

Assignment 6.3 Ai Assisted Coding

Htno:2303a51OC1

Btno:06

Task 1: Classes (Student Class).

Prompt: Write a Python program to create a Student class.

Requirements:

- The class should have attributes: name, roll_number, and branch.
- Use a constructor (`__init__`) to initialize the values.
- Add a method `display_details()` that prints all student details clearly.
- Create at least one student object and call the method.
- Keep the code simple and beginner-friendly.

Input:

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". The left panel displays the following requirements:

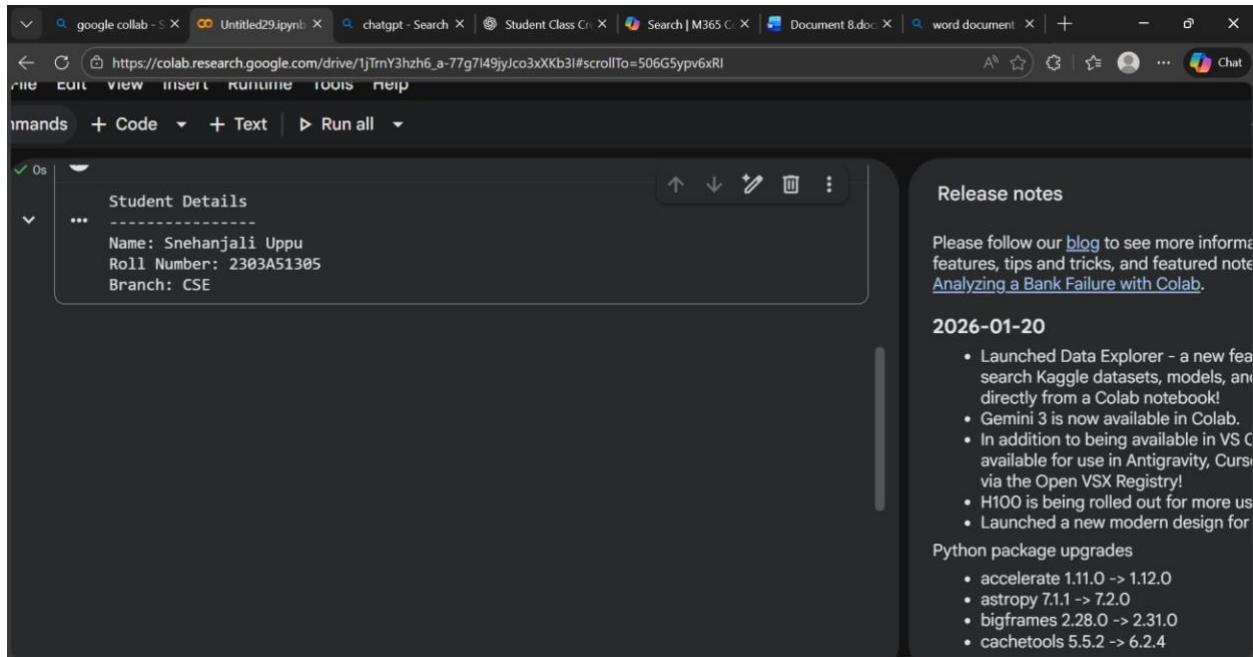
- Add a method `display_details()` that prints all student details clearly.
- Create at least one student object and call the method.
- Keep the code simple and beginner-friendly.

The right panel shows the generated Python code:

```
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"Student Name: {self.name}")  
        print(f"Roll Number: {self.roll_number}")  
        print(f"Branch: {self.branch}")  
  
# Create a student object  
student1 = Student("snehanjali", "101", "Computer Science")  
  
# Call the display_details method  
student1.display_details()
```

The right panel also includes a "Release notes" section with information about Data Explorer, Gemini 3, and Python package upgrades.

Output:



Explanation: The program defines a `Student` class with attributes (name, roll number, branch) initialized using a constructor.

The `display_details()` method prints the student information, showing basic OOP concepts like class, object, and methods.

Task 2: Loops of a number.

Prompt: Write a Python function that prints the first 10 multiples of a given number.

Requirements:

- Use a loop structure.
- The number should be passed as a parameter.
- Print the multiples clearly.
- Keep the code simple and beginner-friendly.

Then write another version of the same program using a different loop (while loop instead of for loop).

Input:

The screenshot shows a Google Colab notebook interface. At the top, there are several tabs: "google collab - 5", "Untitled29.ipynb", "chatgpt - Search", "Student Class Cr", "Search | M365 C", "Document 8.doc", "word document", and a "+" button. The main content area has a title "Task 2: Loops (Multiples of a Number)". Below it, a text box says "Here's the Python function using a `for` loop:". A code cell starts with `Os` and contains the following Python code:

```
Os
▶ def print_multiples_for_loop(number):
    print(f"Multiples of {number} (using for loop):")
    for i in range(1, 11): # Loop from 1 to 10 (inclusive)
        print(f"{number} x {i} = {number * i}")

# Example usage:
print_multiples_for_loop(5)

... Multiples of 5 (using for loop):
```

Output:

The screenshot shows the execution results of the Python code. The code cell output is:

```
print(f'{number} x {i} = {number * i}')

# Example usage:
print_multiples_for_loop(5)

... Multiples of 5 (using for loop):
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
```

2)Input:

And here's the Python function using a `while` loop:

```
[1]  Os  def print_multiples_while_loop(number):
    print(f"\nMultiples of {number} (using while loop):")
    i = 1
    while i <= 10:
        print(f"{number} x {i} = {number * i}")
        i += 1

    # Example usage:
    print_multiples_while_loop(7)

...
Multiples of 7 (using while loop):
7 x 1 = 7
7 x 2 = 14
```

2)Output:

```
[1]  Os  //doorteen_startprint()

Multiples of 7 (using while loop):
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
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Explanation: A loop is used to repeat an action 10 times and calculate multiples of a number using multiplication. `for` loop is best for fixed iterations, while `while` loop gives more control using a condition.

Task 3: Conditional Statements (Age Classification)

Prompt: Write a Python function to classify a person based on age.

Requirements:

- Use nested if-elif-else statements.

- Age groups:

Child: 0–12

Teenager: 13–19

Adult: 20–59

Senior: 60 and above

The function should take age as input and print the category. Then generate another version using an alternative approach (simplified conditions or dictionary-based logic). Keep the code beginner-friendly. Input: using if else statements

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". The code cell contains the following Python function:

```
def classify_person_if_else(age):
    """
    Classifies a person based on age using if-elif-else statements.
    """
    if not isinstance(age, (int, float)) or age < 0:
        print(f"Invalid age: {age}. Age must be a non-negative number.")
        return

    if age <= 12:
        print(f"Age {age}: Child")
    elif age <= 19:
        print(f"Age {age}: Teenager")
    elif age <= 59:
        print(f"Age {age}: Adult")
    else: # age >= 60
        print(f"Age {age}: Senior")

    # Example usage:
    print("--- Using if-elif-else ---")
    classify_person_if_else(5)      # Child
    classify_person_if_else(15)     # Teenager
    classify_person_if_else(30)     # Adult
```

The right sidebar displays "Release notes" for 2026-01-20, which include information about Data Explorer, Gemini 3, and Python package upgrades. The status bar at the bottom shows "10:01AM" and "Python 3".

Output:

The screenshot shows a Google Colab interface with a code cell containing Python code. The code defines a function `classify_person_if_elif_else` that prints age categories based on age values. It includes examples for valid ages (5, 15, 30, 70) and invalid ones (-5, "ten"). The output shows the printed results for each case.

```
print(f"Age {age}: Senior")  
# Example usage:  
print("--- Using if-elif-else ---")  
classify_person_if_elif_else(5) # Child  
classify_person_if_elif_else(15) # Teenager  
classify_person_if_elif_else(30) # Adult  
classify_person_if_elif_else(70) # Senior  
classify_person_if_elif_else(-5) # Invalid age  
classify_person_if_elif_else("ten") # Invalid age  
... --- Using if-elif-else ---  
Age 5: Child  
Age 15: Teenager  
Age 30: Adult  
Age 70: Senior  
Invalid age: -5. Age must be a non-negative number.  
Invalid age: ten. Age must be a non-negative number.
```

2) input: using alternative methods

The screenshot shows a Google Colab interface with a code cell containing Python code for an alternative classification method. The code defines a function `classify_person_alternative` that uses a list of age ranges to categorize an age. It includes validation for non-negative numbers and handles infinity for the upper bound of the last group. The sidebar on the right displays release notes and a list of Python package upgrades.

```
def classify_person_alternative(age):  
    """  
    Classifies a person based on age using an alternative approach  
    (list of age ranges).  
    """  
    if not isinstance(age, (int, float)) or age < 0:  
        print(f"Invalid age: {age}. Age must be a non-negative number.")  
        return  
  
    # Define age groups as (upper_bound, category_name)  
    age_groups = [  
        (12, "Child"),  
        (19, "Teenager"),  
        (59, "Adult"),  
        (float('inf'), "Senior") # Use infinity for the upper bound of the last g  
    ]  
  
    for upper_bound, category in age_groups:  
        if age <= upper_bound:  
            print(f"Age {age}: {category}")  
            return
```

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
- bigframes 2.28.0 -> 2.31.0
- cachetools 5.5.2 -> 6.2.4

Output:

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". In the code cell, the following Python code is run:

```
--- Using Alternative Approach (list of ranges) ---
Age 8: Child
Age 18: Teenager
Age 45: Adult
Age 85: Senior
Invalid age: -10. Age must be a non-negative number.
Invalid age: twenty. Age must be a non-negative number.
```

To the right of the code cell, there is a "Release notes" sidebar. At the top, it says "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](#)". Below this, the date "2026-01-20" is listed, followed by a bulleted list of changes:

- 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.

Under "Python package upgrades", there is a list of packages and their versions:

- accelerate 1.11.0 → 1.12.0
- astropy 7.1.1 → 7.2.0
- bigframes 2.28.0 → 2.31.0
- cachetools 5.5.2 → 6.2.4

At the bottom of the sidebar, it shows "10:01AM" and "Python 3".

Explanation: Conditional statements (`if-elif-else`) check age ranges and assign a category like child, teenager, adult, or senior.

The logic works top-to-bottom, and alternative methods like dictionary mapping make the system more flexible.

Task 4: For and While Loops (Sum of First n Numbers)

Prompt: Write a Python function `sum_to_n(n)` to calculate the sum of the first n natural numbers.

Requirements:

- Use a for loop.
- Keep the code simple.
- Show example function call and output.

Then provide an alternative implementation using:

- 1) a while loop
- 2) a mathematical formula

Explain the differences briefly.

1 Input:

The screenshot shows a Google Colab interface. On the left, a code cell contains Python code for calculating the sum of natural numbers using a for loop. On the right, a sidebar displays "Release notes" for January 2026, listing various updates like Data Explorer, Gemini 3, and Python package upgrades. The status bar at the bottom right shows "10:07 AM" and "Python 3".

```
def sum_to_n_for_loop(n):
    """
    Calculates the sum of the first n natural numbers using a for loop.
    """
    if not isinstance(n, int) or n < 1:
        print("Input must be a positive integer.")
        return None

    total_sum = 0
    for i in range(1, n + 1):
        total_sum += i
    return total_sum

# Example usage:
print(f"Sum of first 10 natural numbers (for loop): {sum_to_n_for_loop(10)}")
print(f"Sum of first 5 natural numbers (for loop): {sum_to_n_for_loop(5)}")
print(f"Sum of first 0 natural numbers (for loop): {sum_to_n_for_loop(0)}") # Invalid input
```

Output:

The screenshot shows the execution output of the Python code. The code runs successfully, printing the sum of natural numbers for n=10, n=5, and n=0. It also prints an error message for n=0. The status bar at the bottom right shows "10:07 AM" and "Python 3".

```
if not isinstance(n, int) or n < 1:
    print("Input must be a positive integer.")
    return None

total_sum = 0
for i in range(1, n + 1):
    total_sum += i
return total_sum

# Example usage:
print(f"Sum of first 10 natural numbers (for loop): {sum_to_n_for_loop(10)}")
print(f"Sum of first 5 natural numbers (for loop): {sum_to_n_for_loop(5)}")
print(f"Sum of first 0 natural numbers (for loop): {sum_to_n_for_loop(0)}") # Invalid input

...
Sum of first 10 natural numbers (for loop): 55
Sum of first 5 natural numbers (for loop): 15
Input must be a positive integer.
Sum of first 0 natural numbers (for loop): None
```

2) input:

Next, an alternative implementation using a `while` loop:

```
[8]  Untitled29.ipynb
def sum_to_n_while_loop(n):
    """
    Calculates the sum of the first n natural numbers using a while loop.
    """
    if not isinstance(n, int) or n < 1:
        print("Input must be a positive integer.")
        return None

    total_sum = 0
    counter = 1
    while counter <= n:
        total_sum += counter
        counter += 1
    return total_sum

# Example usage:
print(f"Sum of first 10 natural numbers (while loop): {sum_to_n_while_loop(10)}")
print(f"Sum of first 5 natural numbers (while loop): {sum_to_n_while_loop(5)}")
```

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 1.11.0 → 1.12.0
- astropy 7.1.1 → 7.2.0
- bigframes 2.28.0 → 2.31.0
- cachetools 5.5.2 → 6.2.4

Variables Terminal ✓ 10:07 AM Python 3

Output:

```
def sum_to_n_while_loop(n):
    """
    Calculates the sum of the first n natural numbers using a while loop.
    """
    if not isinstance(n, int) or n < 1:
        print("Input must be a positive integer.")
        return None

    total_sum = 0
    counter = 1
    while counter <= n:
        total_sum += counter
        counter += 1
    return total_sum

# Example usage:
print(f"Sum of first 10 natural numbers (while loop): {sum_to_n_while_loop(10)}")
print(f"Sum of first 5 natural numbers (while loop): {sum_to_n_while_loop(5)}")
```

... Sum of first 10 natural numbers (while loop): 55
Sum of first 5 natural numbers (while loop): 15

3)input:

Finally, the most efficient approach using a mathematical formula (Gauss's formula):

```
[9] 0s
def sum_to_n_formula(n):
    """
    Calculates the sum of the first n natural numbers using a mathematical formula
    """
    if not isinstance(n, int) or n < 1:
        print("Input must be a positive integer.")
        return None

    return n * (n + 1) // 2 # Using integer division

# Example usage:
print(f"Sum of first 10 natural numbers (formula): {sum_to_n_formula(10)}")
print(f"Sum of first 5 natural numbers (formula): {sum_to_n_formula(5)}")
```

... Sum of first 10 natural numbers (formula): 55

Release notes

Please follow our [blo](#) features, tips and tri [Analyzing a Bank Fail](#)

2026-01-20

- Launched Data search Kaggle directly from a
- Gemini 3 is now
- In addition to b available for us via the Open V
- H100 is being i
- Launched a ne

Python package upg!

- accelerate 1.11.
- astropy 7.1.1 ->

Output:

```
... Sum of first 10 natural numbers (formula): 55
... Sum of first 5 natural numbers (formula): 15
```

Release notes

Please follow our [blo](#) features, tips and tri [Analyzing a Bank Fail](#)

2026-01-20

- Launched Data search Kaggle directly from a
- Gemini 3 is now
- In addition to b available for us via the Open V
- H100 is being i
- Launched a ne

Python package upg!

- accelerate 1.11.
- astropy 7.1.1 ->

Explanation:

- In the **for loop**, the program repeats from 1 to n and keeps adding numbers to total.
- In the **while loop**, the same addition happens but the loop runs based on a condition ($i \leq n$).

Task 5: Classes (Bank Account Class)

Prompt: Write a Python program to create a BankAccount class.

Requirements:

- Attributes: account_holder, balance
- Methods: deposit(amount) – add money withdraw(amount) – deduct money if sufficient balance check_balance() – display current balance
- Include comments in the code.
- Create an object and demonstrate deposit and withdrawal.
- Keep it simple and beginner-friendly.

Input:

The screenshot shows a Google Colab notebook titled "Untitled29.ipynb". The code cell contains the following Python code:

```
class BankAccount:  
    # Constructor to initialize account_holder and balance  
    def __init__(self, account_holder, initial_balance=0):  
        self.account_holder = account_holder  
        self.balance = initial_balance  
        print(f"Account created for {self.account_holder} with initial balance ${self.balance:.2f}")  
  
    # Method to deposit money into the account  
    def deposit(self, amount):  
        if amount > 0:  
            self.balance += amount  
            print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")  
        else:  
            print("Deposit amount must be positive.")  
  
    # Method to withdraw money from the account  
    def withdraw(self, amount):  
        if amount <= 0:  
            print("Withdrawal amount must be positive.")  
        elif amount > self.balance:  
            print("Insufficient balance. Current balance: ${self.balance:.2f}")  
        else:  
            self.balance -= amount  
            print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
```

The right sidebar displays the "Release notes" for 2026-01-20, which include information about Data Explorer, Gemini 3, and Python package upgrades.

```
else:
    self.balance -= amount
    print(f"Withdraw ${amount:.2f}. New balance: ${self.balance:.2f}")

# Method to display the current balance
def check_balance(self):
    print(f"Account holder: {self.account_holder}, Current balance: ${self.ba

# --- Demonstrate the BankAccount class ---

# Create a new bank account object
my_account = BankAccount("John Doe", 100.00)

# Check initial balance
my_account.check_balance()

# Deposit some money
my_account.deposit(50.50)

# Withdraw some money
my_account.withdraw(20.00)
```

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 1.11.0 -> 1.12.0
- astropy 7.1.1 -> 7.2.0
- bigframes 2.28.0 -> 2.31.0
- cachetools 5.5.2 -> 6.2.4

Output:

```
Account created for John Doe with initial balance $100.00
Account holder: John Doe, Current balance: $100.00
...
Deposited $50.50. New balance: $150.50
Withdraw $20.00. New balance: $130.50
Insufficient balance. Current balance: $130.50
Account holder: John Doe, Current balance: $130.50
Deposit amount must be positive.
Withdrawal amount must be positive.
```

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 1.11.0 -> 1.12.0
- astropy 7.1.1 -> 7.2.0
- bigframes 2.28.0 -> 2.31.0
- cachetools 5.5.2 -> 6.2.4

Explanation:

- The **class** represents a bank account.
- The **constructor (`__init__`)** sets account holder name and initial balance.
- deposit()** increases balance.
- withdraw()** checks balance before deducting (prevents overdraft).
- check_balance()** shows account details.

