

# Assignment-6.3

**Name:** M. Preethi Meghana

**Ht.no:** 2303A51584

**Batch:** 05

## Task-01:

Classes (Student Class)

You are developing a simple student information management module.

## Prompt:

The screenshot shows a user interface for a large language model. At the top, there are three buttons: "How can I install Python libraries?", "Load data from Google Drive", and "Show an example of training". Below these buttons is a text input area containing the following prompt:

develop a simple student information management module. The class should include attributes such as name, roll number, and branch. Add a method `display_details` to print student information.

At the bottom left of the input area is a plus sign (+). To the right of the input area, it says "Gemini 2.5 Flash" followed by a dropdown arrow and a right-pointing arrow. On the far right, there is a vertical list of suggestions starting with "• dialog" and ending with "langs".

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

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

Example Usage:
# Create a student object
student1 = Student("Alice", "CS001", "Computer Science")

# Display student details
print("Student 1 Details:")
student1.display_details()

print("\n---\n")

student2 = Student("Bob", "EC005", "Electronics and Communication")
print("Student 2 Details:")
student2.display_details()
```

Release notes

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

**2026-01-20**

- Launched Data Explorer - a new feature that lets you search Kaggle datasets, models, and competitions directly from a Colab notebook!
- Gemini 3 is now available in Colab.
- In addition to being available in VS Code, Colab is also available for use in Antigravity, Cursor, and Windsurf via the Open VSX Registry!
- H100 is being rolled out for more users.
- Launched a new modern design for the Colab UI.

Python package upgrades

- accelerate 1.1.0 → 1.12.0
- astropy 7.1.1 → 7.2.0
- bigframe 2.20.1 → 2.31.0
- cachetools 5.5.2 → 6.2.4
- categoryppr-spark-connect 0.8.3 → 1.0.1
- diffpy 0.35.2 → 0.36.0
- flex 0.1.0.7 → 0.1.2
- google 2.0.3 → 3.0.0
- google-ads 1.17.0 → 1.21.0
- google-auth 2.38.0 → 2.43.0
- google-genai 1.49.0 → 1.55.0
- gradio 5.49.1 → 5.50.0
- holidays 0.84 → 0.88
- humanize 4.14.0 → 4.15.0
- langchain 0.3.27 → 1.2.4
- langsmith 0.4.42 → 0.6.4

## Output:

```
... Student Name: Alice Smith
      Roll Number: CS101
      Branch: Computer Science
```

## Explanation:

- A class named `Student` is created to represent student information.
- The `__init__()` constructor is defined and is called automatically when a `Student` object is created.
- The constructor initializes the student's name, roll number, and branch.
- A method `display_details()` is defined to print the student details.
- A `Student` object is created by passing values for name, roll number, and branch.
- The `display_details()` method is called using the object.

- The student information is displayed on the console.

## Task-02:

### Loops (Multiples of a Number)

#### Scenario

You are writing a utility function to display multiples of a given number.

#### Prompt:



develop a simple student information management module. The class should include attributes such as name, roll number, and branch. Add a

▷ Accept & Run    ✓ Accept    ✘ Cancel

◁ Empty cell ▷

write a utility function to display multiples of a given number. generate a function that prints the first 10 multiples of a given number using a loop.

+

Gemini 2.5 Flash ▾ ▷

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



#### Code:

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** Assignment-6.3.ipynb, File, Edit, View, Insert, Runtime, Tools, Help.
- Toolbar:** Commands, Code, Text, Run all.
- Code Cell:**

```
def display_multiples(number):
    print(f"First 10 multiples of {number}:")
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")

# Example usage:
display_multiples(5)
print("\n---\n")
display_multiples(7)
```
- Output Cell:**

```
First 10 multiples of 5:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

---
First 10 multiples of 7:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
```
- Release Notes:** Please follow our tricks, and features.
- 2026-01-20:**
  - Launched datasets, Gemini 3.0
  - In addition to use in Ant H100 is being Launched
  - Launched or package accelerate (L7), copy 7.1, bidframes (jetool), daraproc-diffusers (flax 0.10.7), google 2.0, google-ac, google-ai, google-ge, gradio 5.4, holidays C, humanize, langchain, langsmith
- AI Interaction:**

P develop a simple student information management module. The class should include attributes such as name, roll number, and branch. Add a method display\_details to print student information.

Sure, here's a simple Python class for student information management, along with an example of how to use it:

Create a list of Student objects | Add a method to the Student class to enroll in a course

What can I help you build?

Gemini 2.5 Flash ▶

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

can help you with today, or would you like to make any modifications to the existing code?

P generate the same functionality using while loop

def display\_multiples(number):  
 ▶ Accept & Run    ✓ Accept    X Cancel

What can I help you build?

+

Gemini 2.5 Flash ▶

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

# Output:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Assignment-6.3.ipynb
- Menu Bar:** File Edit View Insert Runtime Tools Help
- Toolbar:** Commands + Code + Text Run all
- Code Cell [4]:**

```
def display_multiples(number):
    print(f"First 10 multiples of {number}:")
    i = 1
    while i <= 10:
        print(f"{number} x {i} = {number * i}")
        i += 1

# Example usage:
display_multiples(5)
print("\n---\n")
display_multiples(7)
```
- Output Cell:**

```
First 10 multiples of 5:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

---
First 10 multiples of 7:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
```
- Bottom Navigation:** Variables Terminal

## Explanation:

- A function is defined that accepts a number as input.
- A loop is used to repeat the process 10 times.
- In each iteration, the input number is multiplied by the loop counter.
- The result of the multiplication gives one multiple of the number.
- Each multiple is printed on the console.

- The loop continues until the first 10 multiples are displayed.
- When the loop ends, the program stops execution.

### Task-03:

Conditional Statements (Age Classification)

Scenario

You are building a basic classification system based on age.

**Prompt:**

been executed, and their outputs were displayed. Is there anything else I can help you with today, or would you like to make any modifications to the existing code?

[Create more Student objects](#) [Add a new method to the Student class](#) [Explain the display](#)

<> Empty cell X

build a basic classification system based on age. Generate nested if-elif-else conditional statements to classify age groups like child, teenager, adult, senior.

+ Gemini 2.5 Flash ▾ ▶

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

# Code:

The screenshot shows the Gemini AI interface with a code editor and a completion dialog.

**Code Editor Content:**

```
+def classify_age(age):
+    if age < 0:
+        return "Invalid Age"
+    elif age <= 12:
+        return "Child"
+    elif age <= 19:
+        return "Teenager"
+    elif age <= 64:
+        return "Adult"
+    else:
+        return "Senior"

+# Example usage:
+ages = [-5, 5, 15, 30, 75, 100]
+
+for age in ages:
+    classification = classify_age(age)
+    print(f"An individual of age {age} is
```

**Completion Dialog:**

P build a basic classification system based on age. Generate nested if-elif-else conditional statements to classify age groups like child, teenager, adult, senior.

<> Empty cell

▶ Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

+

Gemini 2.5 Flash ▾ ▷

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

Terminal

Commands + Code + Text | Run all

```
[6] ✓ 0s
▶ def classify_age(age):
    if age < 0:
        return "Invalid Age"

    age_categories = [
        {'upper_bound': 12, 'label': 'Child'},
        {'upper_bound': 19, 'label': 'Teenager'},
        {'upper_bound': 64, 'label': 'Adult'}
    ]

    for category_info in age_categories:
        if age <= category_info['upper_bound']:
            return category_info['label']

    return "Senior" # If age is greater than all upper bounds

# Example usage:
ages = [-5, 5, 15, 30, 75, 100]

for age in ages:
    classification = classify_age(age)
    print(f"An individual of age {age} is classified as: {classification}")

...
... An individual of age -5 is classified as: Invalid Age
An individual of age 5 is classified as: Child
An individual of age 15 is classified as: Teenager
An individual of age 30 is classified as: Adult
An individual of age 75 is classified as: Senior
An individual of age 100 is classified as: Senior
```

## Output:

```
▶ def classify_age(age):
    if age < 0:
        return "Invalid Age"
    elif age <= 12:
        return "Child"
    elif age <= 19:
        return "Teenager"
    elif age <= 64:
        return "Adult"
    else:
        return "Senior"

# Example usage:
ages = [-5, 5, 15, 30, 75, 100]

for age in ages:
    classification = classify_age(age)
    print(f"An individual of age {age} is classified as: {classification}")

...
... An individual of age -5 is classified as: Invalid Age
An individual of age 5 is classified as: Child
An individual of age 15 is classified as: Teenager
An individual of age 30 is classified as: Adult
An individual of age 75 is classified as: Senior
An individual of age 100 is classified as: Senior
```

## **Explanation:**

A function is defined that accepts age as an input value.

- The program checks the age using if-elif-else conditions.
- If the age is less than a certain value, it is classified as a child.
- If the age falls in the next range, it is classified as a teenager.
- If the age is in the adult range, it is classified as an adult.
- If the age is above the adult range, it is classified as a senior.
- Only one condition is executed because once a condition is true, the remaining checks are skipped.
- The function returns or prints the appropriate age group.

## **Task-04:**

For and While Loops (Sum of First n Numbers)

Scenario

You need to calculate the sum of the first n natural numbers.

## a) Using For Loop:

### Prompt:

but with a more structured approach.

Like Dislike

[Explain the dictionary-based logic for classify\\_age](#) [Add new methods to the Student class](#)

**Empty cell**

calculate the sum of the first n natural numbers. generate a sum\_to\_n() function using a for loop.

+ Gemini 2.5 Flash ▾ ▶

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

## Code:

```
def sum_to_n(n):
    if n < 1:
        return "Input must be a positive integer."
    total_sum = 0
    for i in range(1, n + 1):
        total_sum += i
    return total_sum

# Example usage:
print(f"Sum of first 5 natural numbers: {sum_to_n(5)}")
print(f"Sum of first 10 natural numbers: {sum_to_n(10)}")
print(f"Sum of first 1 natural numbers: {sum_to_n(1)}")
print(f"Sum of first 0 natural numbers: {sum_to_n(0)}")
```

**P** calculate the sum of the first n natural numbers. generate a sum\_to\_n() function using a for loop.

**Empty cell**

◆ Certainly! Here's a sum\_to\_n function that calculates the sum of the first

▶ Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

+ Gemini 2.5 Flash ▾ ▶

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

# Output:

```
is ◎ def sum_to_n(n):
    if n < 1:
        return "Input must be a positive integer."
    total_sum = 0
    for i in range(1, n + 1):
        total_sum += i
    return total_sum

# Example usage:
print(f"Sum of first 5 natural numbers: {sum_to_n(5)}")
print(f"Sum of first 10 natural numbers: {sum_to_n(10)}")
print(f"Sum of first 1 natural numbers: {sum_to_n(1)}")
print(f"Sum of first 0 natural numbers: {sum_to_n(0)}")

... Sum of first 5 natural numbers: 15
Sum of first 10 natural numbers: 55
Sum of first 1 natural numbers: 1
Sum of first 0 natural numbers: Input must be a positive integer.
```

## b)using While Loop:

### Prompt:

P calculate the sum of the first n natural numbers. generate a sum\_to\_n() function using a for loop.

<> Empty cell

Explain the 'sum\_to\_n' function code Create a 'sum\_to\_n' function using a while loop C

def sum\_to\_n(n): X

suggest an alternative implementation using a while loop

+

Gemini 2.5 Flash ▾ ▶

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

### Code:

The screenshot shows the Gemini AI interface with a Python code editor. The code defines a function `sum_to_n` to calculate the sum of natural numbers up to `n`. The code uses both a `for` loop and a `while` loop implementation. Example usage is shown with `print` statements for `n=5, 10, 1, 0`.

```

def sum_to_n(n):
    if n < 1:
        return "Input must be a positive integer."
    total_sum = 0
    for i in range(1, n + 1):
        i = 1
        while i <= n:
            total_sum += i
            i += 1
    return total_sum

# Example usage:
print(f"Sum of first 5 natural numbers: {sum_to_n(5)}")
print(f"Sum of first 10 natural numbers: {sum_to_n(10)}")
print(f"Sum of first 1 natural numbers: {sum_to_n(1)}")
print(f"Sum of first 0 natural numbers: {sum_to_n(0)}")

```

**P** suggest an alternative implementation using a while loop

def sum\_to\_n(n):  
Certainly! Here's an alternative implementation of the `sum_to_n` function using a `while` loop to calculate the sum of the first '`n`' natural numbers:

▶ Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

Gemini 2.5 Flash ▶

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

## Output:

The screenshot shows the generated code output from the Gemini AI interface. It contains the same `sum_to_n` function definition and example usage as the input, but with some minor differences in variable names and logic flow.

```

def sum_to_n(n):
    if n < 1:
        return "Input must be a positive integer."
    total_sum = 0
    i = 1
    while i <= n:
        total_sum += i
        i += 1
    return total_sum

# Example usage:
print(f"Sum of first 5 natural numbers: {sum_to_n(5)}")
print(f"Sum of first 10 natural numbers: {sum_to_n(10)}")
print(f"Sum of first 1 natural numbers: {sum_to_n(1)}")
print(f"Sum of first 0 natural numbers: {sum_to_n(0)}")

... Sum of first 5 natural numbers: 15
Sum of first 10 natural numbers: 55
Sum of first 1 natural numbers: 1
Sum of first 0 natural numbers: Input must be a positive integer.

```

+ Code + Text

## Explanation:

- A function named `sum_to_n()` is defined to calculate the sum of natural numbers.

- The function takes an integer n as input.
- A variable (for example, total) is initialized to 0 to store the sum.
- A for loop runs from 1 to n.
- In each iteration, the current number is added to total.
- After the loop finishes, total contains the sum of the first n natural numbers.
- The function returns or prints the final sum.
- The output is displayed for the given sample input.

## Task-05:

Classes (Bank Account Class)

Scenario

You are designing a basic banking application.

Prompt:

E Create a Python class named BankAccount with attributes for account holder name and balance. Add methods deposit(), withdraw(), and check\_balance(). Show deposit and withdrawal operations with updated balance. Include comments in the code and explain how the class and methods work.

Code:

```

class BankAccount:
    def __init__(self, account_holder_name, initial_balance=0):
        self.account_holder_name = account_holder_name
        self.initial_balance = initial_balance
        self.balance = initial_balance
        if initial_balance < 0:
            print("Initial balance cannot be negative. Setting balance to 0.")
            self.balance = 0

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

        Args:
            amount (float): The amount to deposit.
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposit of ${amount:.2f} successful!")
        else:
            print("Deposit amount must be positive.")

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

        Args:
            amount (float): The amount to withdraw.
        """
        if amount > 0:
            if self.balance - amount >= 0:
                self.balance -= amount
                print(f"Withdrawal of ${amount:.2f} successful!")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        """Prints the current balance of the account.

        Returns:
            str: Account Holder: [self.account_holder_name], Current Balance: ${self.balance:.2f}"
        """
        print(f"\n--- Demonstrating the BankAccount class ---\n")
        print(f"1. --- Creating Account ---\n")
        my_account = BankAccount("John Doe", 1000.00)
        print(f"2. Perform a deposit operation\n")
        my_account.deposit(500.50)
        print(f"3. Perform a withdrawal operation\n")
        my_account.withdraw(200.75)
        print(f"4. Attempt to withdraw more than available funds\n")
        my_account.withdraw(1000.00)
        print(f"5. Attempt invalid operations (negative deposit/withdrawal)\n")
        my_account.deposit(-100)
        my_account.withdraw(-100)

    def __str__(self):
        return f"Account Holder: {self.account_holder_name}, Current Balance: ${self.balance:.2f}"

```

## Output:

```

my_account.withdraw(0)
...
--- Creating Account ---
Account Holder: John Doe, Current Balance: $1000.00

--- Deposit Operation ---
Deposit of $500.50 successful.
Account Holder: John Doe, Current Balance: $1500.50

--- Withdrawal Operation ---
Withdrawal of $200.75 successful.
Account Holder: John Doe, Current Balance: $1299.75

--- Attempting Over-withdrawal ---
Insufficient funds.
Account Holder: John Doe, Current Balance: $1299.75

--- Attempting Invalid Operations ---
Deposit amount must be positive.
Account Holder: John Doe, Current Balance: $1299.75
Withdrawal amount must be positive.
Account Holder: John Doe, Current Balance: $1299.75

```

## Explanation:

- A class named `BankAccount` is created to represent a bank account.
- The constructor (`__init__`) initializes the account holder's name and the starting balance.

- The deposit() method is used to add a given amount to the current balance.
- The withdraw() method subtracts an amount from the balance after checking if sufficient funds are available.
- If the balance is not enough, the withdrawal is not allowed and a message is shown.
- The check\_balance() method displays the current account balance.
- An object of the BankAccount class is created to perform banking operations.
- Deposit and withdrawal methods are called, and the updated balance is shown after each operation.