Lab Assignment 6.3

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.

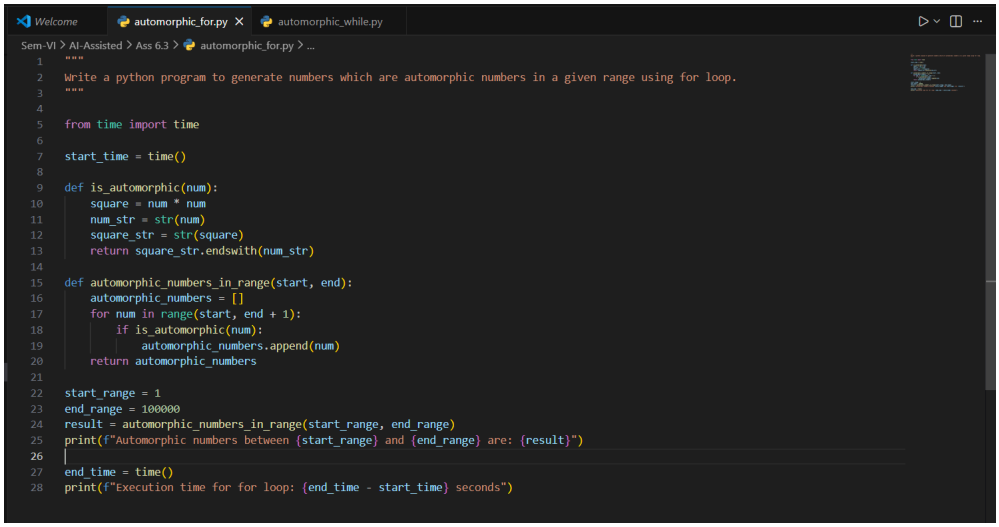
   Instructions:

   - Get AI-generated code to list Automorphic numbers using a for loop.
   - Analyze the correctness and efficiency of the generated logic.
   - Ask AI to regenerate using a while loop and compare both implementations.

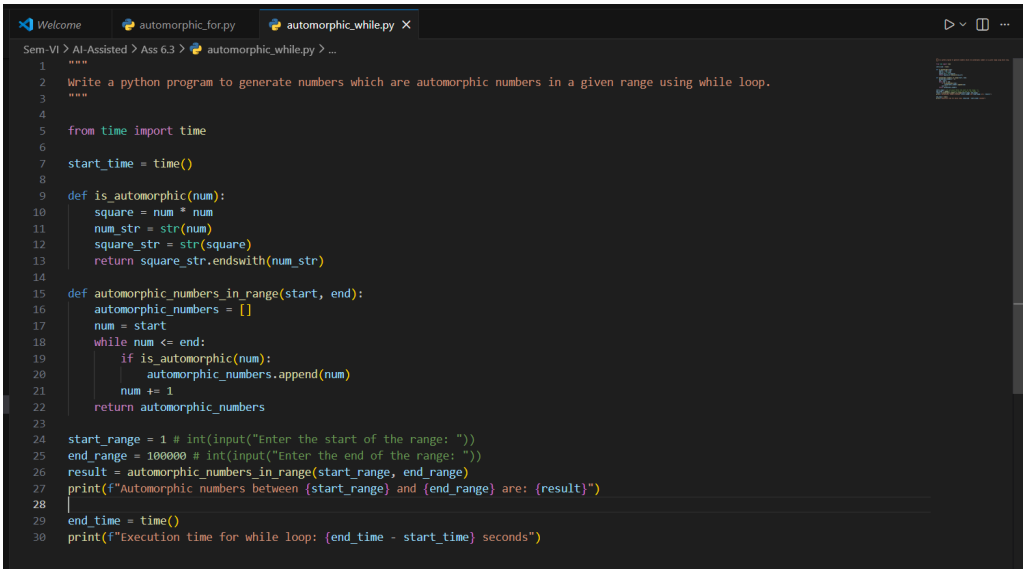
   Expected Output:

   Correct implementation that lists Automorphic numbers using both loop types, with explanation.

For Loop:



While Loop:



Execution time difference:

```
PS C:\Users\rhars\Documents\.Dev\Uni> & C:/Python314/python.exe "c:/Users/rhars/Documents/.Dev/Uni/Sem-VI/AI-Assisted/Ass 6.3/automorphic_for.py"
Automorphic numbers between 1 and 100000 are: [1, 5, 6, 25, 76, 376, 625, 9376, 90625]
Execution time for for loop: 0.03836226463317871 seconds
PS C:\Users\rhars\Documents\.Dev\Uni> & C:/Python314/python.exe "c:/Users/rhars/Documents/.Dev/Uni/Sem-VI/AI-Assisted/Ass 6.3/automorphic_while.py"
Automorphic numbers between 1 and 100000 are: [1, 5, 6, 25, 76, 376, 625, 9376, 90625]
Execution time for while loop: 0.04222440719604492 seconds
```

Observation:

Since the range is given we should use for loop, as while loop is used for unknown ranges, and for this particular situation for loop has a better execution time.

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).

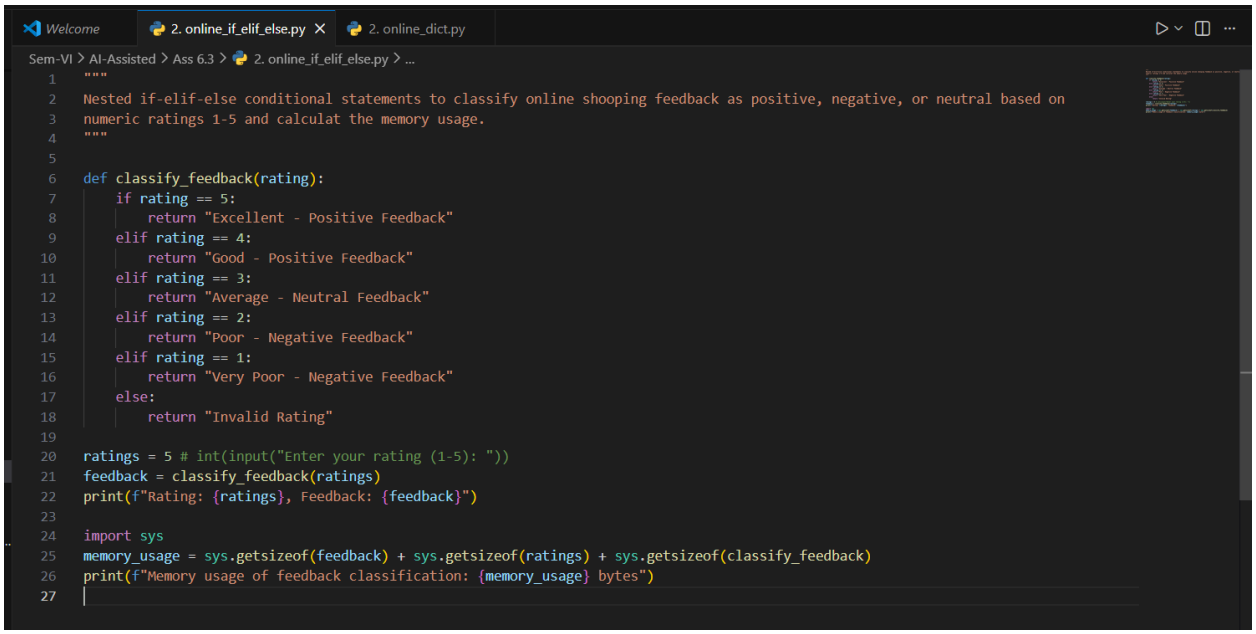
Instructions:

- Generate initial code using nested if-elif-else.
- Analyze correctness and readability.
- Ask AI to rewrite using dictionary-based or match-case structure.

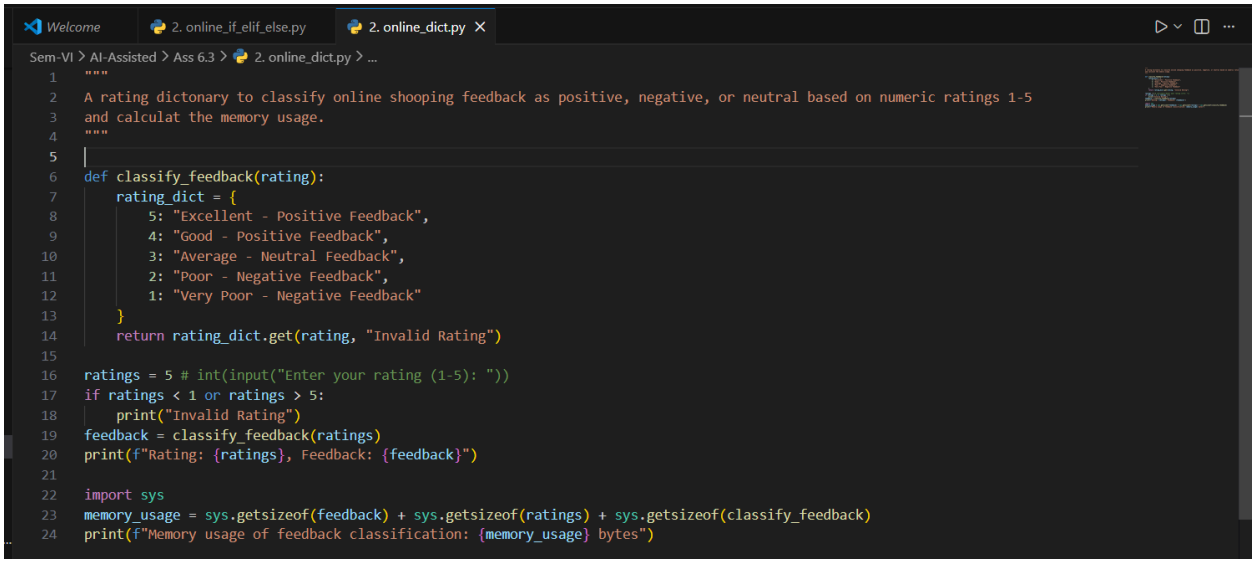
Expected Output:

Feedback classification function with explanation and an alternative approach.

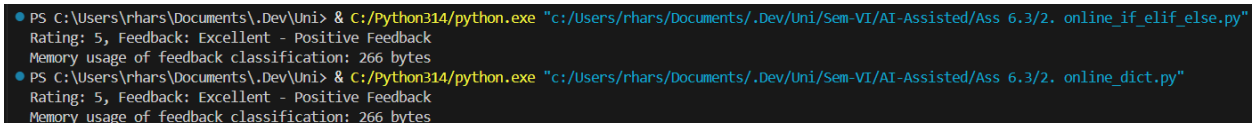
Nested if-elif-else based classification:



Dictionary based classification:



Memory usage comparison:



Observation:

- Both Dictionary and Nested if-elif-else based comparison has the same memory usage
- The major difference comes from the readability and the preference, where dict is always preferred as its lookup time is O(1)

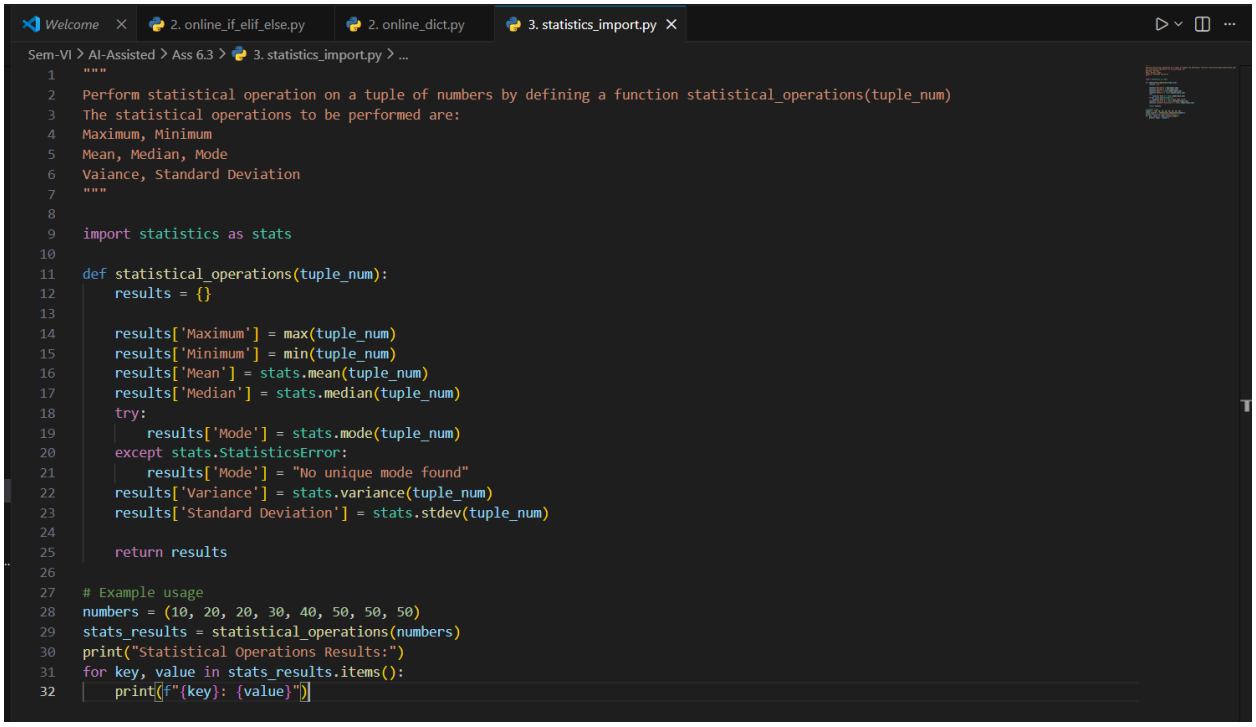
3. Statistical Operations

Task:

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



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.

5. Zero-Shot Prompting with Conditional Validation

Task:

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

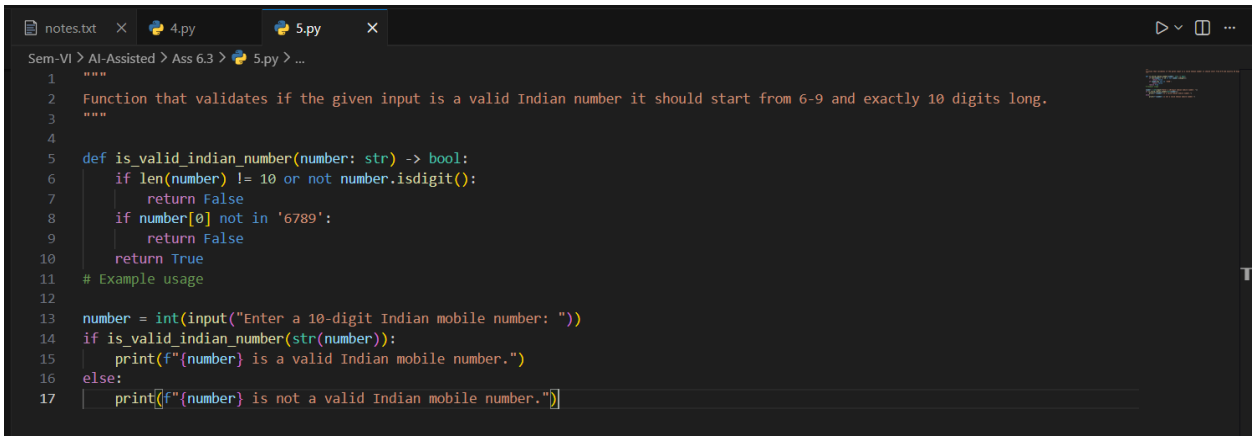
Requirements:

- The function must ensure the mobile number:
- Starts with 6, 7, 8, or 9

- Contains exactly 10 digits

Expected Output:

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



6. (Loops – Armstrong Numbers in a Range)

Task:

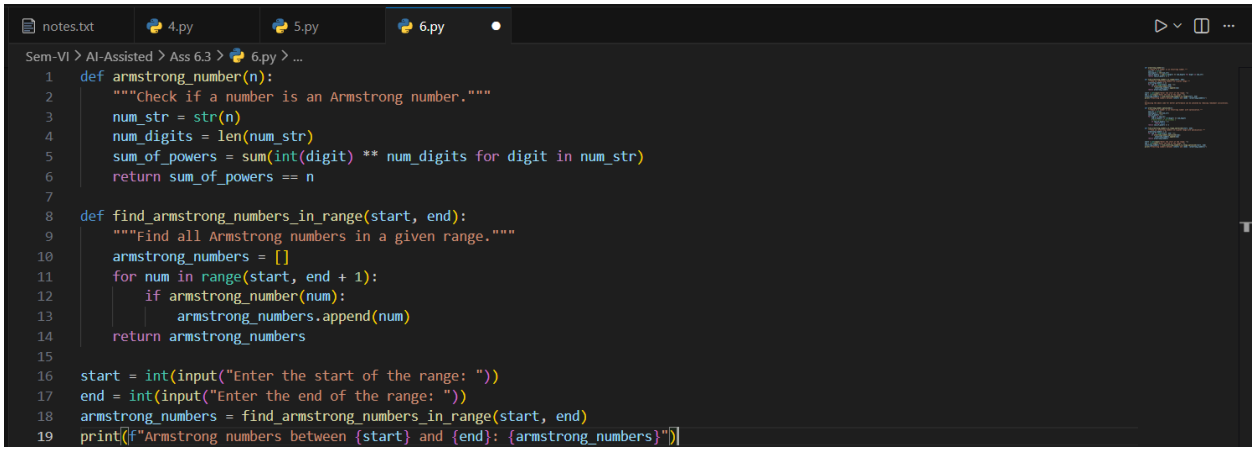
Write a function using AI that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).

Instructions:

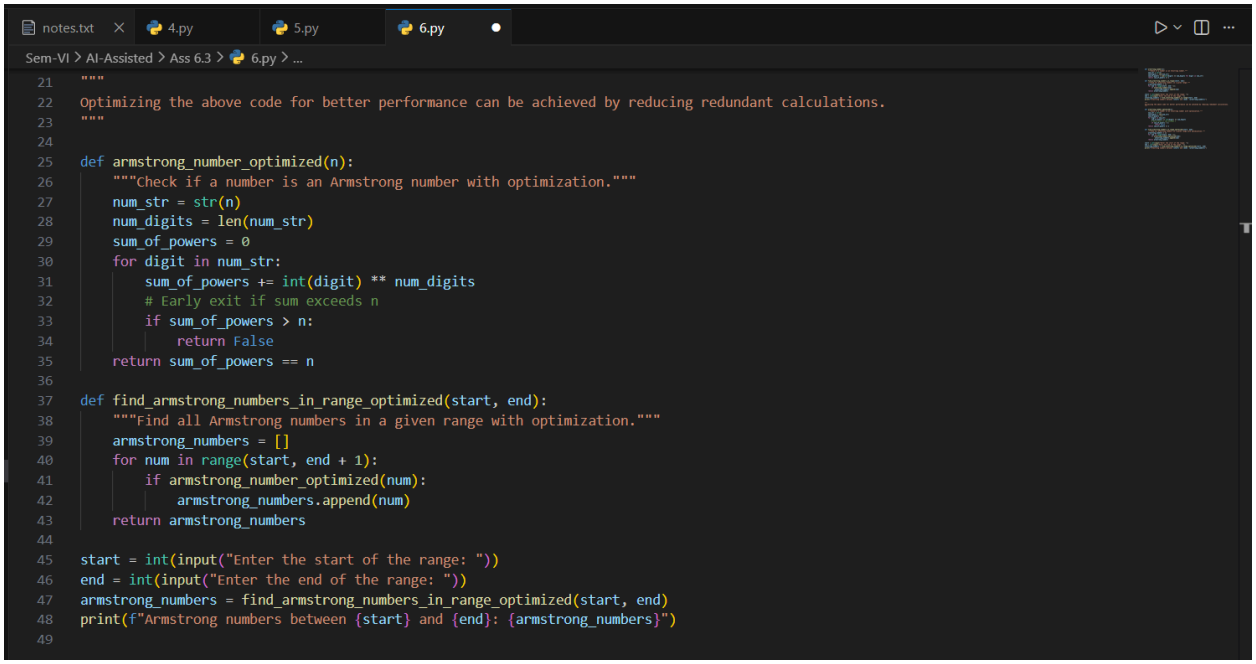
- Use a for loop and digit power logic.
- Validate correctness by checking known Armstrong numbers (153, 370, etc.).
- Ask AI to regenerate an optimized version (using list comprehensions).

Expected Output:

- Python program listing Armstrong numbers in the range.
- Optimized version with explanation.



Optimized Version:



7. (Loops – Happy Numbers in a Range)

Task:

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:

- Python program that prints all Happy Numbers within a range.
- Optimized version using cycle detection with explanation.

notes.txt | 4.py | 5.py | 6.py | 7.py

Sem-VI > AI-Assisted > Ass 6.3 > 7.py > ...

```python

def happy_number(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit) ** 2 for digit in str(n))
    return n == 1

def find_happy_numbers_in_range(start, end):
    happy_numbers = []
    for num in range(start, end + 1):
        if happy_number(num):
            happy_numbers.append(num)
    return happy_numbers

start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))
happy_numbers = find_happy_numbers_in_range(start, end)
print(f"Happy numbers between {start} and {end}: {happy_numbers}")
```

8. (Loops – Strong Numbers in a Range)

Task:

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:

- Python program that lists Strong Numbers.
- Optimized version with explanation.

notes.txt | 4.py | 5.py | 6.py | 7.py | 8.py

Sem-VI > AI-Assisted > Ass 6.3 > 8.py > ...

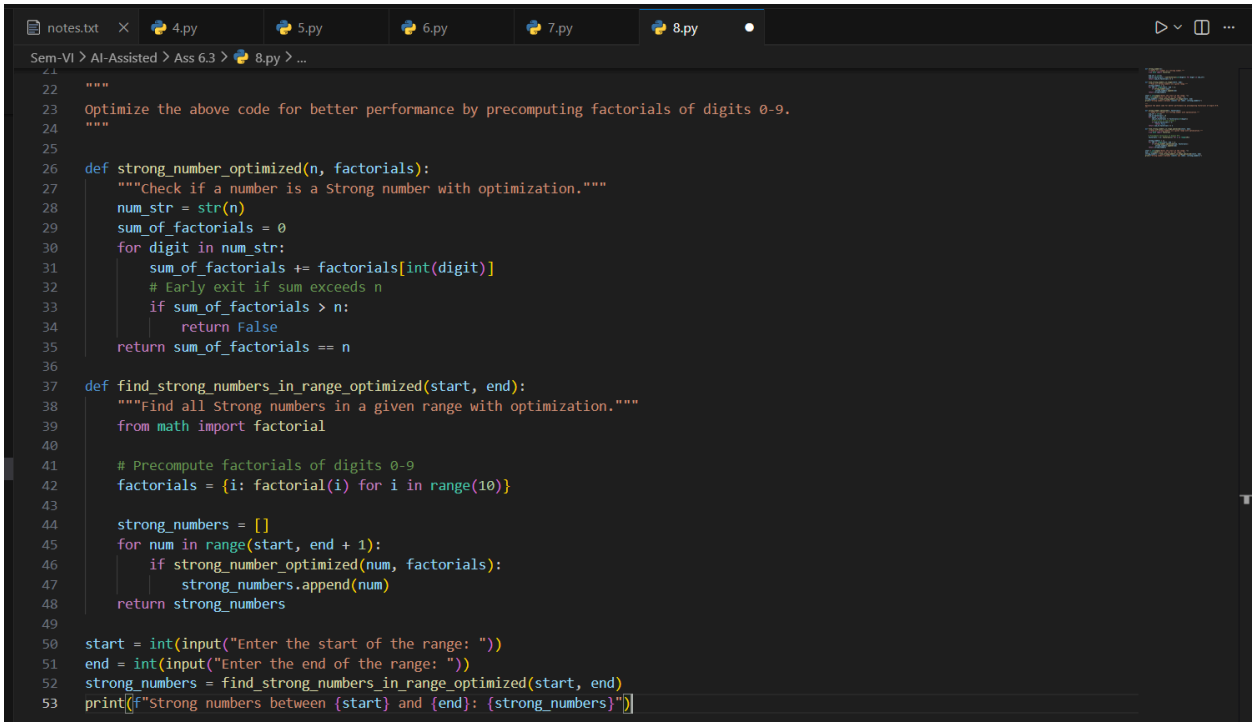
```python
def strong_number(n):
    """Check if a number is a Strong number."""
    from math import factorial

    num_str = str(n)
    sum_of_factorials = sum(factorial(int(digit)) for digit in num_str)
    return sum_of_factorials == n

def find_strong_numbers_in_range(start, end):
    """Find all Strong numbers in a given range."""
    strong_numbers = []
    for num in range(start, end + 1):
        if strong_number(num):
            strong_numbers.append(num)
    return strong_numbers

start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))
strong_numbers = find_strong_numbers_in_range(start, end)
print(f"Strong numbers between {start} and {end}: {strong_numbers}")
```

Optimized Version:



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:

- Full Name
- Branch
- SGPA

Expected Output:

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

notes.txt | 4.py | 5.py | 6.py | 7.py | 8.py | 9.py

Sem-VI > AI-Assisted > Ass 6.3 > 9.py > ...

```python
"""
A function that parses a nested dictonary representing student information it should return Full Name, Branch, SGPA.
Example:
Input:
ex1 = {
    "student_1" : {
        "Full Name": "Alice Johnson",
        "Branch": "Computer Science",
        "SGPA": 9.1
    }
}
ex2 = {
    "student_2" : {
        "Full Name": "Bob Smith",
        "Branch": "Mechanical Engineering",
        "SGPA": 8.5
    }
}
ex3 = {
    "student_3" : {
        "Full Name": "Charlie Brown",
        "Branch": "Electrical Engineering",
        "SGPA": 8.9
    },
    "student_4" : {
        "Full Name": "Diana Prince",
        "Branch": "Civil Engineering",
        "SGPA": 9.3
    }
}
"""

def parse_student_info(student_dict):
    """Parse a nested dictionary to extract student information."""
    for student_key, 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")
        return full_name, branch, sgpa
# Example usage
ex1 = {
    "student_1" : {
        "Full Name": "Alice Johnson",
        "Branch": "Computer Science",
        "SGPA": 9.1
    }
}

full_name, branch, sgpa = parse_student_info(ex1)
print(f"Full Name: {full_name}, Branch: {branch}, SGPA: {sgpa}")
```

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).
    - 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:

- Python program that lists Perfect Numbers in the given range.
- Optimized version with explanation.