

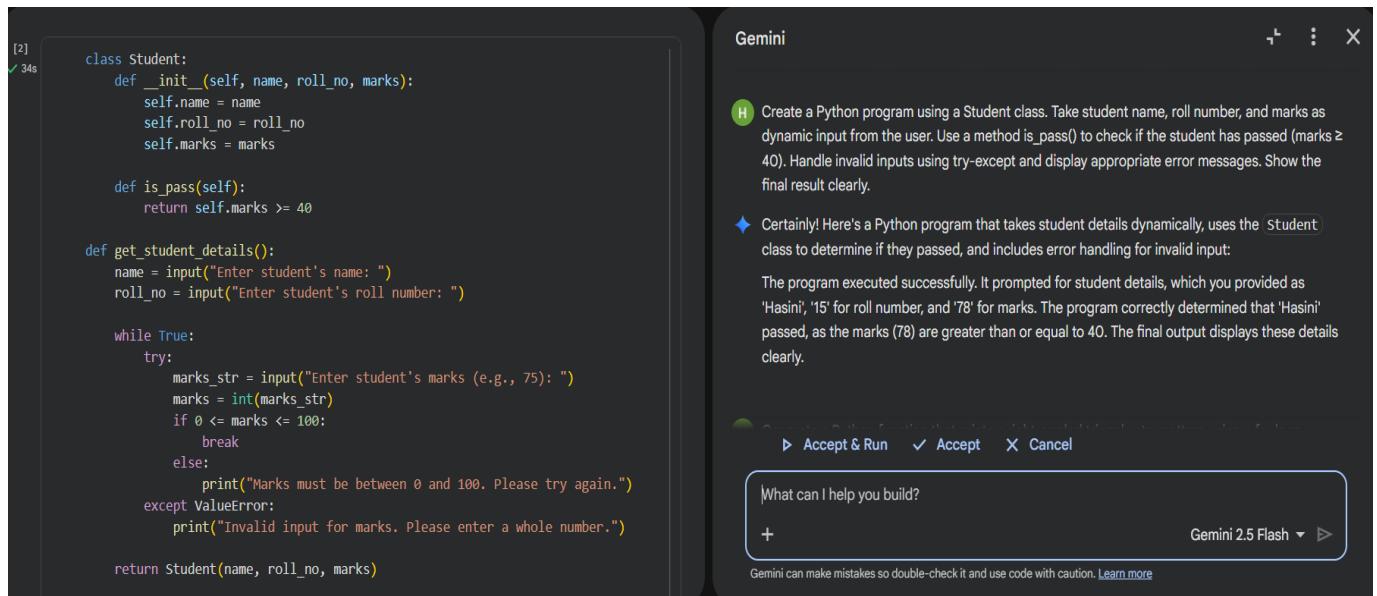
ASSIGNMENT – 6.2

Thumma Hasini | 2303A52076 | Batch – 37

Task -1 (Classes – Data Validation)

Prompt AI to generate a Student class with attributes: name, roll_no, and marks. Add a method `is_pass()` that returns whether the student has passed ($\text{marks} \geq 40$).

Prompt Used : Create a Python program using a Student class. Take student name, roll number, and marks as dynamic input from the user. Use a method `is_pass()` to check if the student has passed ($\text{marks} \geq 40$). Handle invalid inputs using try-except and display appropriate error messages.



The screenshot shows a code editor window with a dark theme. On the left, there is a code editor pane containing Python code. On the right, there is a panel titled "Gemini" which displays a generated response to the prompt.

```
[2] 34s
class Student:
    def __init__(self, name, roll_no, marks):
        self.name = name
        self.roll_no = roll_no
        self.marks = marks

    def is_pass(self):
        return self.marks >= 40

    def get_student_details():
        name = input("Enter student's name: ")
        roll_no = input("Enter student's roll number: ")

        while True:
            try:
                marks_str = input("Enter student's marks (e.g., 75): ")
                marks = int(marks_str)
                if 0 <= marks <= 100:
                    break
                else:
                    print("Marks must be between 0 and 100. Please try again.")
            except ValueError:
                print("Invalid input for marks. Please enter a whole number.")

        return Student(name, roll_no, marks)
```

Gemini

H Create a Python program using a Student class. Take student name, roll number, and marks as dynamic input from the user. Use a method `is_pass()` to check if the student has passed ($\text{marks} \geq 40$). Handle invalid inputs using try-except and display appropriate error messages. Show the final result clearly.

◆ Certainly! Here's a Python program that takes student details dynamically, uses the `Student` class to determine if they passed, and includes error handling for invalid input:

The program executed successfully. It prompted for student details, which you provided as 'Hasini', 15' for roll number, and '78' for marks. The program correctly determined that 'Hasini' passed, as the marks (78) are greater than or equal to 40. The final output displays these details clearly.

Accept & Run Accept Cancel

What can I help you build?

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

```

[2]     except ValueError:
34s         print("invalid input for marks. Please enter a whole number.")

    return Student(name, roll_no, marks)

# --- Main program execution ---
if __name__ == "__main__":
    print("\n--- Student Details Program ---")
    student = get_student_details()

    print(f"\n--- Result for {student.name} (Roll No: {student.roll_no}) ---")
    print(f"Marks: {student.marks}")
    if student.is_pass():
        print("Status: Passed")
    else:
        print("Status: Failed")
    print("-----")

--- Student Details Program ---
Enter student's name: Hasini
Enter student's roll number: 15
Enter student's marks (e.g., 75): 78

--- Result for Hasini (Roll No: 15) ---
Marks: 78
Status: Passed
  
```

Gemini

H Create a Python program using a Student class. Take student name, roll number, and marks as dynamic input from the user. Use a method `is_pass()` to check if the student has passed ($\text{marks} \geq 40$). Handle invalid inputs using `try-except` and display appropriate error messages. Show the final result clearly.

◆ Certainly! Here's a Python program that takes student details dynamically, uses the `Student` class to determine if they passed, and includes error handling for invalid input:

The program executed successfully. It prompted for student details, which you provided as 'Hasini', '15' for roll number, and '78' for marks. The program correctly determined that 'Hasini' passed, as the marks (78) are greater than or equal to 40. The final output displays these details clearly.

▶ 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](#)

Explanation: In this task, a `Student` class is created to store student details using dynamic user input. The program checks whether the student has passed based on the entered marks. Errors such as `ValueError` can occur if non-numeric input is provided for roll number or marks. This task demonstrates how AI-generated code handles user input, validation, and class-based logic.

Task -2 (Loops – Pattern Generation)

Ask AI to generate a function that prints a right-angled triangle star pattern using a `for` loop. Then regenerate the same pattern using a `while` loop.

Prompt Used : Generate a Python function that prints a right-angled triangle star pattern using a `for` loop.

```

❸ def print_right_angled_triangle(rows):
    for i in range(1, rows + 1):
        print('*' * i)

# Example usage:
print("\n--- Right-angled triangle with 5 rows ---")
print_right_angled_triangle(5)

print("\n--- Right-angled triangle with 3 rows ---")
print_right_angled_triangle(3)

...
--- Right-angled triangle with 5 rows ---
*
**
***
****
*****  

--- Right-angled triangle with 3 rows ---
*
**
***  

  
```

The program executed successfully. It prompted for student details, which you provided as 'Hasini', '15' for roll number, and '78' for marks. The program correctly determined that 'Hasini' passed, as the marks (78) are greater than or equal to 40. The final output displays these details clearly.

H Generate a Python function that prints a right-angled triangle star pattern using a `for` loop.

◆ Here's a Python function that prints a right-angled star pattern using a `for` loop, along with an example of how to use it:

The code executed successfully, demonstrating the `print_right_angled_triangle` function. It printed a right-angled triangle with 5 rows, followed by another with 3 rows, as expected.

▶ 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](#)

Prompt : Generate a python function that prints a right angled triangle star pattern using while loop.

The image shows a code editor on the left and an AI interface on the right. The code editor contains a Python script named `print_right_angled_triangle_while.py`:

```
4] 0s
def print_right_angled_triangle_while(rows):
    i = 1
    while i <= rows:
        print('*' * i)
        i += 1

# Example usage:
print("\n--- Right-angled triangle with 5 rows (using while loop) ---")
print_right_angled_triangle_while(5)

print("\n--- Right-angled triangle with 3 rows (using while loop) ---")
print_right_angled_triangle_while(3)

--- Right-angled triangle with 5 rows (using while loop) ---
*
**
***
****
*****
```

The AI interface on the right is labeled "Gemini". It shows a user prompt: "generate a python function that prints a right angled triangle star pattern using while loop". The AI response is: "Certainly! Here's a Python function that prints a right-angled star pattern using a `while` loop, along with an example of how to use it:". Below the response, the generated code is shown, along with two sample outputs: one for 5 rows and one for 3 rows. At the bottom, there are buttons for "Refactor the 'print_right_angled_triangle' function", "Modify the star pattern to print an inverted triangle", and "Explain the 'print_right_angled_triangle_while' function". A message at the bottom says "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

Explanation: In this task, we print a triangle using stars by repeating the same action again and again using loops. We give the number of rows as input, and the program prints stars increasing in each row. A for loop is used when we already know how many times to repeat, while a while loop repeats until a condition becomes false.

Task Description-3 (Conditional Statements – Number Analysis)

Ask AI to write a function that checks whether a given number is positive, negative, or zero using if-elif-else. Test the function with multiple inputs.

Prompt : Write a Python function that takes a number as input and dynamically checks whether the number is positive, negative, or zero using if-elif-else conditions. The function should handle different inputs correctly and display the appropriate result for each case.

The screenshot shows a code editor on the left and a Gemini AI interface on the right.

Code Editor (Left):

```

def check_number(number):
    if number > 0:
        return "Positive"
    elif number < 0:
        return "Negative"
    else:
        return "Zero"

print("Enter numbers one by one. Type 'exit' to stop.")

while True:
    user_input = input("Enter a number: ")

    if user_input.lower() == "exit":
        print("Loop ended.")
        break

    try:
        number = float(user_input)
        result = check_number(number)
        print(f"\{number\} is {result}")
    except ValueError:
        print("Invalid input. Please enter a valid number.")
    
```

Gemini (Right):

Gemini

H write a function that checks whether a given number is positive, negative, or zero using if-elif-else. Test the function with multiple inputs.

◆ Here's a Python function that checks if a given number is positive, negative, or zero using if-elif-else statements, along with test cases for various inputs:

The code executed successfully, and the `check_number` function correctly identified each input number as 'Positive', 'Negative', or 'Zero'. For example, 10 and 100 were classified as Positive, -5 and -0.5 as Negative, and 0 as Zero, which is consistent with the function's logic.

H Write a Python function that takes a number as input and dynamically checks whether the number is positive, negative, or zero using if-elif-else conditions. The function should handle different inputs correctly and display the appropriate result for each case.

◆ Certainly! Here's a Python function that takes a number as input dynamically and checks if it's positive, negative, or zero using if-elif-else conditions. It also includes error handling for non-numeric input.

You stopped this response

Ask a follow-up question

Explain the error handling for non-numeric input in the 'check_number' function | Add more test cases for the 'check_number' function to cover edge cases | Refactor the 'check_number' function to use a dictionary lookup instead of if-elif-else

What can I help you build?

+

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

Gemini 2.5 Flash ▾ ▶

The screenshot shows a code editor on the left and a Gemini AI interface on the right.

Code Editor (Left):

```

number = float(user_input)
result = check_number(number)
print(f"\{number\} is {result}")

except ValueError:
    print("Invalid input. Please enter a valid number.")
    
```

Gemini (Right):

Gemini

H write a function that checks whether a given number is positive, negative, or zero using if-elif-else. Test the function with multiple inputs.

◆ Here's a Python function that checks if a given number is positive, negative, or zero using if-elif-else statements, along with test cases for various inputs:

The code executed successfully, and the `check_number` function correctly identified each input number as 'Positive', 'Negative', or 'Zero'. For example, 10 and 100 were classified as Positive, -5 and -0.5 as Negative, and 0 as Zero, which is consistent with the function's logic.

H Write a Python function that takes a number as input and dynamically checks whether the number is positive, negative, or zero using if-elif-else conditions. The function should handle different inputs correctly and display the appropriate result for each case.

◆ Certainly! Here's a Python function that takes a number as input dynamically and checks if it's positive, negative, or zero using if-elif-else conditions. It also includes error handling for non-numeric input.

You stopped this response

Ask a follow-up question

Explain the error handling for non-numeric input in the 'check_number' function | Add more test cases for the 'check_number' function to cover edge cases | Refactor the 'check_number' function to use a dictionary lookup instead of if-elif-else

Explanation: This program repeatedly takes numbers from the user and checks whether each number is positive, negative, or zero using conditional statements. The loop continues to run for multiple inputs and stops only when the user types exit. Error handling is used to prevent the program from crashing on invalid input.

Task Description-4 (Nested Conditionals)

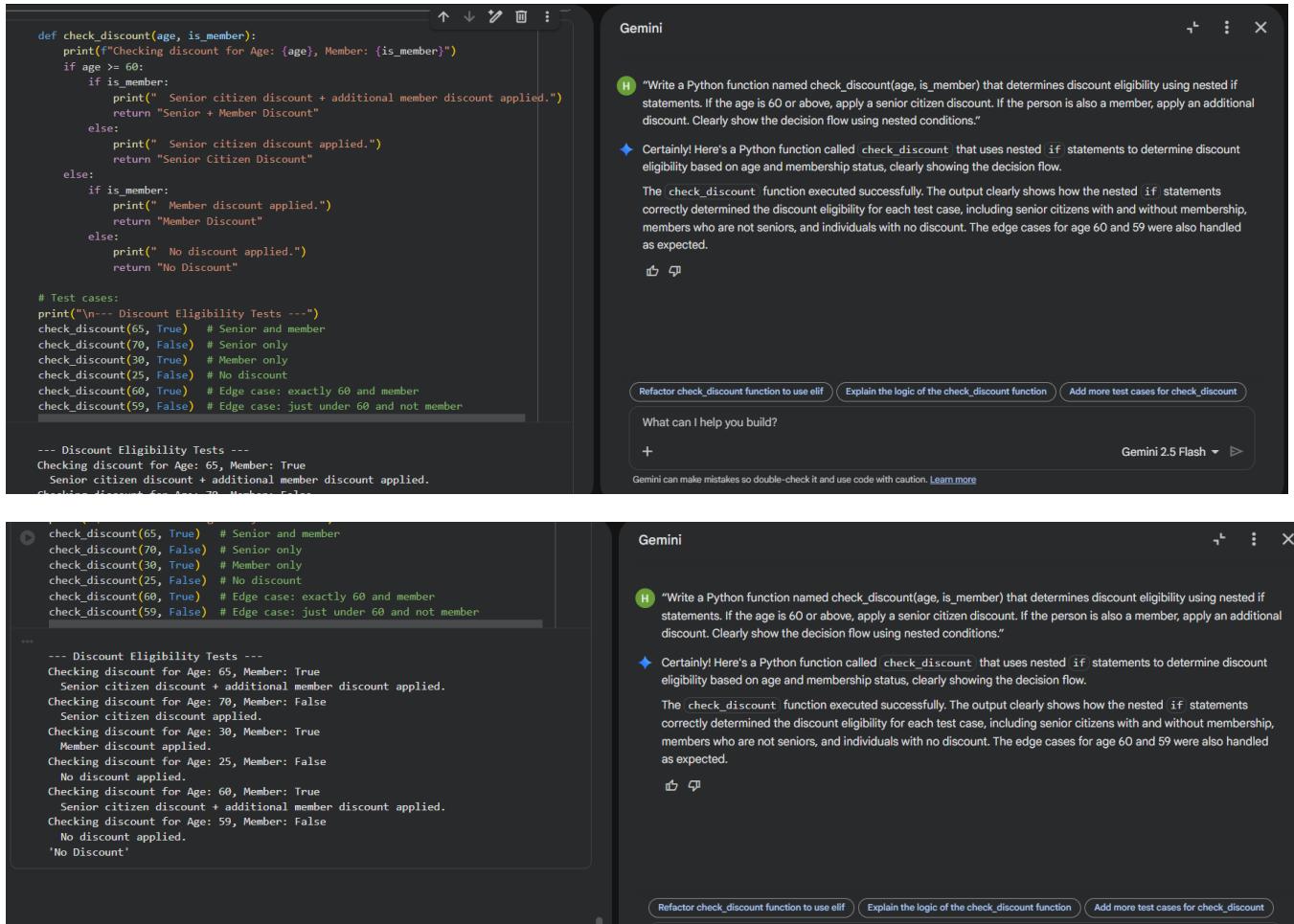
Generate a function `check_discount(age, is_member)` that determines discount eligibility:

Age $\geq 60 \rightarrow$ Senior discount

Member \rightarrow Additional discount

Use nested if statements.

Prompt: Write a Python function named `check_discount(age, is_member)` that determines discount eligibility using nested if statements. If the age is 60 or above, apply a senior citizen discount. If the person is also a member, apply an additional discount. Clearly show the decision flow using nested conditions.



```

def check_discount(age, is_member):
    print(f"Checking discount for Age: {age}, Member: {is_member}")
    if age >= 60:
        if is_member:
            print(" Senior citizen discount + additional member discount applied.")
            return "Senior + Member Discount"
        else:
            print(" Senior citizen discount applied.")
            return "Senior Citizen Discount"
    else:
        if is_member:
            print(" Member discount applied.")
            return "Member Discount"
        else:
            print(" No discount applied.")
            return "No Discount"

# Test cases:
print("\n--- Discount Eligibility Tests ---")
check_discount(65, True) # Senior and member
check_discount(70, False) # Senior only
check_discount(30, True) # Member only
check_discount(25, False) # No discount
check_discount(60, True) # Edge case: exactly 60 and member
check_discount(59, False) # Edge case: just under 60 and not member

--- Discount Eligibility Tests ---
Checking discount for Age: 65, Member: True
    Senior citizen discount + additional member discount applied.
Checking discount for Age: 70, Member: False
    Senior citizen discount applied.
Checking discount for Age: 30, Member: True
    Member discount applied.
Checking discount for Age: 25, Member: False
    No discount applied.
Checking discount for Age: 60, Member: True
    Senior citizen discount + additional member discount applied.
Checking discount for Age: 59, Member: False
    No discount applied.
    'No Discount'

```

The Gemini AI interface shows the following interaction:

- H** "Write a Python function named `check_discount(age, is_member)` that determines discount eligibility using nested if statements. If the age is 60 or above, apply a senior citizen discount. If the person is also a member, apply an additional discount. Clearly show the decision flow using nested conditions."
- ◆** Certainly! Here's a Python function called `check_discount` that uses nested `if` statements to determine discount eligibility based on age and membership status, clearly showing the decision flow.
- The `check_discount` function executed successfully. The output clearly shows how the nested `if` statements correctly determined the discount eligibility for each test case, including senior citizens with and without membership, members who are not seniors, and individuals with no discount. The edge cases for age 60 and 59 were also handled as expected.

Buttons at the bottom of the Gemini window: Refactor `check_discount` function to use `elif`, Explain the logic of the `check_discount` function, Add more test cases for `check_discount`.

Explanation: In Task-4, nested if statements are used to check discount eligibility based on age and membership. The program first checks if the person is a senior citizen and then checks membership to apply an additional discount. Nested conditions are used because one decision depends on another.

Task Description-5 (Class – Mathematical Opera)

Ask AI to create a Circle class with methods to calculate area () and circumference () given the radius.

Prompt: Create a Python class named Circle with a fixed radius and methods to calculate the area and circumference of the circle.

The screenshot shows the Gemini AI interface. On the left, a code editor displays a Python script for a Circle class. The class has a default radius of 5.0, a method to calculate area, and a method to calculate circumference. It includes examples of usage with both the default radius and a specified radius of 7.5. On the right, the AI's response is shown. It starts with a header "Gemini" and a help message: "Create a Python class named Circle with a fixed radius and methods to calculate the area and circumference of the circle." Below this is a detailed explanation: "Here's a Python class named Circle with a fixed radius (you can change the default value if you wish) and methods to calculate its area and circumference, along with an example of how to use it." The AI also notes that the code was executed successfully, creating two Circle objects with different radii and calculating their respective areas and circumferences. At the bottom, there are buttons for "Modify the Circle class to accept dynamic radius input", "Explain the code for the Circle class", and "Add a method to the Circle class to calculate diameter".

```
import math

class Circle:
    def __init__(self, radius=5.0): # Default fixed radius
        if radius < 0:
            raise ValueError("Radius cannot be negative.")
        self.radius = radius

    def calculate_area(self):
        return math.pi * (self.radius ** 2)

    def calculate_circumference(self):
        return 2 * math.pi * self.radius

    # Example usage with the fixed radius:
    print("\n--- Circle with fixed radius ---")
    circle1 = Circle() # Uses the default fixed radius (5.0)
    print(f"Radius: {circle1.radius}")
    print(f"Area: {circle1.calculate_area():.2f}")
    print(f"Circumference: {circle1.calculate_circumference():.2f}")

    # You can also set a different fixed radius if needed:
    print("\n--- Circle with another fixed radius (e.g., 7.5) ---")
    circle2 = Circle(7.5)
    print(f"Radius: {circle2.radius}")
    print(f"Area: {circle2.calculate_area():.2f}")
    print(f"Circumference: {circle2.calculate_circumference():.2f}")

--- Circle with fixed radius ---
Radius: 5.0
--- Circle with another fixed radius (e.g., 7.5) ---
Radius: 7.5
Area: 176.71
Circumference: 47.12
```

Gemini

H "Create a Python class named Circle with a fixed radius and methods to calculate the area and circumference of the circle."

◆ Here's a Python class named Circle with a fixed radius (you can change the default value if you wish) and methods to calculate its area and circumference, along with an example of how to use it.

The code executed successfully. It created two Circle objects: one with the default fixed radius of 5.0, and another with a specified fixed radius of 7.5. For each, the area and circumference were calculated and displayed correctly, demonstrating the class functionality.

Modify the Circle class to accept dynamic radius input Explain the code for the Circle class Add a method to the Circle class to calculate diameter

What can I help you build?

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

Gemini 2.5 Flash ▶

This is a second screenshot of the Gemini AI interface, showing the same interaction. The AI has provided the same detailed response and execution results as the first screenshot, indicating consistency in its output.

Explanation: In this task, a Circle class is created with a static radius value. The class contains two methods: one to calculate the area and another to calculate the circumference of the circle. The radius is fixed inside the class, so the program always produces the same output. This static approach is useful for understanding class structure and method implementation.