AIAC-Assignment-6.3

2303A51680

K Rishitha

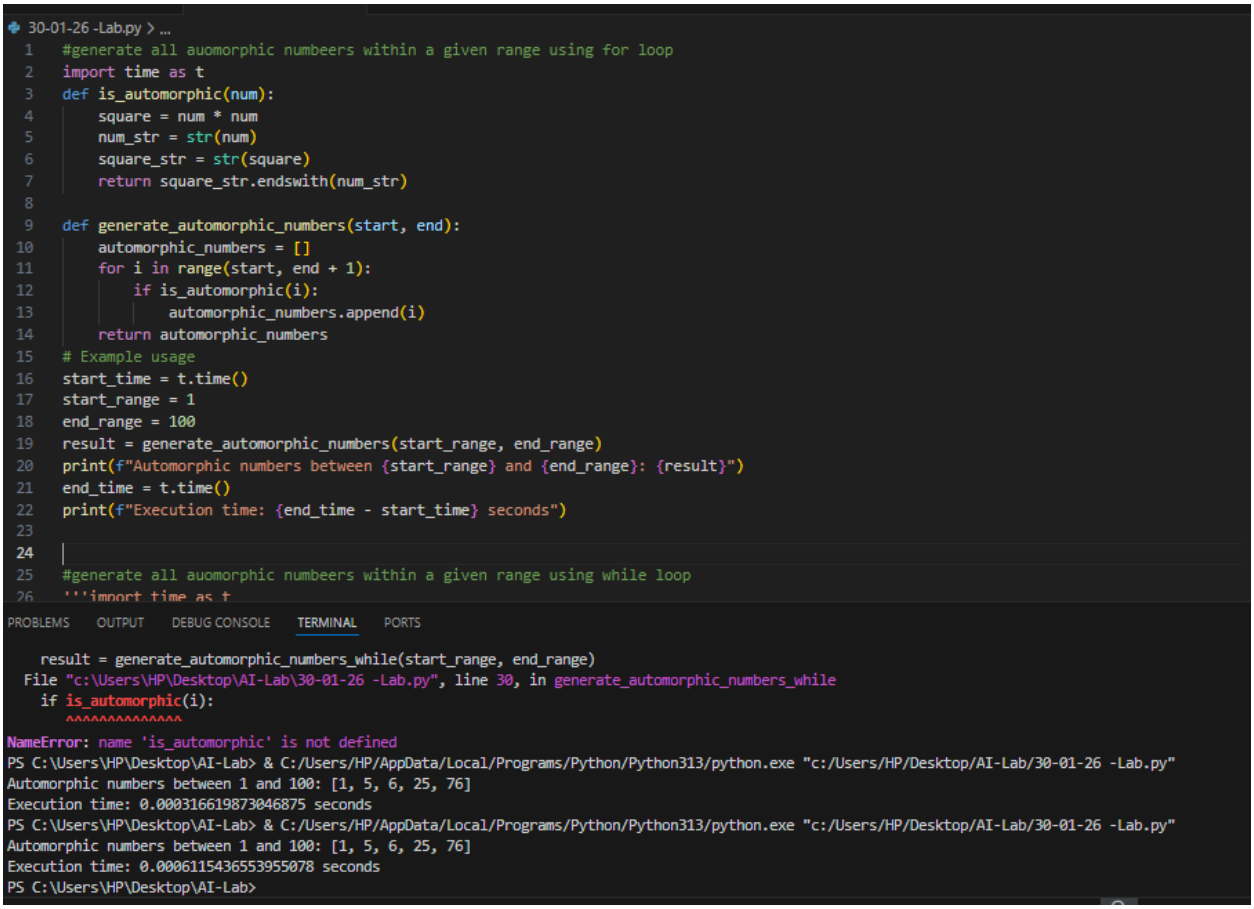
Batch-23

Task-1:

Task Description #1 (Loops – Automorphic Numbers in a Range)
• Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
• Instructions:
o Get AI-generated code to list Automorphic numbers using a for loop.

o Analyze the correctness and efficiency of the generated logic.
o Ask AI to regenerate using a while loop and compare both implementations.
Expected Output #1:
• Correct implementation that lists Automorphic numbers using both loop types, with explanation

Using for loop:

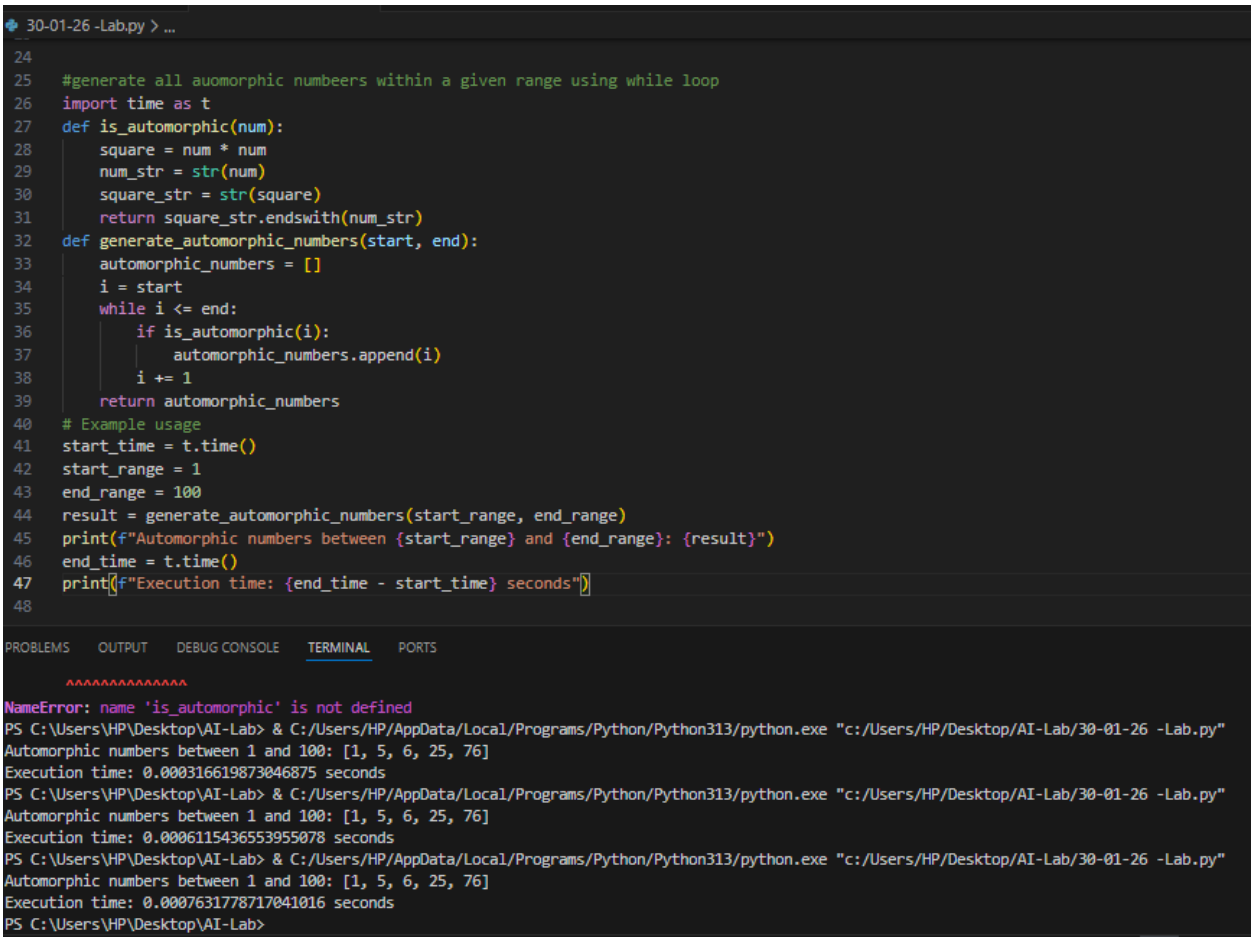


```
30-01-26 -Lab.py > ...
1   #generate all auomorphic numbeers within a given range using for loop
2   import time as t
3   def is_automorphic(num):
4       square = num * num
5       num_str = str(num)
6       square_str = str(square)
7       return square_str.endswith(num_str)
8
9   def generate_automorphic_numbers(start, end):
10      automorphic_numbers = []
11      for i in range(start, end + 1):
12          if is_automorphic(i):
13              automorphic_numbers.append(i)
14      return automorphic_numbers
15  # Example usage
16  start_time = t.time()
17  start_range = 1
18  end_range = 100
19  result = generate_automorphic_numbers(start_range, end_range)
20  print(f"Automorphic numbers between {start_range} and {end_range}: {result}")
21  end_time = t.time()
22  print(f"Execution time: {end_time - start_time} seconds")
23
24
25  #generate all auomorphic numbeers within a given range using while loop
26  '''import time as t
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
    result = generate_automorphic_numbers_while(start_range, end_range)
  File "c:\Users\HP\Desktop\AI-Lab\30-01-26 -Lab.py", line 30, in generate_automorphic_numbers_while
    if is_automorphic(i):
       ^^^^^^^^^^^^^^
NameError: name 'is_automorphic' is not defined
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/Desktop/AI-Lab/30-01-26 -Lab.py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.000316619873046875 seconds
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/Desktop/AI-Lab/30-01-26 -Lab.py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.0006115436553955078 seconds
PS C:\Users\HP\Desktop\AI-Lab>
```

Using While loop:



```
30-01-26 -Lab.py > ...
24
25  #generate all auomorphic numbeers within a given range using while loop
26  import time as t
27  def is_automorphic(num):
28      square = num * num
29      num_str = str(num)
30      square_str = str(square)
31      return square_str.endswith(num_str)
32  def generate_automorphic_numbers(start, end):
33      automorphic_numbers = []
34      i = start
35      while i <= end:
36          if is_automorphic(i):
37              automorphic_numbers.append(i)
38          i += 1
39      return automorphic_numbers
40  # Example usage
41  start_time = t.time()
42  start_range = 1
43  end_range = 100
44  result = generate_automorphic_numbers(start_range, end_range)
45  print(f"Automorphic numbers between {start_range} and {end_range}: {result}")
46  end_time = t.time()
47  print(f"Execution time: {end_time - start_time} seconds")
48

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

      ^^^^^^^^^^^^^
NameError: name 'is_automorphic' is not defined
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/Desktop/AI-Lab/30-01-26 -Lab.py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.000316619873046875 seconds
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/Desktop/AI-Lab/30-01-26 -Lab.py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.0006115436553955078 seconds
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/Desktop/AI-Lab/30-01-26 -Lab.py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.0007631778717041016 seconds
PS C:\Users\HP\Desktop\AI-Lab>
```

Task-2:

Conditional Statements – Online Shopping Feedback Classification

• Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).

• Instructions:

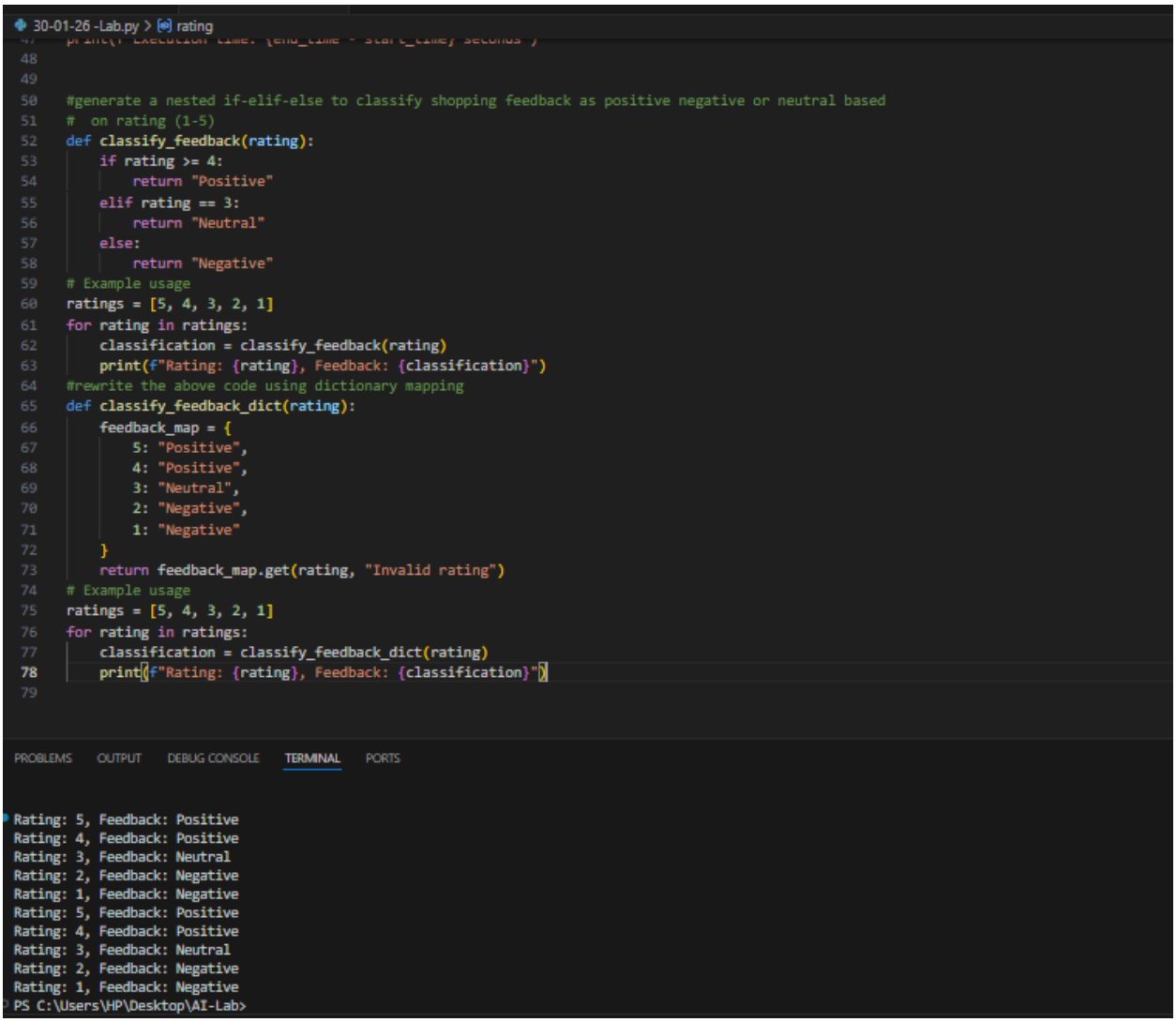
o Generate initial code using nested if-elif-else.

o Analyze correctness and readability.

o Ask AI to rewrite using dictionary-based or match-case structure.

Expected Output #2:

• Feedback classification function with explanation and an alternative approach.



Task-3:

Statistical_operations

Define a function named statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers:

• Minimum, Maximum

• Mean, Median, Mode

• Variance, Standard Deviation

While writing the function, observe the code suggestions provided by GitHub

Copilot.Make decisions to accept, reject, or modify the suggestions based on their relevance and correctness

```python
#generate a function named statistical_operations that takes a list of numbers as input and calculate minimum, maximum, mean, median,
# and standard deviation of the list.
import time as t
import time as t
def is_automorphic(num):
    square = num * num
    num_str = str(num)
    square_str = str(square)
    return square_str.endswith(num_str)
def generate_automorphic_numbers(start, end):
    automorphic_numbers = []
    for num in range(start, end + 1):
        if is_automorphic(num):
            automorphic_numbers.append(num)
    return automorphic_numbers
start_time = t.time()
start_range = 1
end_range = 1000
result = generate_automorphic_numbers(start_range, end_range)
end_time = t.time()
print(f"Automorphic numbers between {start_range} and {end_range}: {result}")
print(f"Execution time: {end_time - start_time} seconds")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/AI-ass.py
Automorphic numbers between 1 and 1000: [1, 5, 6, 25, 76, 376, 625]
Execution time: 0.00022482872009277344 seconds
PS C:\Users\HP\Desktop\AI-Lab>
```

Task 4:

Teacher Profile

• Prompt: Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.

• Expected Output: Class with initializer, method, and object creation.

```python
#Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.
class Teacher:
    def __init__(self, teacher_id, name, subject, experience):
        self.teacher_id = teacher_id
        self.name = name
        self.subject = subject
        self.experience = experience

    def display_details(self):
        print(f"Teacher ID: {self.teacher_id}")
        print(f"Name: {self.name}")
        print(f"Subject: {self.subject}")
        print(f"Experience: {self.experience} years")
# Example usage
teacher = Teacher(1, "Alice Smith", "Mathematics", 10)
teacher.display_details()
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/AI-ass.py
Teacher ID: 1
Name: Alice Smith
Subject: Mathematics
Experience: 10 years
PS C:\Users\HP\Desktop\AI-Lab>
```

Task-5:

Zero-Shot Prompting with Conditional Validation

Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.

Requirements

• The function must ensure the mobile number:

o Starts with 6, 7, 8, or 9

o Contains exactly 10 digits

Expected Output

• A valid Python function that performs all required validations without using any input-output examples in the prompt.

```python
#generate a python code that validates an Indian mobile number.starts with 6,7,9 or 9
import re
def validate_indian_mobile_number(mobile_number):
    pattern = r'^[6-9]\d{9}$'
    if re.match(pattern, mobile_number):
        return True
    else:
        return False
# Example usage
mobile_number = input("Enter an Indian mobile number: ")
if validate_indian_mobile_number(mobile_number):
    print("Valid Indian mobile number.")
else:
    print("Invalid Indian mobile number.")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/AI-ass.py
Enter an Indian mobile number: 7093374718
Valid Indian mobile number.
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/AI-ass.py
Enter an Indian mobile number: 9876543210
Valid Indian mobile number.
PS C:\Users\HP\Desktop\AI-Lab>
```

Task -6:

Loops – Armstrong Numbers in a Range
Task: Write a function using AI that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).
Instructions:
• Use a for loop and digit power logic.
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).
• Ask AI to regenerate an optimized version (using list comprehensions).
Expected Output #7:
• Python program listing Armstrong numbers in the range.
• Optimized version with explanation

```python
#Write a pyhton code that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).using for loop and digit power log
def is_armstrong_number(num):
    num_str = str(num)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == num
def armstrong_numbers_in_range(start, end):
    armstrong_numbers = []
    for num in range(start, end + 1):
        if is_armstrong_number(num):
            armstrong_numbers.append(num)
    return armstrong_numbers
start_range = 1
end_range = 1000
result = armstrong_numbers_in_range(start_range, end_range)
print(f"Armstrong numbers between {start_range} and {end_range}: {result}")
```

```
Armstrong numbers between 1 and 1000: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
PS C:\Users\HP\Desktop\AI-Lab>
```

Task-7:

Loops – Happy Numbers in a Range

Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).
Instructions:

• Implement the logic using a loop: repeatedly replace a number with the sum of the squares of its digits until the result is either 1 (Happy Number) or enters a cycle (Not Happy).
• Validate correctness by checking known Happy Numbers (e.g., 1, 7, 10, 13, 19, 23, 28…).
• Ask AI to regenerate an optimized version (e.g., by using a set to detect cycles instead of infinite loops).
Expected Output #8:
• Python program that prints all Happy Numbers within a range.
• Optimized version using cycle detection with explanation.

```python
#Generate a function that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).
def is_happy_number(num):
    seen = set()
    while num != 1 and num not in seen:
        seen.add(num)
        num = sum(int(digit) ** 2 for digit in str(num))
    return num == 1
def happy_numbers_in_range(start, end):
    happy_numbers = []
    for num in range(start, end + 1):
        if is_happy_number(num):
            happy_numbers.append(num)
    return happy_numbers
start_range = 1
end_range = 500
result = happy_numbers_in_range(start_range, end_range)
print(f"Happy numbers between {start_range} and {end_range}: {result}")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/AI-ass.py
c:\Users\HP\Desktop\AI-Lab\AI-ass.py:47: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
Happy numbers between 1 and 500: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100, 103, 109, 129, 130, 133, 139, 167, 176,
 188, 190, 192, 193, 203, 208, 219, 226, 230, 236, 239, 262, 263, 280, 291, 293, 301, 302, 310, 313, 319, 320, 326, 329, 331, 338, 356, 362, 365, 367, 3
68, 376, 379, 383, 386, 391, 392, 397, 404, 409, 440, 446, 464, 469, 478, 487, 490, 496]
PS C:\Users\HP\Desktop\AI-Lab>
```

Task-8:

Loops – Strong Numbers in a Range
Generate a function using AI that displays all Strong Numbers (sum of factorial of digits equals the number, e.g., 145 = 1!+4!+5!) within a given range.
Instructions:

• Use loops to extract digits and calculate factorials.
• Validate with examples (1, 2, 145).
• Ask AI to regenerate an optimized version (precompute digit factorials).
Expected Output #9:
• Python program that lists Strong Numbers.
• Optimized version with explanation

```python
#Generate a function displays all Strong Numbers (sum of factorial of digits equals the number,
# e.g., 145 = 1!+4!+5!) within a given range. with 1,2,145
import math
def is_strong_number(num):
    sum_of_factorials = sum(math.factorial(int(digit)) for digit in str(num))
    return sum_of_factorials == num
def strong_numbers_in_range(start, end):
    strong_numbers = []
    for num in range(start, end + 1):
        if is_strong_number(num):
            strong_numbers.append(num)
    return strong_numbers
start_range = 1
end_range = 500
result = strong_numbers_in_range(start_range, end_range)
print(f"Strong numbers between {start_range} and {end_range}: {result}")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/AI-ass.py
c:\Users\HP\Desktop\AI-Lab\AI-ass.py:47: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
Strong numbers between 1 and 500: [1, 2, 145]
PS C:\Users\HP\Desktop\AI-Lab>
```

Task-9:

Few-Shot Prompting for Nested Dictionary Extraction
Objective
Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.
Requirements
• The function should extract and return:
o Full Name
o Branch

o SGPA

Expected Output

A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples

```python
#generate a python code that parses a nested dictionary representing student information.
#the dictionary contains Full Name Branch SGPA
def parse_student_info(student_dict):
    for student_id, info in student_dict.items():
        full_name = info.get("Full Name", "N/A")
        branch = info.get("Branch", "N/A")
        sgpa = info.get("SGPA", "N/A")
        print(f"Student ID: {student_id}")
        print(f"Full Name: {full_name}")
        print(f"Branch: {branch}")
        print(f"SGPA: {sgpa}")
        print("-" * 20)
# Example usage
students = {
    101: {"Full Name": "John Doe", "Branch": "Computer Science", "SGPA": 8.5},
    102: {"Full Name": "Jane Smith", "Branch": "Electrical Engineering", "SGPA": 9.0},
    103: {"Full Name": "Alice Johnson", "Branch": "Mechanical Engineering", "SGPA": 8.8},
}
parse_student_info(students)
```

```
Student ID: 101
Full Name: John Doe
Branch: Computer Science
SGPA: 8.5
--------------------
--------------------
--------------------
Student ID: 102
Full Name: Jane Smith
Branch: Electrical Engineering
SGPA: 9.0
--------------------
Student ID: 103
Full Name: Alice Johnson
Branch: Mechanical Engineering
SGPA: 8.8
--------------------
PS C:\Users\HP\Desktop\AI-Lab>
```

Task – 10:

Loops – Perfect Numbers in a Range

Task: Generate a function using AI that displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000).

Instructions:

• A Perfect Number is a positive integer equal to the sum of its proper divisors (excluding itself).

o Example: 6 = 1 + 2 + 3, 28 = 1 + 2 + 4 + 7 + 14.

• Use a for loop to find divisors of each number in the range.

• Validate correctness with known Perfect Numbers (6, 28, 496…).

• Ask AI to regenerate an optimized version (using divisor check only up to √n).

Expected Output #12:

• Python program that lists Perfect Numbers in the given range.

• Optimized version with explanation.

```python
#Generate a python code displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000). using for loop
def is_perfect_number(num):
    if num < 2:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum == num
def perfect_numbers_in_range(start, end):
    perfect_numbers = []
    for num in range(start, end + 1):
        if is_perfect_number(num):
            perfect_numbers.append(num)
    return perfect_numbers
start_range = 1
end_range = 1000
result = perfect_numbers_in_range(start_range, end_range)
print(f"Perfect numbers between {start_range} and {end_range}: {result}")
```

```
Perfect numbers between 1 and 1000: [6, 28, 496]
PS C:\Users\HP\Desktop\AI-Lab>
```