AI Assisted Coding

LAB ASSIGNMENT - 6.3

Name : G.Abhiram
HT.No : 2303A51087
Batch No : 02

Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals

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.

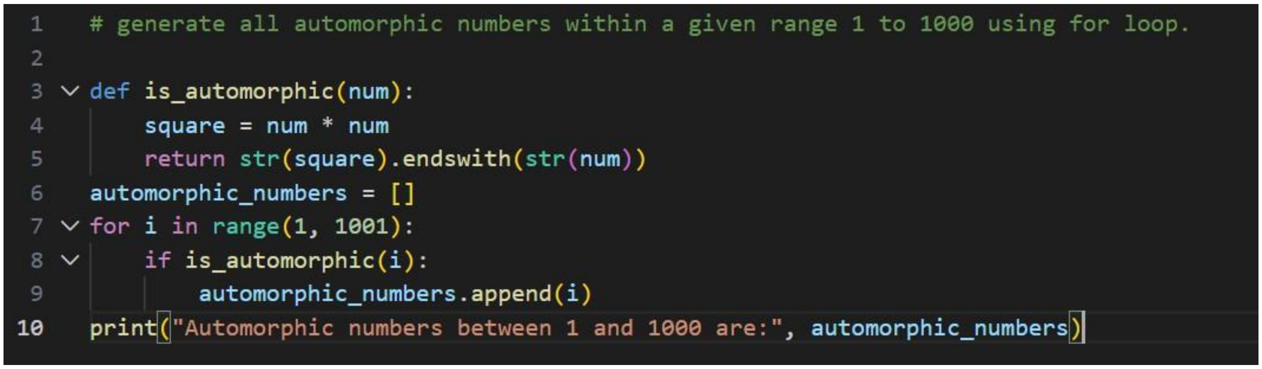
• **Instructions:**

- 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 #1:

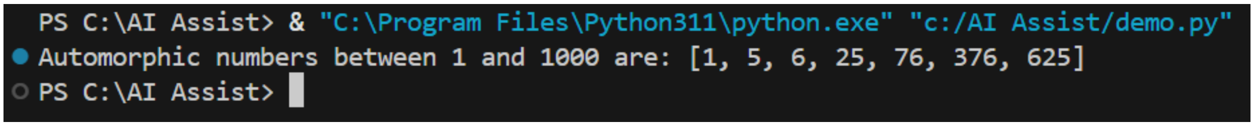
- Correct implementation that lists Automorphic numbers using both loop types, with explanation.

For Loop Code :



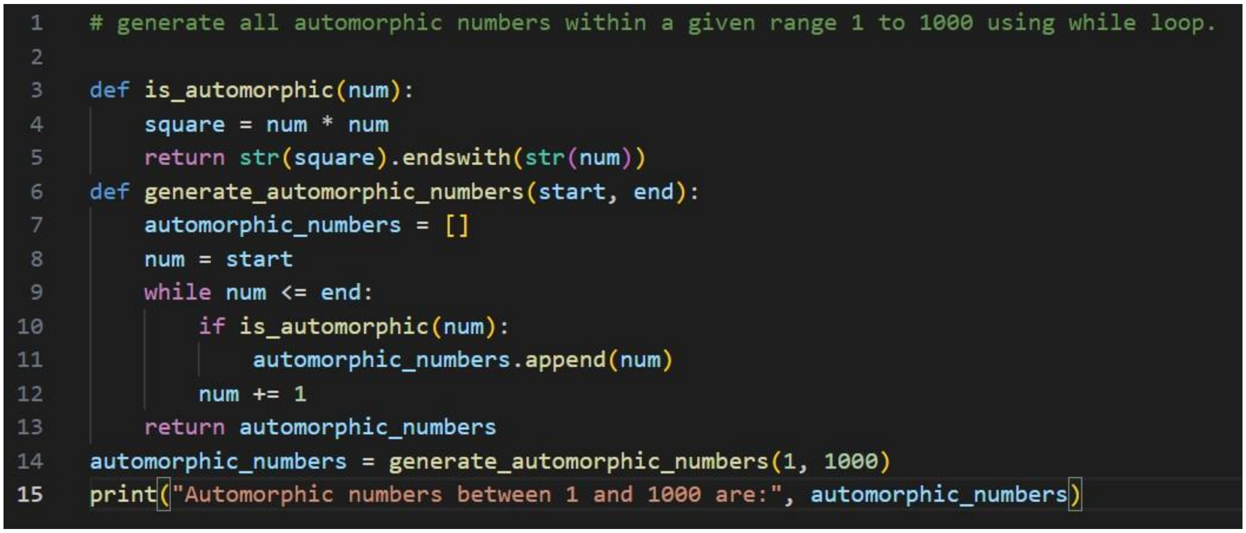
```python
# generate all automorphic numbers within a given range 1 to 1000 using for loop.

def is_automorphic(num):
    square = num * num
    return str(square).endswith(str(num))
automorphic_numbers = []
for i in range(1, 1001):
    if is_automorphic(i):
        automorphic_numbers.append(i)
print("Automorphic numbers between 1 and 1000 are:", automorphic_numbers)
```

Output :



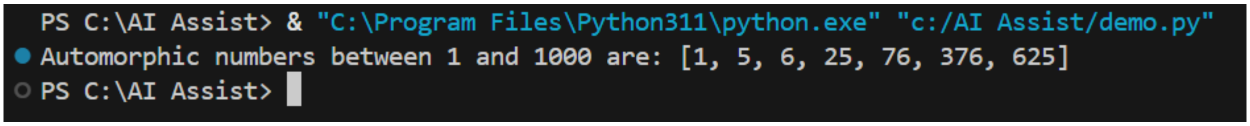
```
 PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]
○ PS C:\AI Assist> █
```

While Loop Code :



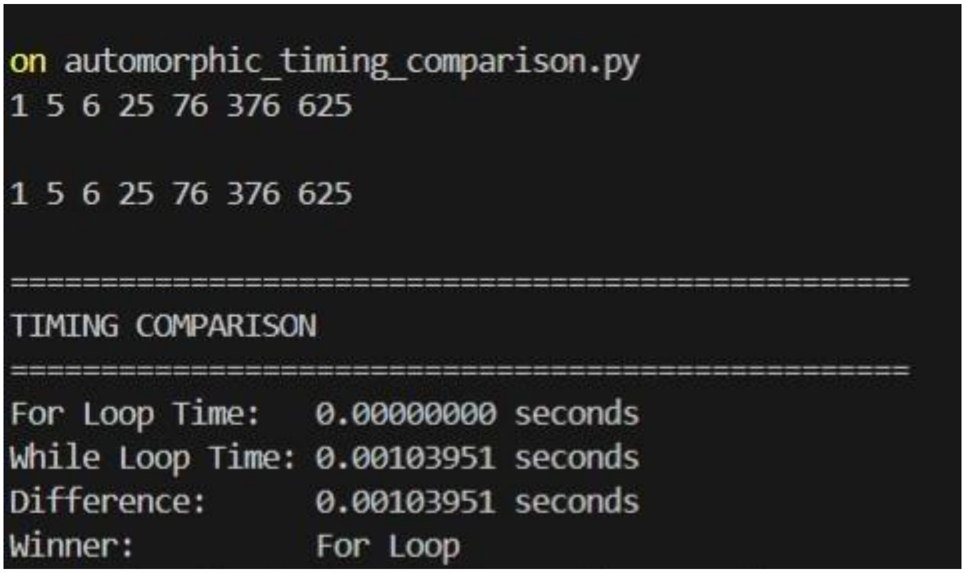
```python
# generate all automorphic numbers within a given range 1 to 1000 using while loop.

def is_automorphic(num):
    square = num * num
    return str(square).endswith(str(num))
def generate_automorphic_numbers(start, end):
    automorphic_numbers = []
    num = start
    while num <= end:
        if is_automorphic(num):
            automorphic_numbers.append(num)
        num += 1
    return automorphic_numbers
automorphic_numbers = generate_automorphic_numbers(1, 1000)
print("Automorphic numbers between 1 and 1000 are:", automorphic_numbers)
```

Output :



```
 PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]
○ PS C:\AI Assist> █
```

Comparison of for loop & while loop :



```
on automorphic_timing_comparison.py
1 5 6 25 76 376 625

1 5 6 25 76 376 625

==================================================
TIMING COMPARISON
==================================================
For Loop Time:   0.00000000 seconds
While Loop Time: 0.00103951 seconds
Difference:      0.00103951 seconds
Winner:          For Loop
```

Task Description #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).

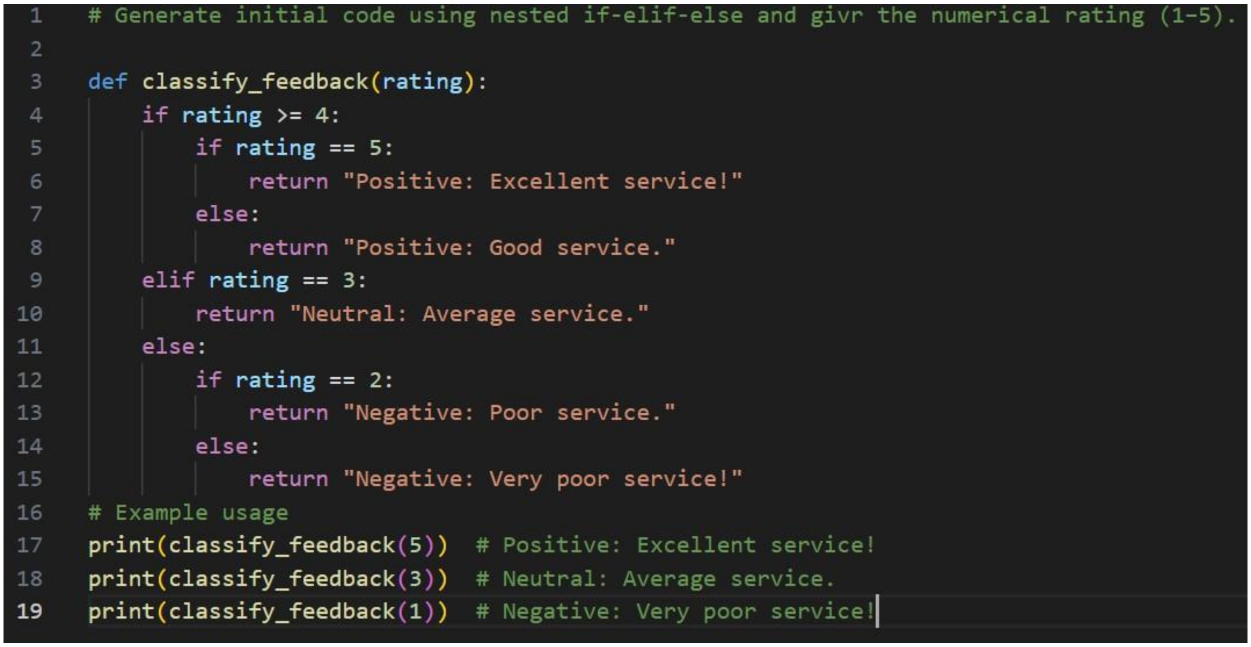
• Instructions:

- 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 #2:

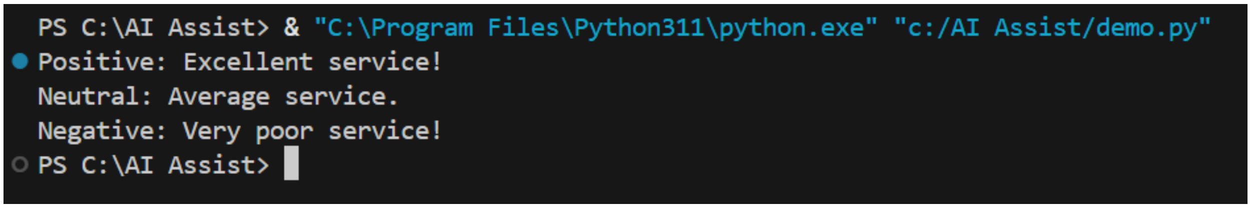
• Feedback classification function with explanation and an alternative Approach.

Code :



```python
# Generate initial code using nested if-elif-else and givr the numerical rating (1-5).

def classify_feedback(rating):
    if rating >= 4:
        if rating == 5:
            return "Positive: Excellent service!"
        else:
            return "Positive: Good service."
    elif rating == 3:
        return "Neutral: Average service."
    else:
        if rating == 2:
            return "Negative: Poor service."
        else:
            return "Negative: Very poor service!"
# Example usage
print(classify_feedback(5))  # Positive: Excellent service!
print(classify_feedback(3))  # Neutral: Average service.
print(classify_feedback(1))  # Negative: Very poor service!
```

Output :



```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
Positive: Excellent service!
Neutral: Average service.
Negative: Very poor service!
PS C:\AI Assist>
```

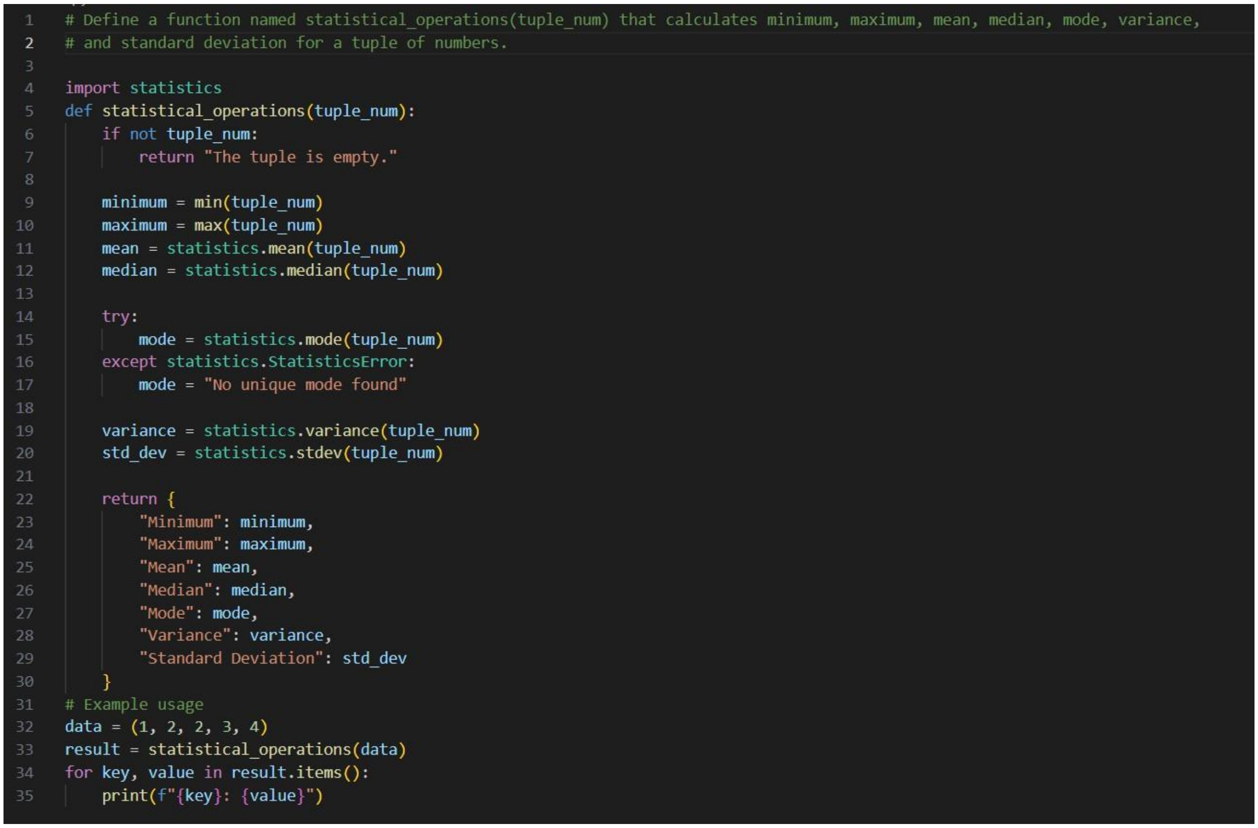
Task 3: Statistical_operations

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

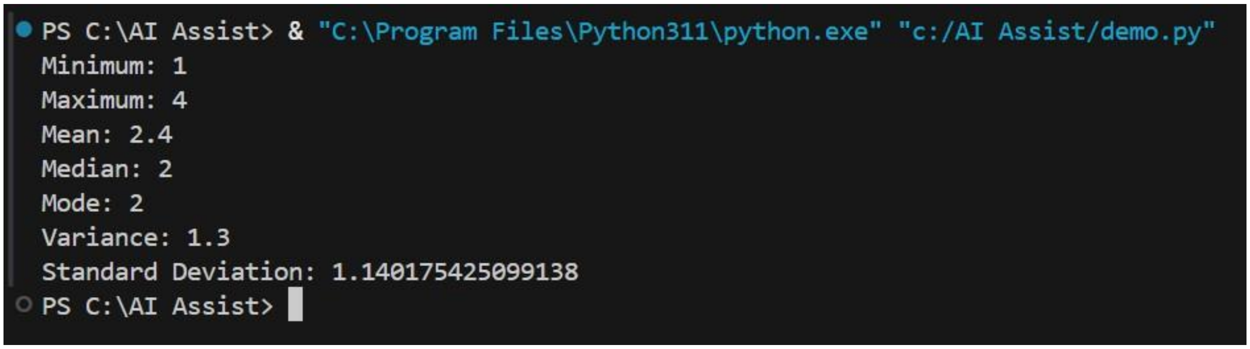
While writing the function, observe the code suggestions provided by GitHub Copilot.Make decisions to accept, reject, or modify the suggestions based on their relevance and correctness.

Code :



```python
# Define a function named statistical_operations(tuple_num) that calculates minimum, maximum, mean, median, mode, variance,
# and standard deviation for a tuple of numbers.

import statistics
def statistical_operations(tuple_num):
    if not tuple_num:
        return "The tuple is empty."

    minimum = min(tuple_num)
    maximum = max(tuple_num)
    mean = statistics.mean(tuple_num)
    median = statistics.median(tuple_num)

    try:
        mode = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        mode = "No unique mode found"

    variance = statistics.variance(tuple_num)
    std_dev = statistics.stdev(tuple_num)

    return {
        "Minimum": minimum,
        "Maximum": maximum,
        "Mean": mean,
        "Median": median,
        "Mode": mode,
        "Variance": variance,
        "Standard Deviation": std_dev
    }
# Example usage
data = (1, 2, 2, 3, 4)
result = statistical_operations(data)
for key, value in result.items():
    print(f"{key}: {value}")
```

Output :

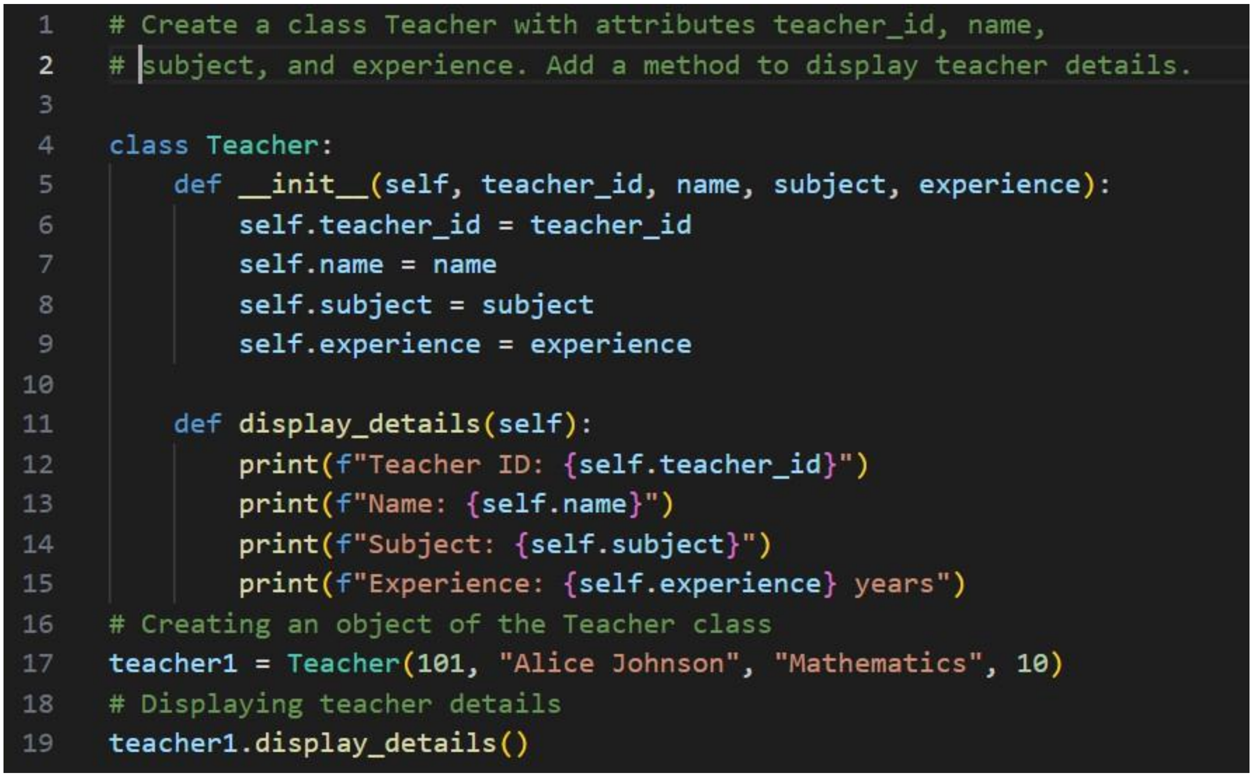


```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
Minimum: 1
Maximum: 4
Mean: 2.4
Median: 2
Mode: 2
Variance: 1.3
Standard Deviation: 1.140175425099138
PS C:\AI Assist>
```

Task 4: Teacher Profile

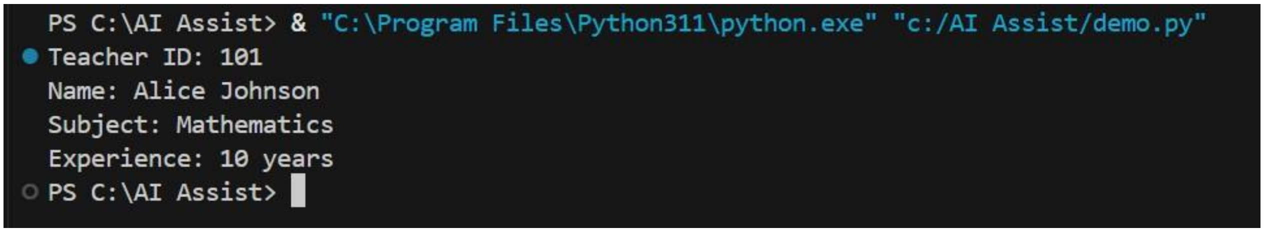
- **Prompt:** Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.
- **Expected Output:** Class with initializer, method, and object creation.

Code :



```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")
# Creating an object of the Teacher class
teacher1 = Teacher(101, "Alice Johnson", "Mathematics", 10)
# Displaying teacher details
teacher1.display_details()
```

Output :



```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
Teacher ID: 101
Name: Alice Johnson
Subject: Mathematics
Experience: 10 years
PS C:\AI Assist>
```

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.

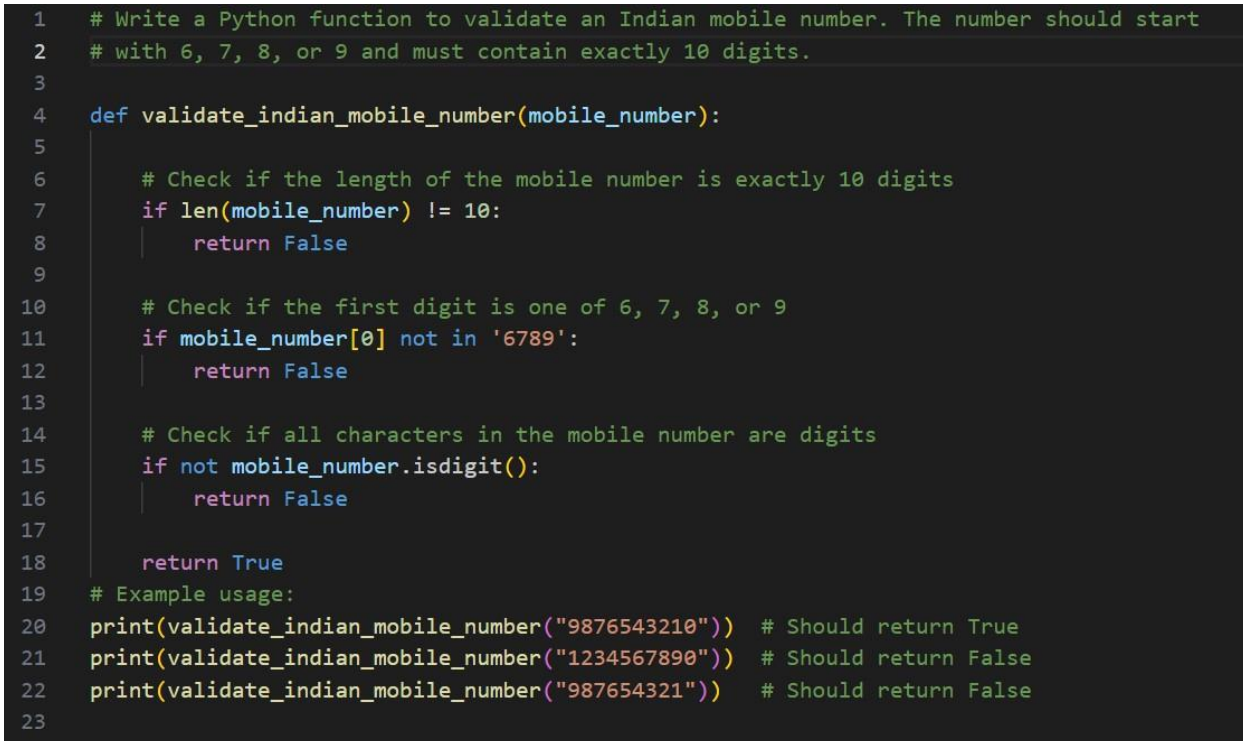
Requirements

- The function must ensure the mobile number:
- Starts with 6, 7, 8, or 9
- Contains exactly 10 digits

Expected Output

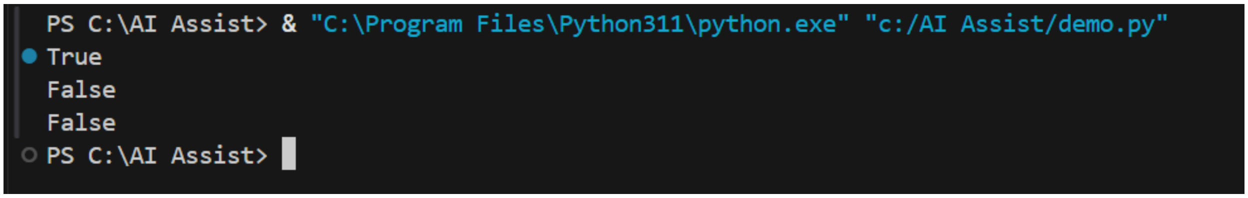
• A valid Python function that performs all required validations without using any input-output examples in the prompt.

Code :



```python
# Write a Python function to validate an Indian mobile number. The number should start
# with 6, 7, 8, or 9 and must contain exactly 10 digits.

def validate_indian_mobile_number(mobile_number):

    # Check if the length of the mobile number is exactly 10 digits
    if len(mobile_number) != 10:
        return False

    # Check if the first digit is one of 6, 7, 8, or 9
    if mobile_number[0] not in '6789':
        return False

    # Check if all characters in the mobile number are digits
    if not mobile_number.isdigit():
        return False

    return True
# Example usage:
print(validate_indian_mobile_number("9876543210"))  # Should return True
print(validate_indian_mobile_number("1234567890"))  # Should return False
print(validate_indian_mobile_number("987654321"))   # Should return False

```

Output :



```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
True
False
False
PS C:\AI Assist>
```

Task Description #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).

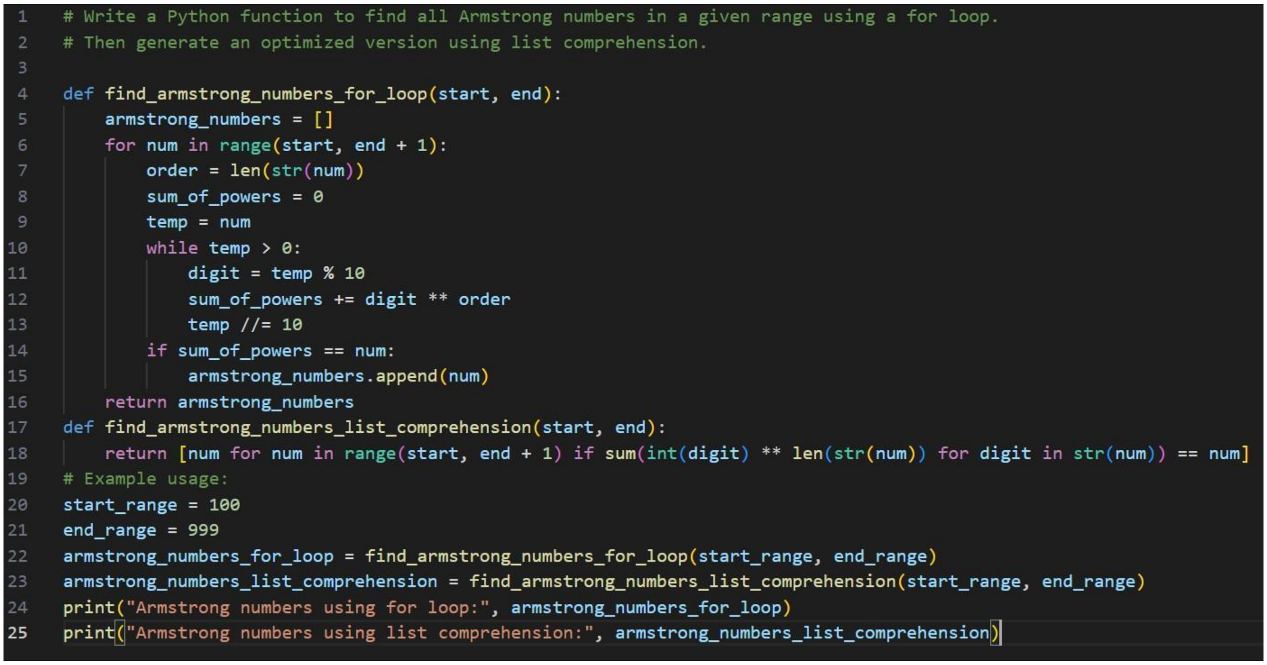
Instructions:

- 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 :

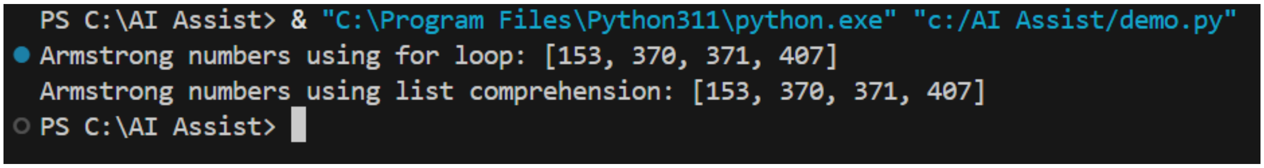
- Python program listing Armstrong numbers in the range.
- Optimized version with explanation.

Code :



```python
# Write a Python function to find all Armstrong numbers in a given range using a for loop.
# Then generate an optimized version using list comprehension.

def find_armstrong_numbers_for_loop(start, end):
    armstrong_numbers = []
    for num in range(start, end + 1):
        order = len(str(num))
        sum_of_powers = 0
        temp = num
        while temp > 0:
            digit = temp % 10
            sum_of_powers += digit ** order
            temp //= 10
        if sum_of_powers == num:
            armstrong_numbers.append(num)
    return armstrong_numbers
def find_armstrong_numbers_list_comprehension(start, end):
    return [num for num in range(start, end + 1) if sum(int(digit) ** len(str(num)) for digit in str(num)) == num]
# Example usage:
start_range = 100
end_range = 999
armstrong_numbers_for_loop = find_armstrong_numbers_for_loop(start_range, end_range)
armstrong_numbers_list_comprehension = find_armstrong_numbers_list_comprehension(start_range, end_range)
print("Armstrong numbers using for loop:", armstrong_numbers_for_loop)
print("Armstrong numbers using list comprehension:", armstrong_numbers_list_comprehension)
```

Output :



```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
Armstrong numbers using for loop: [153, 370, 371, 407]
Armstrong numbers using list comprehension: [153, 370, 371, 407]
PS C:\AI Assist>
```

Task Description #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).

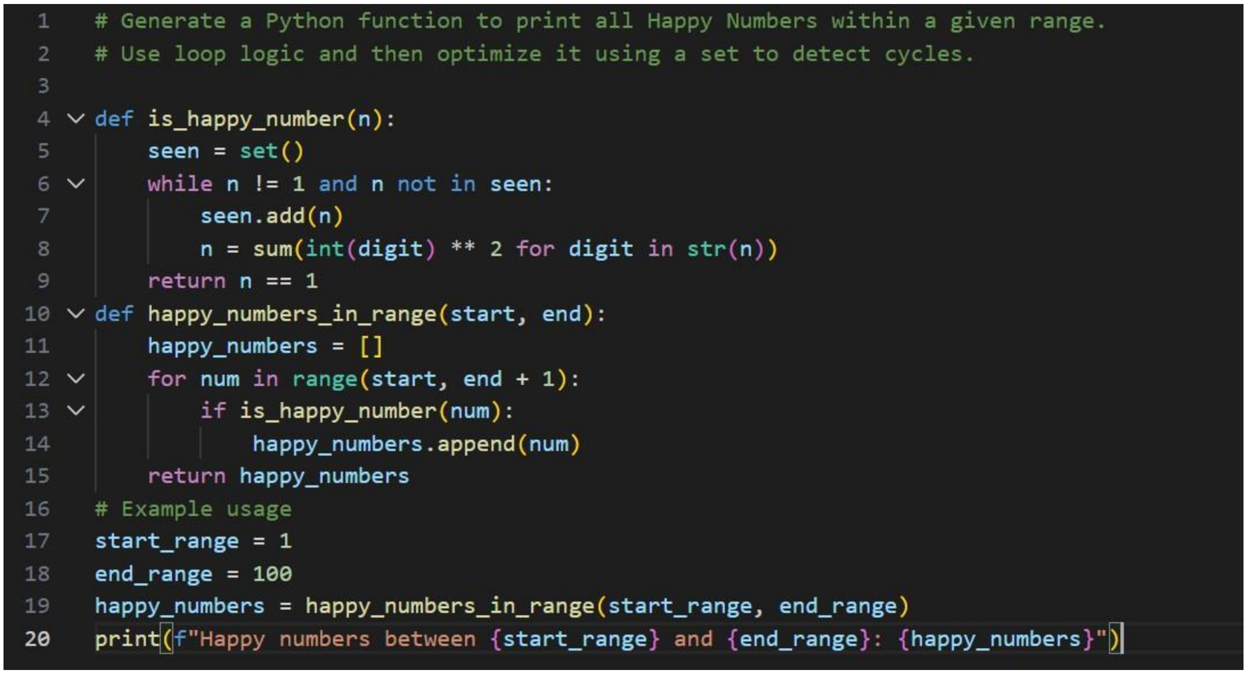
Instructions:

- 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 #8:

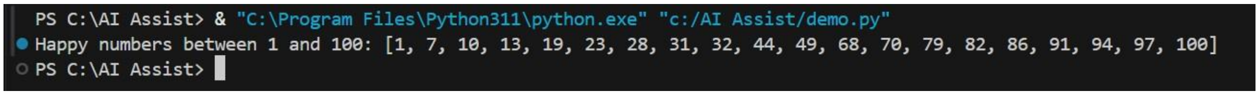
- Python program that prints all Happy Numbers within a range.
- Optimized version using cycle detection with explanation.

Code :



```python
# Generate a Python function to print all Happy Numbers within a given range.
# Use loop logic and then optimize it using a set to detect cycles.

def is_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 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
# Example usage
start_range = 1
end_range = 100
happy_numbers = happy_numbers_in_range(start_range, end_range)
print(f"Happy numbers between {start_range} and {end_range}: {happy_numbers}")
```

Output :



```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
Happy numbers between 1 and 100: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
PS C:\AI Assist> 
```

Task Description #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.

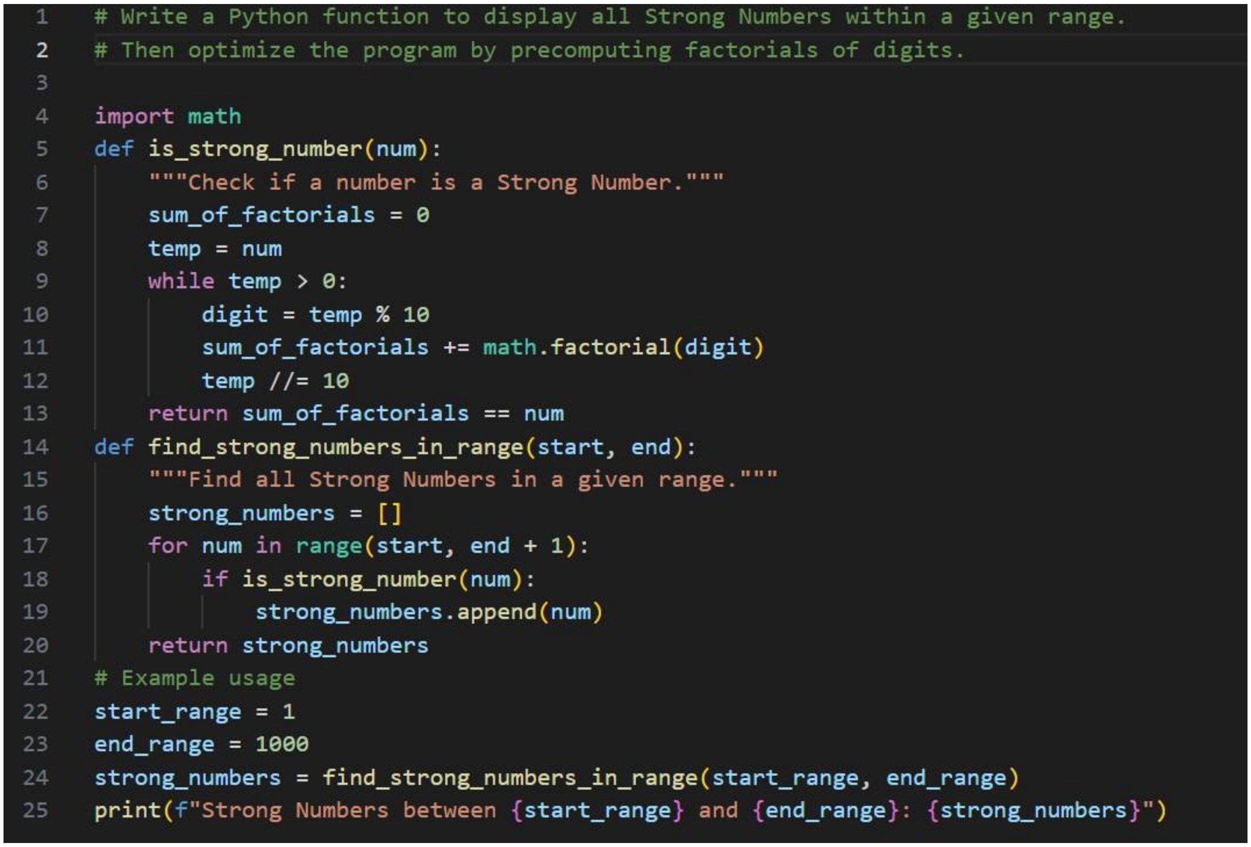
Instructions:

- 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 #9:

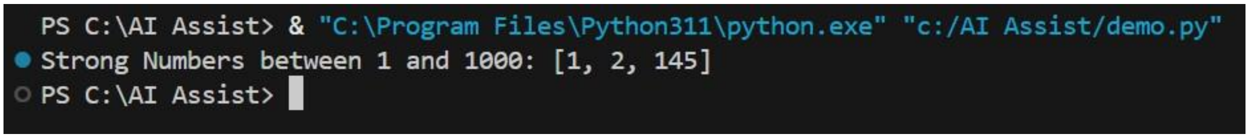
- Python program that lists Strong Numbers.
- Optimized version with explanation.

Code :



```python
# Write a Python function to display all Strong Numbers within a given range.
# Then optimize the program by precomputing factorials of digits.

import math
def is_strong_number(num):
    """Check if a number is a Strong Number."""
    sum_of_factorials = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum_of_factorials += math.factorial(digit)
        temp //= 10
    return sum_of_factorials == num
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 is_strong_number(num):
            strong_numbers.append(num)
    return strong_numbers
# Example usage
start_range = 1
end_range = 1000
strong_numbers = find_strong_numbers_in_range(start_range, end_range)
print(f"Strong Numbers between {start_range} and {end_range}: {strong_numbers}")
```

Output :



```
  PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Strong Numbers between 1 and 1000: [1, 2, 145]
○ PS C:\AI Assist> 
```

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.

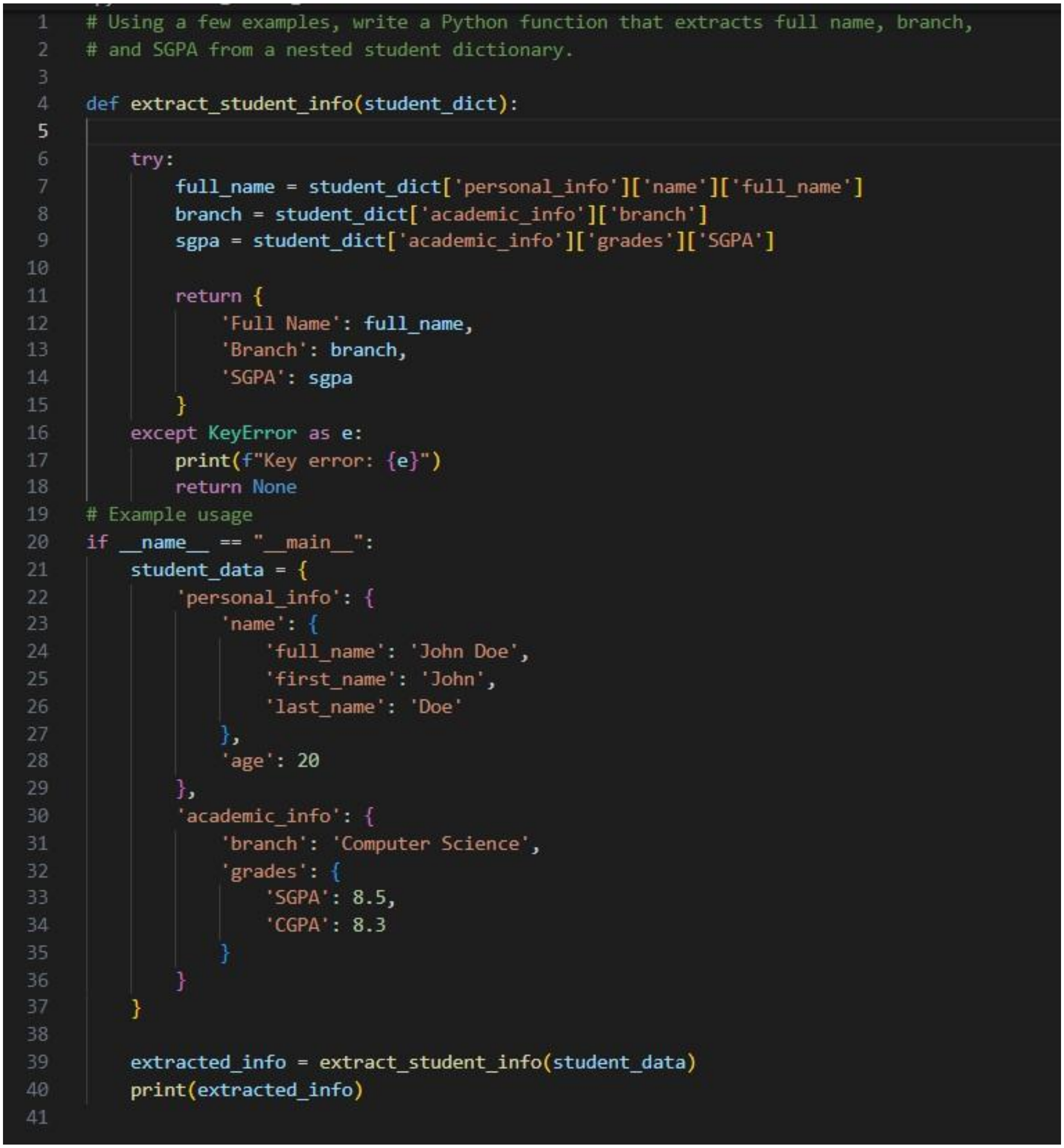
Requirements

- The function should extract and return:
- Full Name
- Branch
- SGPA

Expected Output

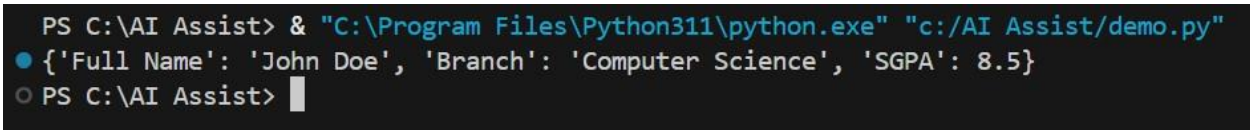
A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples

Code :



```python
# Using a few examples, write a Python function that extracts full name, branch,
# and SGPA from a nested student dictionary.

def extract_student_info(student_dict):

    try:
        full_name = student_dict['personal_info']['name']['full_name']
        branch = student_dict['academic_info']['branch']
        sgpa = student_dict['academic_info']['grades']['SGPA']

        return {
            'Full Name': full_name,
            'Branch': branch,
            'SGPA': sgpa
        }
    except KeyError as e:
        print(f"Key error: {e}")
        return None
# Example usage
if __name__ == "__main__":
    student_data = {
        'personal_info': {
            'name': {
                'full_name': 'John Doe',
                'first_name': 'John',
                'last_name': 'Doe'
            },
            'age': 20
        },
        'academic_info': {
            'branch': 'Computer Science',
            'grades': {
                'SGPA': 8.5,
                'CGPA': 8.3
            }
        }
    }

    extracted_info = extract_student_info(student_data)
    print(extracted_info)

```

Output :



```
  PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● {'Full Name': 'John Doe', 'Branch': 'Computer Science', 'SGPA': 8.5}
○ PS C:\AI Assist> 
```

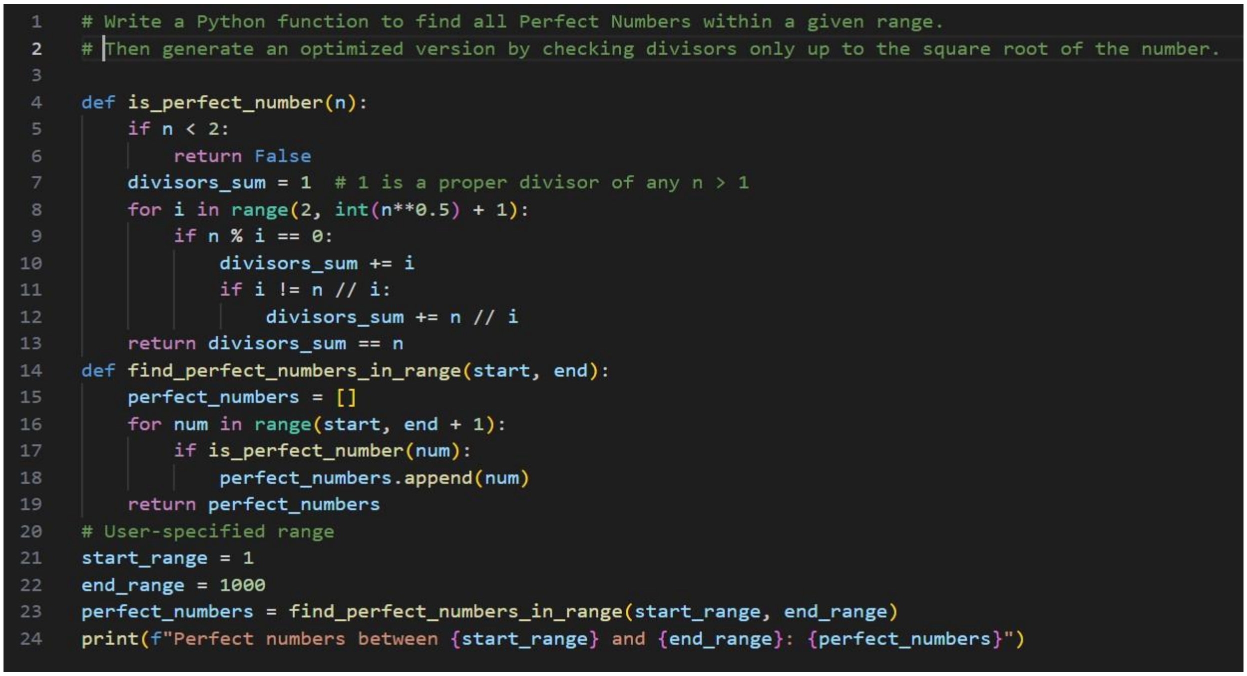
Task Description #10 (Loops – Perfect Numbers in a Range)

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

Instructions:

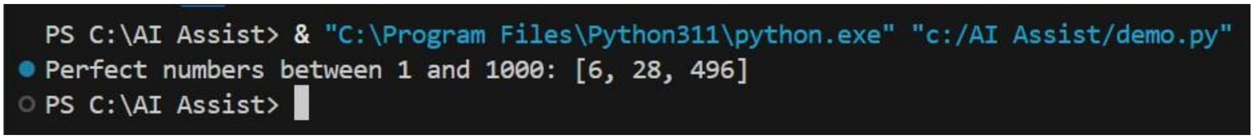
- A Perfect Number is a positive integer equal to the sum of its proper divisors (excluding itself).
  Example: 6 = 1 + 2 + 3, 28 = 1 + 2 + 4 + 7 + 14.
- Use a for loop to find divisors of each number in the range.
- Validate correctness with known Perfect Numbers (6, 28, 496…).
- Ask AI to regenerate an optimized version (using divisor check only up to √n).

Code :



```python
# Write a Python function to find all Perfect Numbers within a given range.
# Then generate an optimized version by checking divisors only up to the square root of the number.

def is_perfect_number(n):
    if n < 2:
        return False
    divisors_sum = 1  # 1 is a proper divisor of any n > 1
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            divisors_sum += i
            if i != n // i:
                divisors_sum += n // i
    return divisors_sum == n
def find_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
# User-specified range
start_range = 1
end_range = 1000
perfect_numbers = find_perfect_numbers_in_range(start_range, end_range)
print(f"Perfect numbers between {start_range} and {end_range}: {perfect_numbers}")
```

Output :



```
  PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Perfect numbers between 1 and 1000: [6, 28, 496]
○ PS C:\AI Assist> 
```