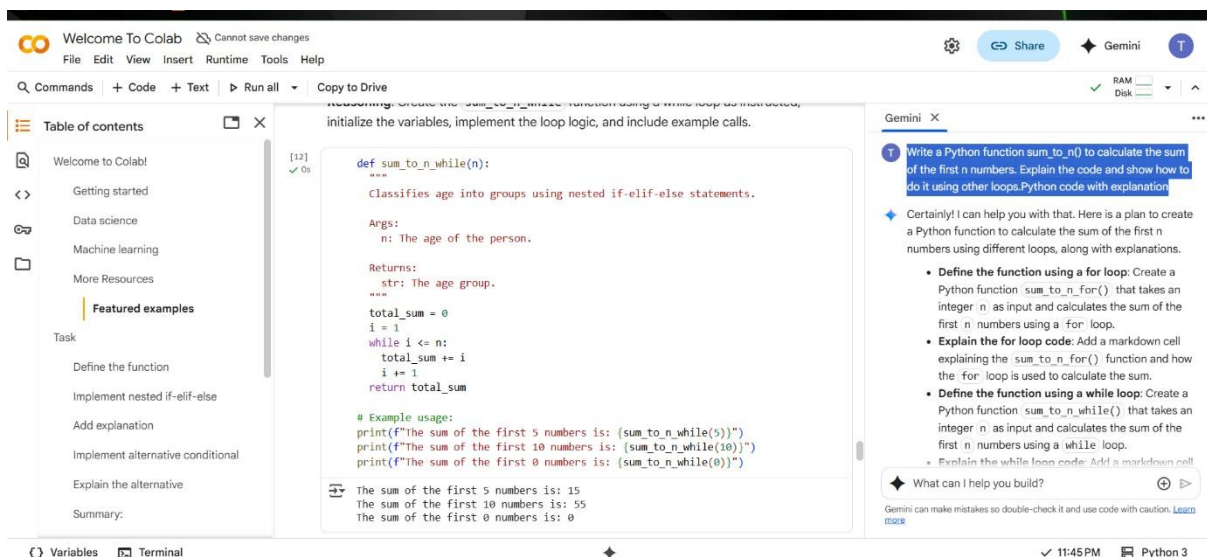
Explanation :

- **Define the function using a for loop:** Create a Python function `sum_to_n_for()` that takes an integer n as input

and calculates the sum of the first n numbers using a for loop.

- **Explain the for loop code:** Add a markdown cell explaining the `sum_to_n_for()` function and how the for loop is used to calculate the sum.
- **Define the function using a while loop:** Create a Python function `sum_to_n_while()` that takes an integer n as input and calculates the sum of the first n numbers using a while loop.
- **Explain the while loop code:** Add a markdown cell explaining the `sum_to_n_while()` function and how the while loop is used.

## OUTPUT:



The screenshot shows a Google Colab notebook interface. The main code cell contains a Python function `sum_to_n_while(n)` that calculates the sum of the first n numbers using a while loop. The function is defined as follows:

```
def sum_to_n_while(n):  
    """  
    Classifies age into groups using nested if-elif-else statements.  
    Args:  
        n: The age of the person.  
    Returns:  
        str: The age group.  
    """  
    total_sum = 0  
    i = 1  
    while i <= n:  
        total_sum += i  
        i += 1  
    return total_sum  
  
# Example usage:  
print(f"The sum of the first 5 numbers is: {sum_to_n_while(5)}")  
print(f"The sum of the first 10 numbers is: {sum_to_n_while(10)}")  
print(f"The sum of the first 0 numbers is: {sum_to_n_while(0)}")
```

The output of the code cell shows the results of the function calls:

```
The sum of the first 5 numbers is: 15  
The sum of the first 10 numbers is: 55  
The sum of the first 0 numbers is: 0
```

On the right side of the notebook, there is a Gemini chat window. The user has asked: "Write a Python function sum\_to\_n() to calculate the sum of the first n numbers. Explain the code and show how to do it using other loops. Python code with explanation". The Gemini response provides a plan to create a Python function to calculate the sum of the first n numbers using different loops, along with explanations. The plan includes three main steps:

- **Define the function using a for loop:** Create a Python function `sum_to_n_for()` that takes an integer `n` as input and calculates the sum of the first `n` numbers using a `for` loop.
- **Explain the for loop code:** Add a markdown cell explaining the `sum_to_n_for()` function and how the `for` loop is used to calculate the sum.
- **Define the function using a while loop:** Create a Python function `sum_to_n_while()` that takes an integer `n` as input and calculates the sum of the first `n` numbers using a `while` loop.

The Gemini chat window also includes a prompt: "Explain the while loop code. Add a markdown cell".

## TASK 5 :



## PROMPT :

Create a Python class called BankAccount that has methods for depositing money, withdrawing money, and checking the balance. Include error checks for negative amounts and insufficient funds. Add comments and explain each method clearly."

## Explanation :

- **Define the bankaccount class:** Create a Python class named BankAccount with a constructor to initialize the account balance.
- **Implement deposit method:** Add a deposit method to the BankAccount class that takes an amount as input and adds it to the balance. Include comments.
- **Implement withdraw method:** Add a withdraw method to the BankAccount class that takes an amount as input and subtracts it from the balance, ensuring there are sufficient funds. Include comments.
- **Implement balance method:** Add a balance method to the BankAccount class that returns the current account balance. Include comments.
- **Add example usage:** Create instances of the BankAccount class and demonstrate the usage of the deposit, withdraw, and balance methods.



- **Explain the code:** Provide a detailed explanation of the BankAccount class, its methods, and how they work together.
- **Explain the output:** Explain the output generated from the example usage, showing how the balance changes after each operation.

The screenshot shows a Google Colab notebook with the following code in the first cell:

```
class BankAccount:
    """Represents a simple bank account."""

    def __init__(self, account_holder, initial_balance=0.0):
        """
        Initializes a new bank account.

        Args:
            account_holder (str): The name of the account holder.
            initial_balance (float): The starting balance of the account. Defaults to 0.0.
        """
        # Ensure initial balance is not negative
        if initial_balance < 0:
            print("Warning: Initial balance cannot be negative. Setting balance to 0.")
            self.balance = 0.0
        else:
            self.balance = initial_balance
        self.account_holder = account_holder
        print(f"Account created for {self.account_holder} with an initial balance of ${self.balance:.2f}")

    def deposit(self, amount):
        """
        Deposits money into the account.

        Args:
            amount (float): The amount to deposit.
        """
        # Check if the deposit amount is valid
        if amount <= 0:
```

Gemini's response on the right explains the code:

Create a Python class called BankAccount that has methods for depositing money, withdrawing money, and checking the balance. Include error checks for negative amounts and insufficient funds. Add comments and explain each method clearly.

Absolutely! I can create a BankAccount class with methods for depositing, withdrawing, and checking the balance, including the error checks you mentioned. I'll also add comments and explanations for each method.

The BankAccount class has been successfully created with the requested methods and error handling, and the example usage demonstrates its functionality.

The screenshot shows the same Google Colab notebook, but the code in the first cell is now complete:

```
        # Check if the deposit amount is valid
        if amount <= 0:
            print("Error: Deposit amount must be positive.")
        else:
            self.balance += amount
            print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")

    def withdraw(self, amount):
        """
        Withdraws money from the account.

        Args:
            amount (float): The amount to withdraw.
        """
```

Gemini's response on the right is identical to the first screenshot, explaining the code and providing a summary of the class's functionality.

Welcome To Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Copy to Drive

Table of contents

Welcome to Colab!

Getting started

Data science

Machine learning

More Resources

Featured examples

+ Section

```
# Check if the withdrawal amount is valid
if amount <= 0:
    print("Error: Withdrawal amount must be positive.")
# check for sufficient funds
elif amount > self.balance:
    print("Error: Insufficient funds.")
else:
    self.balance -= amount
    print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")

def check_balance(self):
    """
    Displays the current account balance.
    """
    print(f"Current balance for {self.account_holder}: ${self.balance:.2f}")

# Example usage:
my_account = BankAccount("John Doe", 1000.00)
my_account.deposit(500.00)
my_account.withdraw(200.00)
my_account.check_balance()
my_account.withdraw(1500.00) # Example of insufficient funds
my_account.deposit(-100.00) # Example of invalid deposit

Account created for John Doe with an initial balance of $1000.00
Deposited $500.00. New balance: $1500.00
Withdrew $200.00. New balance: $1300.00
Current balance for John Doe: $1300.00
Error: Insufficient funds.
Error: Deposit amount must be positive.
```

Gemini

Create a Python class called BankAccount that has methods for depositing money, withdrawing money, and checking the balance. Include error checks for negative amounts and insufficient funds. Add comments and explain each method clearly.

Absolutely! I can create a BankAccount class with methods for depositing, withdrawing, and checking the balance, including the error checks you mentioned. I'll also add comments and explanations for each method.

The BankAccount class has been successfully created with the requested methods and error handling, and the example usage demonstrates its functionality.

What can I help you build?

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

Variables

Terminal

2:23 PM Python 3