AIAC Assignment-6

Name : K.Nikshitha

Ht.no : 2303A51692

Batch 24

#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.

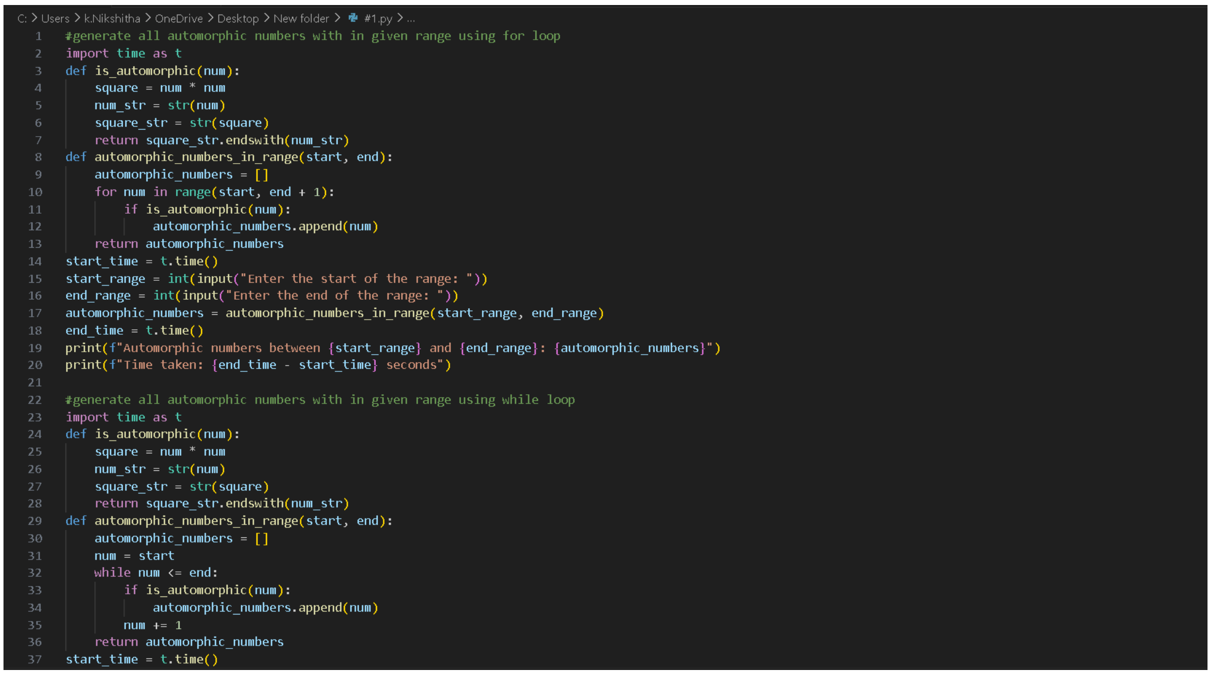
o Ask AI to regenerate using a while loop and compare both

implementations.

Expected Output #1:

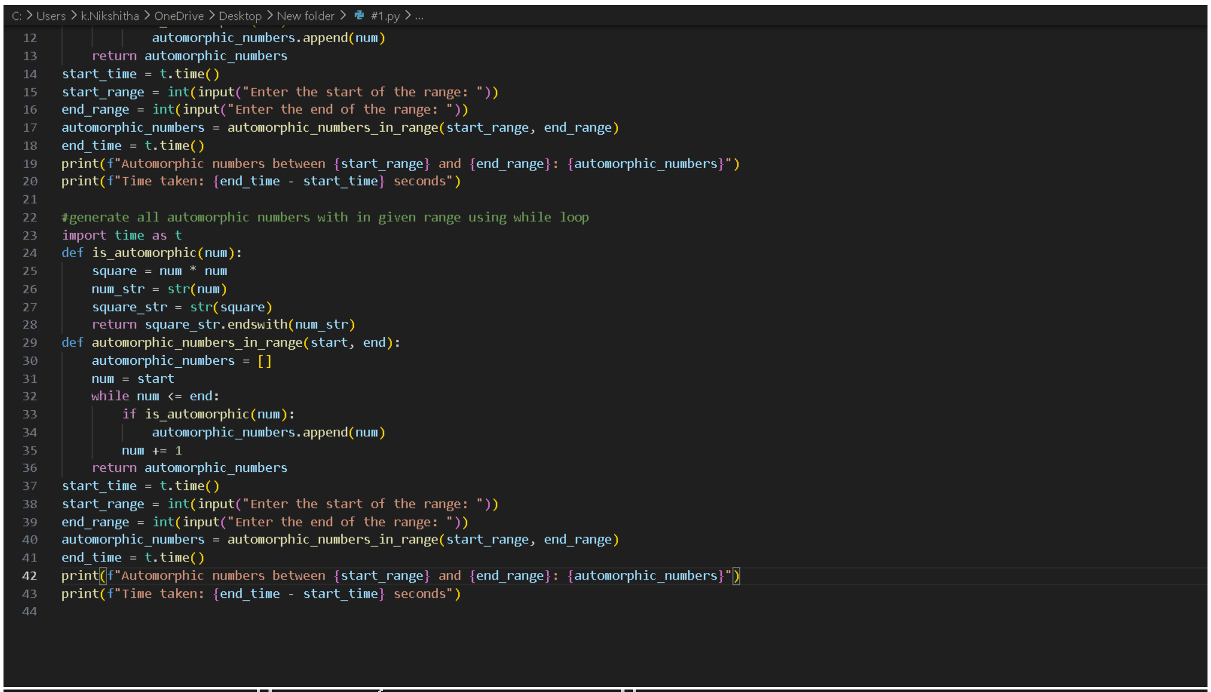
• Correct implementation that lists Automorphic numbers using both

loop types, with explanation.

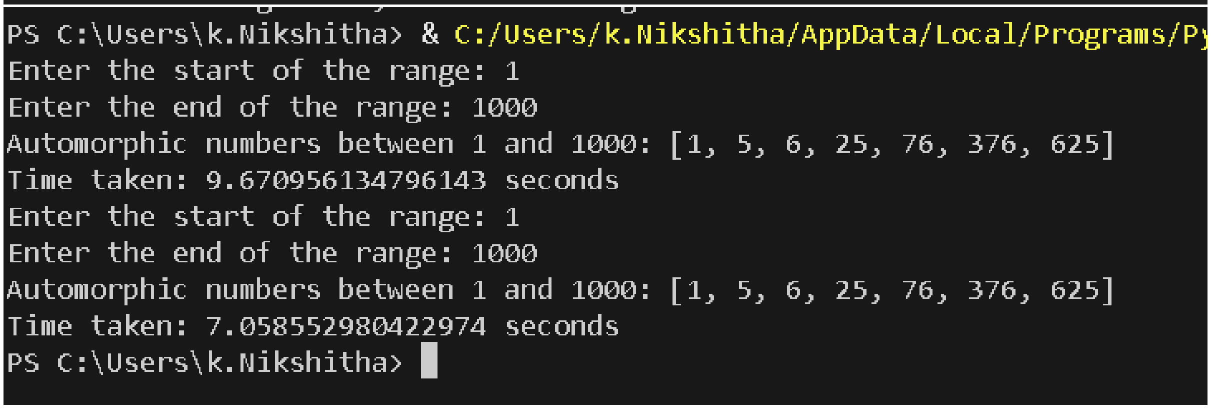


C: > Users > k.Nikshitha > OneDrive > Desktop > New folder > #1.py > ...

```python
#generate all automorphic numbers with in given range using for loop
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 automorphic_numbers_in_range(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 = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
automorphic_numbers = automorphic_numbers_in_range(start_range, end_range)
end_time = t.time()
print(f"Automorphic numbers between {start_range} and {end_range}: {automorphic_numbers}")
print(f"Time taken: {end_time - start_time} seconds")

#generate all automorphic numbers with in given range using while loop
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 automorphic_numbers_in_range(start, end):
    automorphic_numbers = []
    num = start
    while num <= end:
        if is_automorphic(num):
            automorphic_numbers.append(num)
        num += 1
    return automorphic_numbers
start_time = t.time()
```


C: > Users > k.Nikshitha > OneDrive > Desktop > New folder > #1.py > ...

```python
                automorphic_numbers.append(num)
        return automorphic_numbers
start_time = t.time()
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
automorphic_numbers = automorphic_numbers_in_range(start_range, end_range)
end_time = t.time()
print(f"Automorphic numbers between {start_range} and {end_range}: {automorphic_numbers}")
print(f"Time taken: {end_time - start_time} seconds")

#generate all automorphic numbers with in given range using while loop
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 automorphic_numbers_in_range(start, end):
    automorphic_numbers = []
    num = start
    while num <= end:
        if is_automorphic(num):
            automorphic_numbers.append(num)
        num += 1
    return automorphic_numbers
start_time = t.time()
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
automorphic_numbers = automorphic_numbers_in_range(start_range, end_range)
end_time = t.time()
print(f"Automorphic numbers between {start_range} and {end_range}: {automorphic_numbers}")
print(f"Time taken: {end_time - start_time} seconds")
```



```
PS C:\Users\k.Nikshitha> & C:/Users/k.Nikshitha/AppData/Local/Programs/Py
Enter the start of the range: 1
Enter the end of the range: 1000
Automorphic numbers between 1 and 1000: [1, 5, 6, 25, 76, 376, 625]
Time taken: 9.670956134796143 seconds
Enter the start of the range: 1
Enter the end of the range: 1000
Automorphic numbers between 1 and 1000: [1, 5, 6, 25, 76, 376, 625]
Time taken: 7.058552980422974 seconds
PS C:\Users\k.Nikshitha> 
```

2

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:

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

o Analyze correctness and readability.

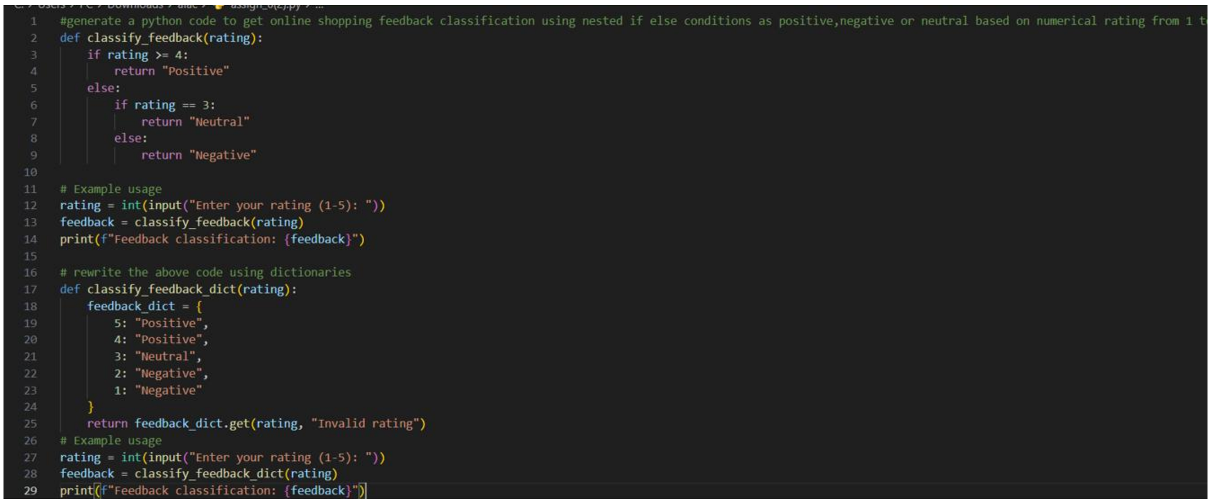
o Ask AI to rewrite using dictionary-based or match-case

structure.

Expected Output #2:

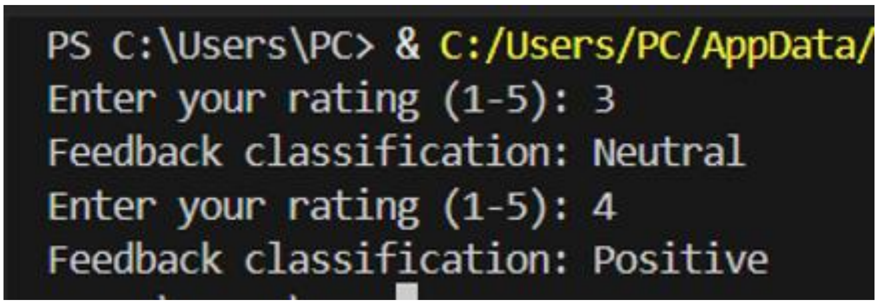
• Feedback classification function with explanation and an alternative

approach.



```python
#generate a python code to get online shopping feedback classification using nested if else conditions as positive,negative or neutral based on numerical rating from 1 t
def classify_feedback(rating):
    if rating >= 4:
        return "Positive"
    else:
        if rating == 3:
            return "Neutral"
        else:
            return "Negative"

# Example usage
rating = int(input("Enter your rating (1-5): "))
feedback = classify_feedback(rating)
print(f"Feedback classification: {feedback}")

# rewrite the above code using dictionaries
def classify_feedback_dict(rating):
    feedback_dict = {
        5: "Positive",
        4: "Positive",
        3: "Neutral",
        2: "Negative",
        1: "Negative"
    }
    return feedback_dict.get(rating, "Invalid rating")
# Example usage
rating = int(input("Enter your rating (1-5): "))
feedback = classify_feedback_dict(rating)
print(f"Feedback classification: {feedback}")
```

OUTPUT :



```
PS C:\Users\PC> & C:/Users/PC/AppData/
Enter your rating (1-5): 3
Feedback classification: Neutral
Enter your rating (1-5): 4
Feedback classification: Positive
```

Explanation :

The dictionary approach is more Pythonic and preferred for mapping-style problems. Both produce identical output for valid inputs (1-5), but the dictionary version handles edge cases better with its default parameter.

3

Task 3: Statistical_operations

Define a function named statistical_operations(tuple_num) that performs the

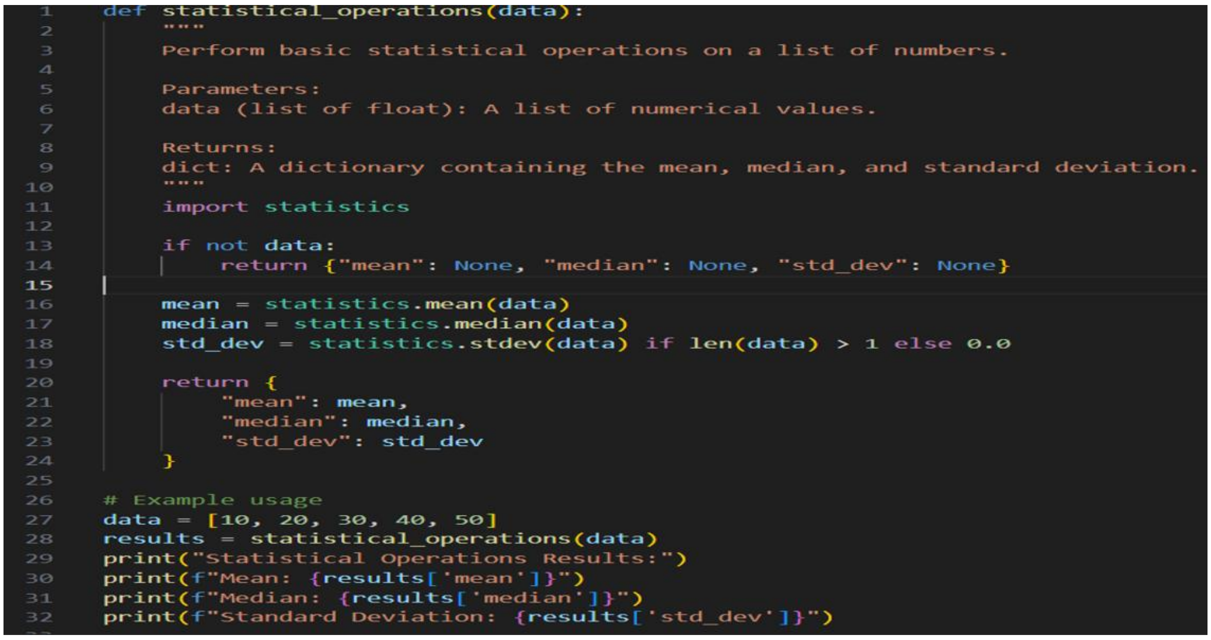
following statistical operations on a tuple of numbers:

• Minimum, Maximum

• Mean, Median, Mode

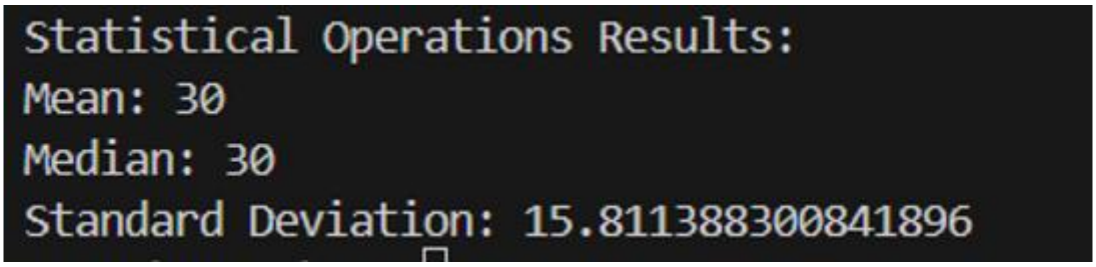
• Variance, Standard Deviation

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
def statistical_operations(data):
    """
    Perform basic statistical operations on a list of numbers.

    Parameters:
    data (list of float): A list of numerical values.

    Returns:
    dict: A dictionary containing the mean, median, and standard deviation.
    """
    import statistics

    if not data:
        return {"mean": None, "median": None, "std_dev": None}

    mean = statistics.mean(data)
    median = statistics.median(data)
    std_dev = statistics.stdev(data) if len(data) > 1 else 0.0

    return {
        "mean": mean,
        "median": median,
        "std_dev": std_dev
    }

# Example usage
data = [10, 20, 30, 40, 50]
results = statistical_operations(data)
print("Statistical Operations Results:")
print(f"Mean: {results['mean']}")
print(f"Median: {results['median']}")
print(f"Standard Deviation: {results['std_dev']}")
```

OUTPUT :



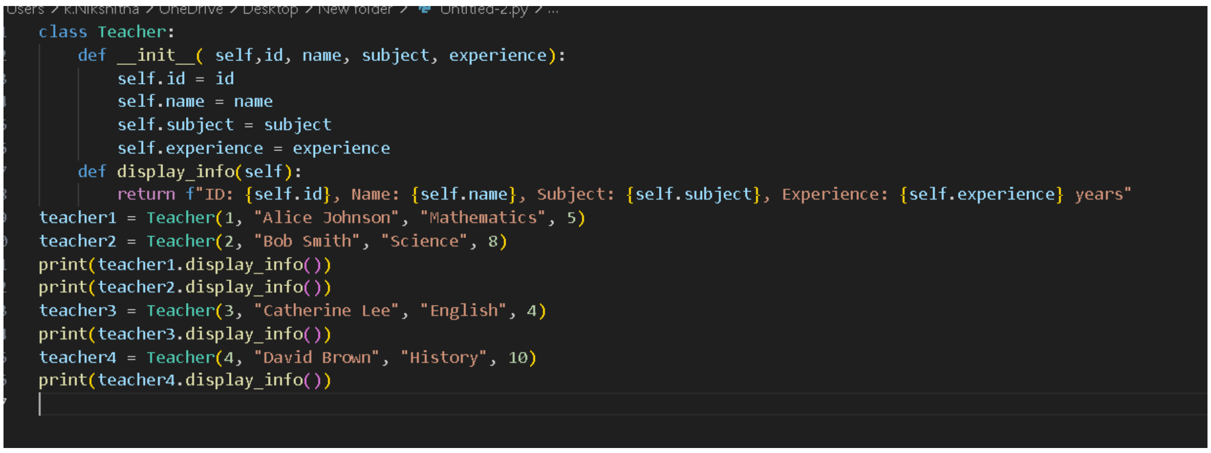
```
Statistical Operations Results:
Mean: 30
Median: 30
Standard Deviation: 15.811388300841896
```

4

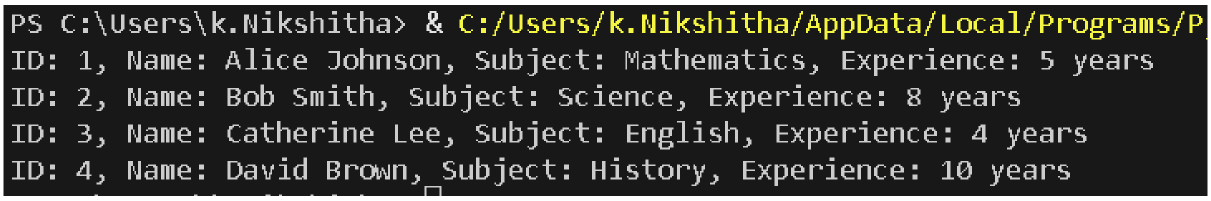
Task 4: Teacher Profile

• 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.



```
Users > k.Nikshitha > OneDrive > Desktop > New folder > Untitled-2.py > ...
class Teacher:
    def __init__( self,id, name, subject, experience):
        self.id = id
        self.name = name
        self.subject = subject
        self.experience = experience
    def display_info(self):
        return f"ID: {self.id}, Name: {self.name}, Subject: {self.subject}, Experience: {self.experience} years"
teacher1 = Teacher(1, "Alice Johnson", "Mathematics", 5)
teacher2 = Teacher(2, "Bob Smith", "Science", 8)
print(teacher1.display_info())
print(teacher2.display_info())
teacher3 = Teacher(3, "Catherine Lee", "English", 4)
print(teacher3.display_info())
teacher4 = Teacher(4, "David Brown", "History", 10)
print(teacher4.display_info())
```



```
PS C:\Users\k.Nikshitha> & C:/Users/k.Nikshitha/AppData/Local/Programs/P
ID: 1, Name: Alice Johnson, Subject: Mathematics, Experience: 5 years
ID: 2, Name: Bob Smith, Subject: Science, Experience: 8 years
ID: 3, Name: Catherine Lee, Subject: English, Experience: 4 years
ID: 4, Name: David Brown, Subject: History, Experience: 10 years
```

5

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

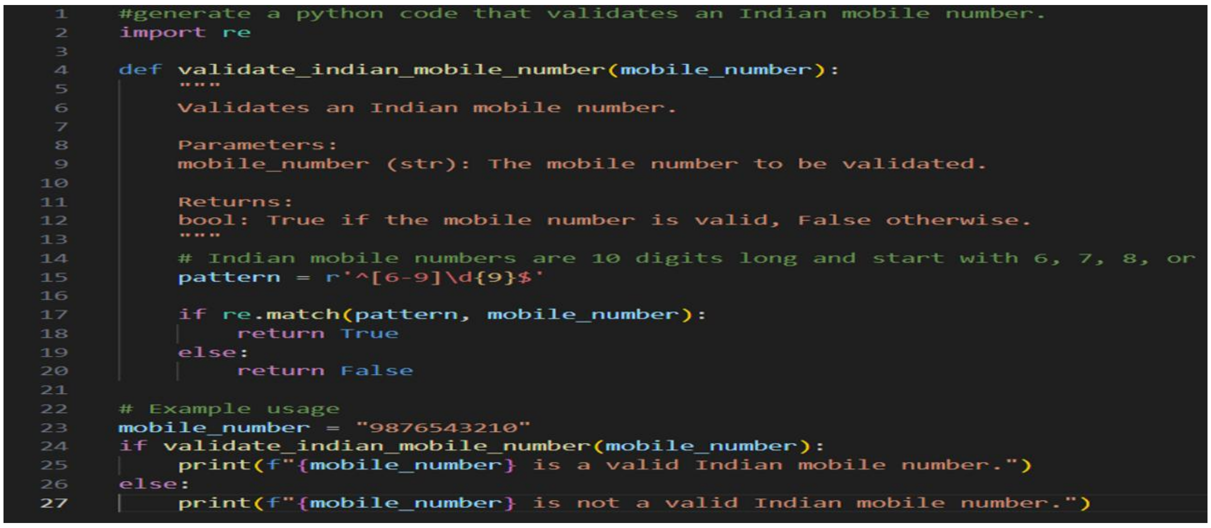
o Contains exactly 10 digits

Expected Output

• A valid Python function that performs all required validations

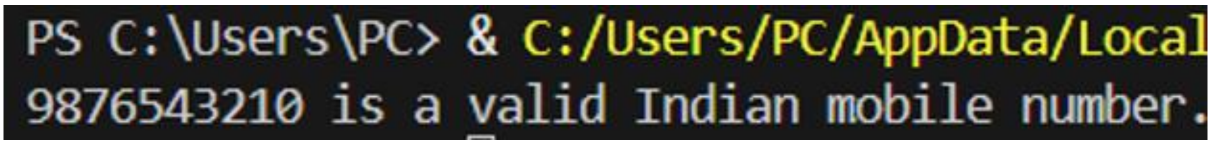
without using any input-output examples in the prompt.

CODE :



```python
#generate a python code that validates an Indian mobile number.
import re

def validate_indian_mobile_number(mobile_number):
    """
    Validates an Indian mobile number.

    Parameters:
    mobile_number (str): The mobile number to be validated.

    Returns:
    bool: True if the mobile number is valid, False otherwise.
    """
    # Indian mobile numbers are 10 digits long and start with 6, 7, 8, or
    pattern = r'^[6-9]\d{9}$'

    if re.match(pattern, mobile_number):
        return True
    else:
        return False

# Example usage
mobile_number = "9876543210"
if validate_indian_mobile_number(mobile_number):
    print(f"{mobile_number} is a valid Indian mobile number.")
else:
    print(f"{mobile_number} is not a valid Indian mobile number.")
```

OUTPUT :



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local
9876543210 is a valid Indian mobile number.
```

6

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.

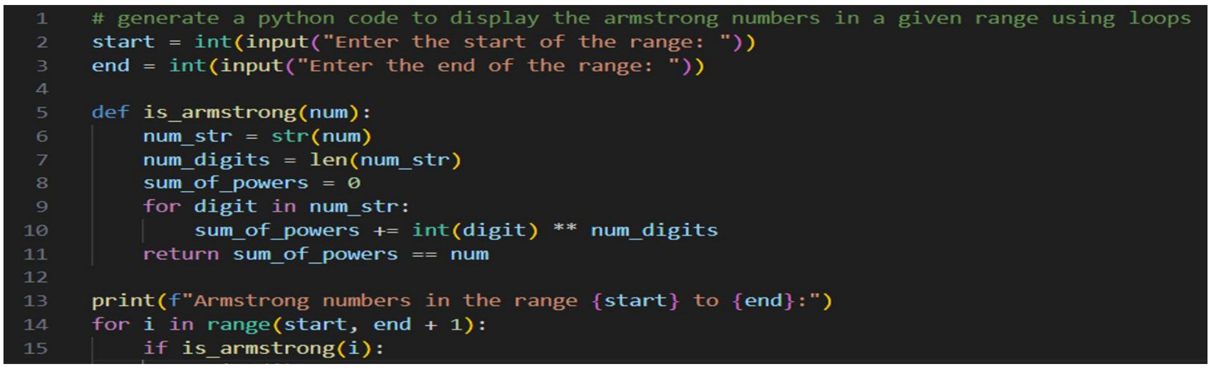
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).

• Ask AI to regenerate an optimized version (using list comprehensions).

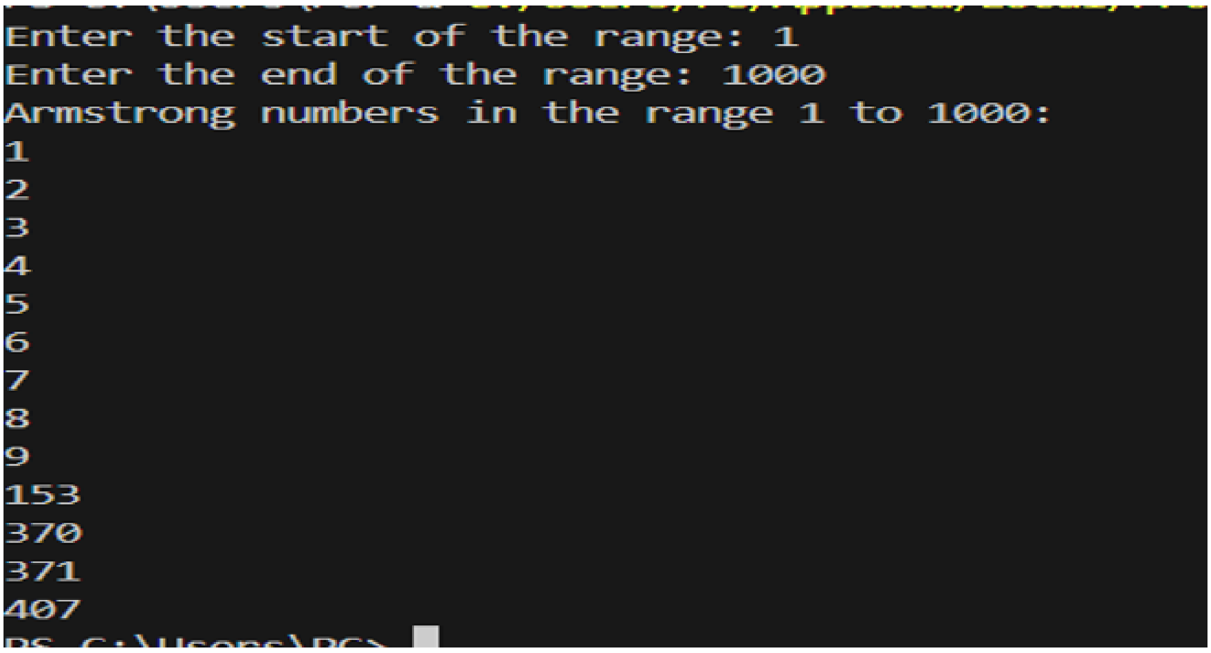
Expected Output #7:

• Python program listing Armstrong numbers in the range.

• Optimized version with explanation.



```python
# generate a python code to display the armstrong numbers in a given range using loops
start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))

def is_armstrong(num):
    num_str = str(num)
    num_digits = len(num_str)
    sum_of_powers = 0
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits
    return sum_of_powers == num

print(f"Armstrong numbers in the range {start} to {end}:")
for i in range(start, end + 1):
    if is_armstrong(i):
```



```
Enter the start of the range: 1
Enter the end of the range: 1000
Armstrong numbers in the range 1 to 1000:
1
2
3
4
5
6
7
8
9
153
370
371
407
```

7

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…).

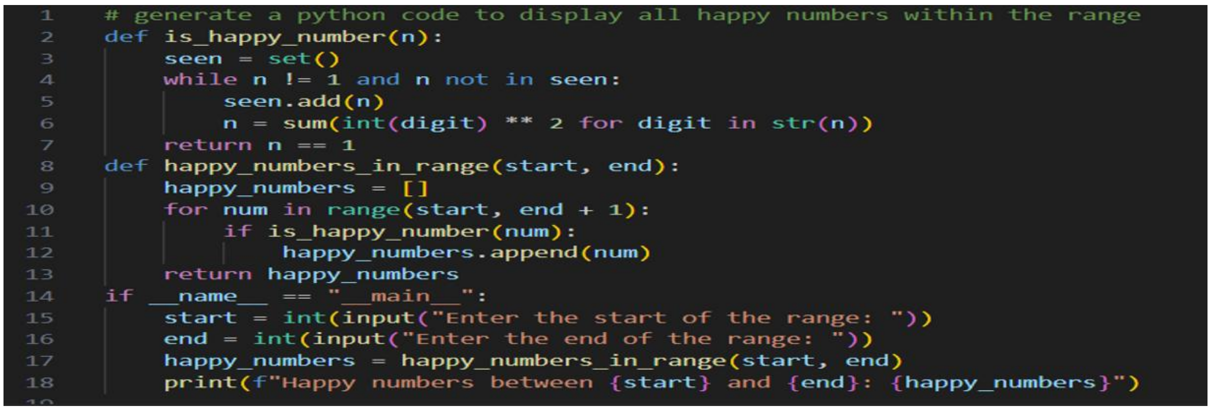
• Ask AI to regenerate an optimized version (e.g., by using a set to detect

cycles instead of infinite loops).

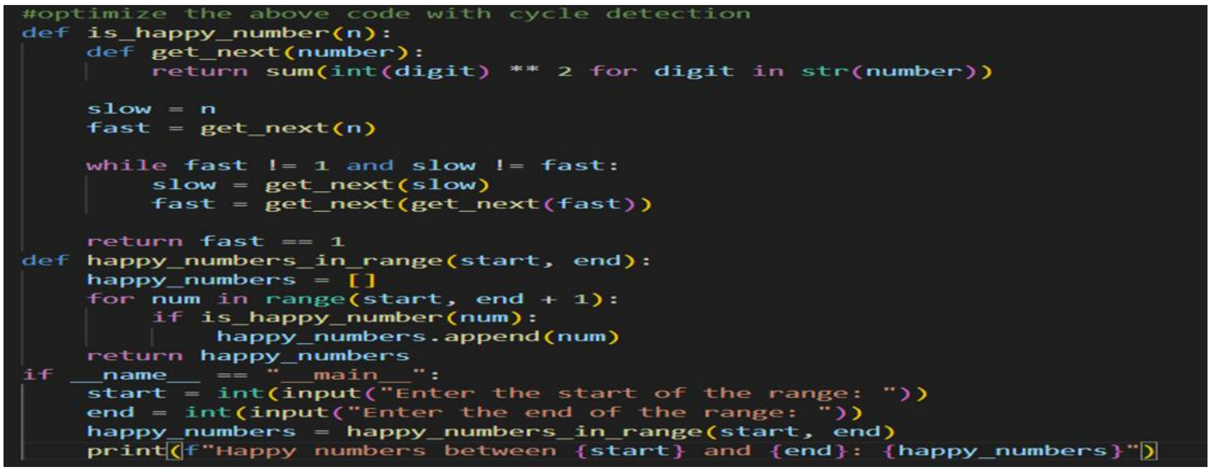
Expected Output #8:

• Python program that prints all Happy Numbers within a range.

• Optimized version using cycle detection with explanation.

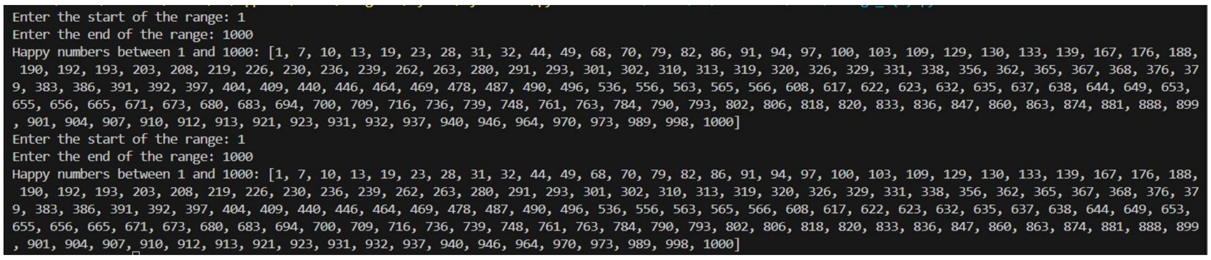


```python
# generate a python code to display all happy numbers within the range
def is_happy_number(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit) ** 2 for digit in str(n))
    return n == 1
def happy_numbers_in_range(start, end):
    happy_numbers = []
    for num in range(start, end + 1):
        if is_happy_number(num):
            happy_numbers.append(num)
    return happy_numbers
if __name__ == "__main__":
    start = int(input("Enter the start of the range: "))
    end = int(input("Enter the end of the range: "))
    happy_numbers = happy_numbers_in_range(start, end)
    print(f"Happy numbers between {start} and {end}: {happy_numbers}")
```



```python
#optimize the above code with cycle detection
def is_happy_number(n):
    def get_next(number):
        return sum(int(digit) ** 2 for digit in str(number))

    slow = n
    fast = get_next(n)

    while fast != 1 and slow != fast:
        slow = get_next(slow)
        fast = get_next(get_next(fast))

    return fast == 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
if __name__ == "__main__":
    start = int(input("Enter the start of the range: "))
    end = int(input("Enter the end of the range: "))
    happy_numbers = happy_numbers_in_range(start, end)
    print(f"Happy numbers between {start} and {end}: {happy_numbers}")
```

OUTPUT :



```
Enter the start of the range: 1
Enter the end of the range: 1000
Happy numbers between 1 and 1000: [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, 368, 376, 37
9, 383, 386, 391, 392, 397, 404, 409, 440, 446, 464, 469, 478, 487, 490, 496, 536, 556, 563, 565, 566, 608, 617, 622, 623, 632, 635, 637, 638, 644, 649, 653,
655, 656, 665, 671, 673, 680, 683, 694, 700, 709, 716, 736, 739, 748, 761, 763, 784, 790, 793, 802, 806, 818, 820, 833, 836, 847, 860, 863, 874, 881, 888, 899
, 901, 904, 907, 910, 912, 913, 921, 923, 931, 932, 937, 940, 946, 964, 970, 973, 989, 998, 1000]
Enter the start of the range: 1
Enter the end of the range: 1000
Happy numbers between 1 and 1000: [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, 368, 376, 37
9, 383, 386, 391, 392, 397, 404, 409, 440, 446, 464, 469, 478, 487, 490, 496, 536, 556, 563, 565, 566, 608, 617, 622, 623, 632, 635, 637, 638, 644, 649, 653,
655, 656, 665, 671, 673, 680, 683, 694, 700, 709, 716, 736, 739, 748, 761, 763, 784, 790, 793, 802, 806, 818, 820, 833, 836, 847, 860, 863, 874, 881, 888, 899
, 901, 904, 907, 910, 912, 913, 921, 923, 931, 932, 937, 940, 946, 964, 970, 973, 989, 998, 1000]
```

Explanation :

The first approach uses O(k) space where k is the number of unique values encountered before determining happiness. The second approach uses O(1) constant space but performs more function calls. For typical ranges, both are fast—the optimization shines with very large numbers or massive ranges where memory becomes a bottleneck.

8 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.

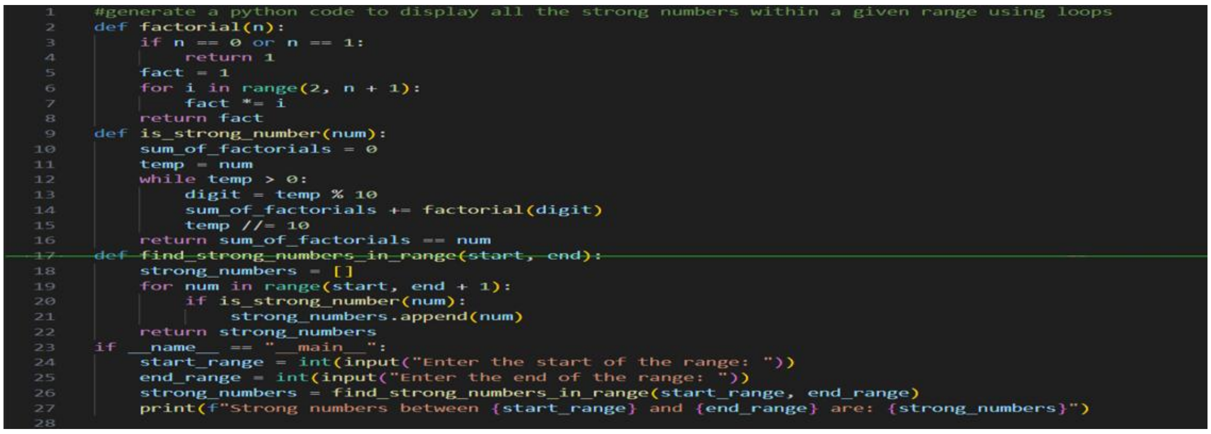
• Validate with examples (1, 2, 145).

• Ask AI to regenerate an optimized version (precompute digit factorials).

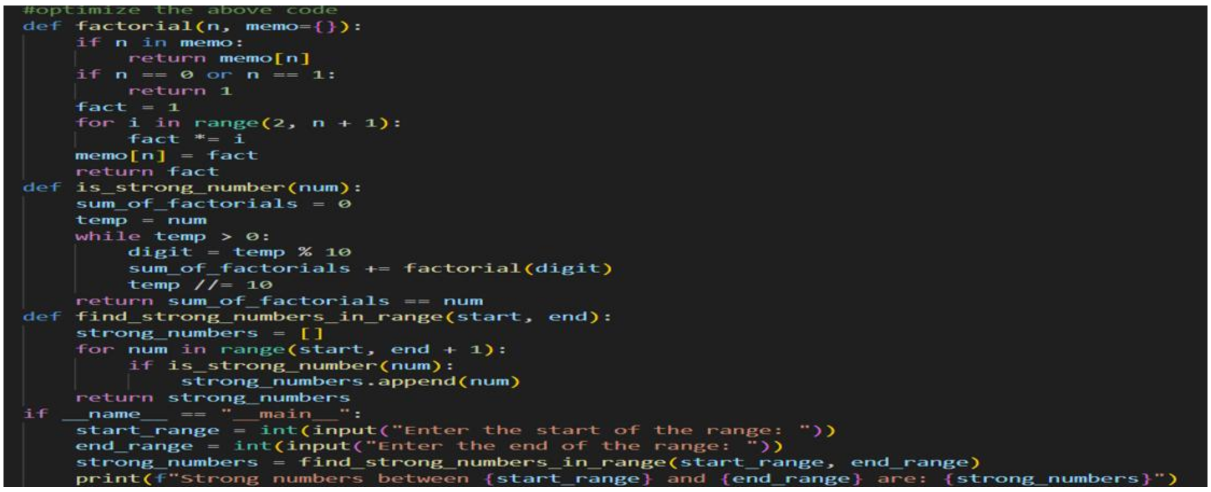
Expected Output #9:

• Python program that lists Strong Numbers.

• Optimized version with explanation.

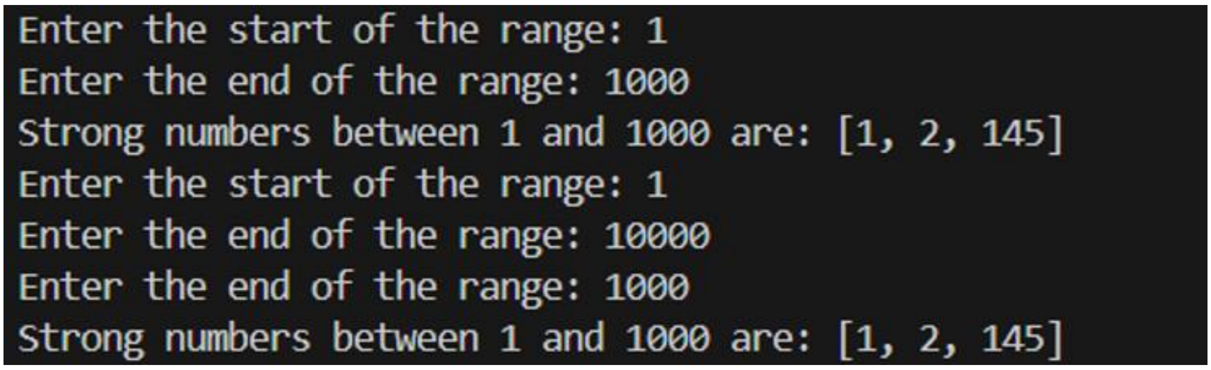


```python
#generate a python code to display all the strong numbers within a given range using loops
def factorial(n):
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(2, n + 1):
        fact *= i
    return fact
def is_strong_number(num):
    sum_of_factorials = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum_of_factorials += factorial(digit)
        temp //= 10
    return sum_of_factorials == num
def find_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
if __name__ == "__main__":
    start_range = int(input("Enter the start of the range: "))
    end_range = int(input("Enter the end of the range: "))
    strong_numbers = find_strong_numbers_in_range(start_range, end_range)
    print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers}")
```



```python
#optimize the above code
def factorial(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(2, n + 1):
        fact *= i
    memo[n] = fact
    return fact
def is_strong_number(num):
    sum_of_factorials = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum_of_factorials += factorial(digit)
        temp //= 10
    return sum_of_factorials == num
def find_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
if __name__ == "__main__":
    start_range = int(input("Enter the start of the range: "))
    end_range = int(input("Enter the end of the range: "))
    strong_numbers = find_strong_numbers_in_range(start_range, end_range)
    print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers}")
```

OUTPUT :



```
Enter the start of the range: 1
Enter the end of the range: 1000
Strong numbers between 1 and 1000 are: [1, 2, 145]
Enter the start of the range: 1
Enter the end of the range: 10000
Enter the end of the range: 1000
Strong numbers between 1 and 1000 are: [1, 2, 145]
```

Explanation :

The optimization provides dramatic speedup for large ranges. Without memoization, a range like 1-10,000 recalculates 5! thousands of times. With memoization, each factorial computes only once, delivering consistent O(1) lookup for subsequent calls.

9

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

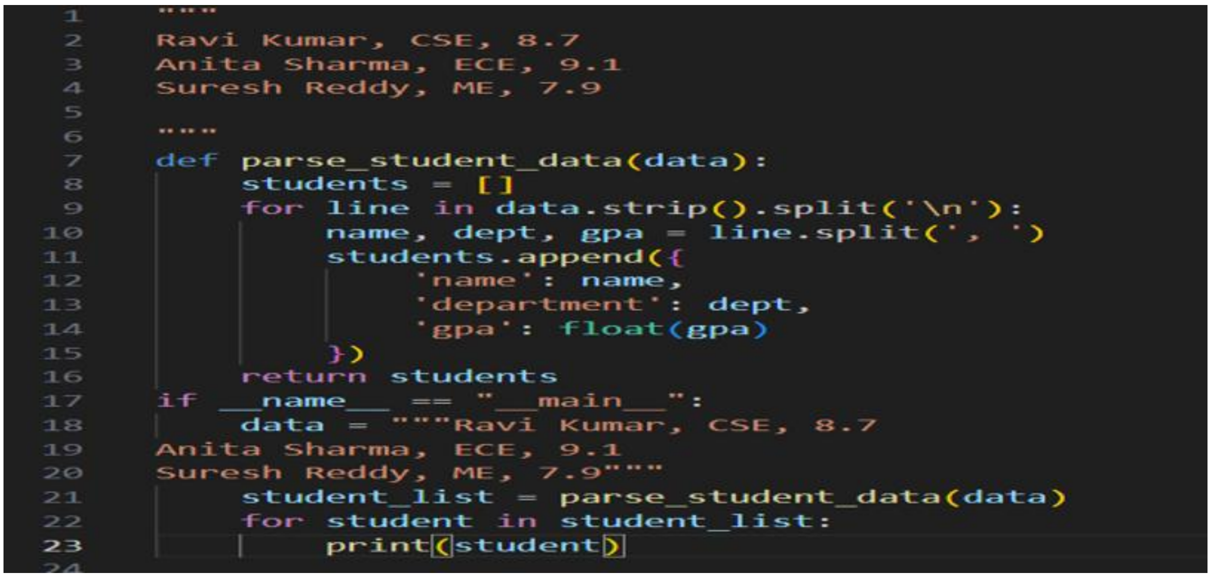
o Branch

o SGPA

Expected Output

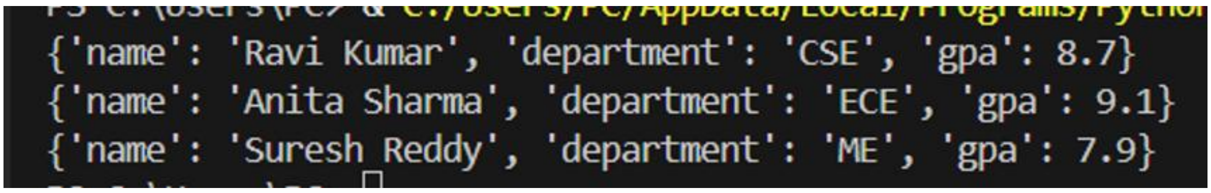
A reusable Python function that correctly navigates and extracts values

from nested dictionaries based on the provided examples



```python
"""
Ravi Kumar, CSE, 8.7
Anita Sharma, ECE, 9.1
Suresh Reddy, ME, 7.9

"""
def parse_student_data(data):
    students = []
    for line in data.strip().split('\n'):
        name, dept, gpa = line.split(', ')
        students.append({
            'name': name,
            'department': dept,
            'gpa': float(gpa)
        })
    return students
if __name__ == "__main__":
    data = """Ravi Kumar, CSE, 8.7
Anita Sharma, ECE, 9.1
Suresh Reddy, ME, 7.9"""
    student_list = parse_student_data(data)
    for student in student_list:
        print(student)
```

OUTPUT :



```
{'name': 'Ravi Kumar', 'department': 'CSE', 'gpa': 8.7}
{'name': 'Anita Sharma', 'department': 'ECE', 'gpa': 9.1}
{'name': 'Suresh Reddy', 'department': 'ME', 'gpa': 7.9}
```

10

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).

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…).

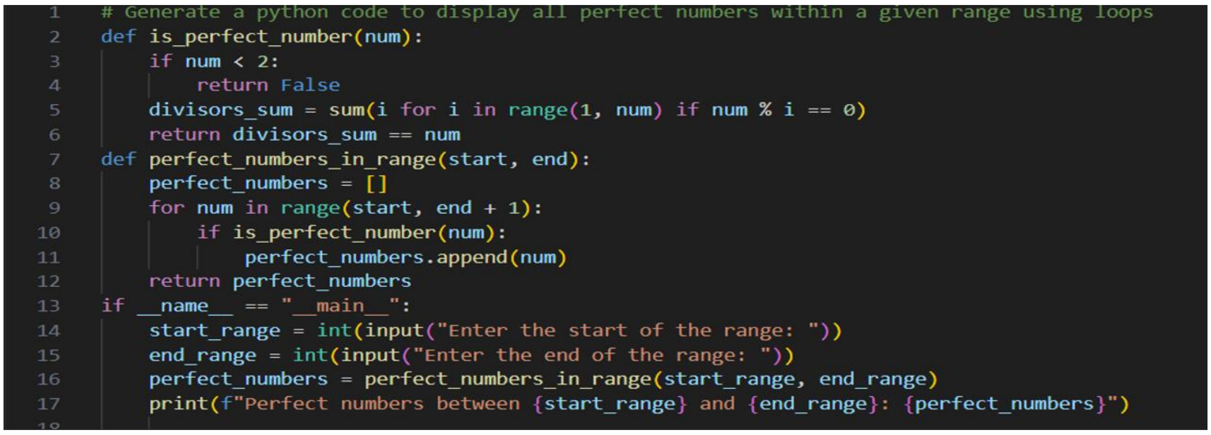
• Ask AI to regenerate an optimized version (using divisor check only up to

√n).

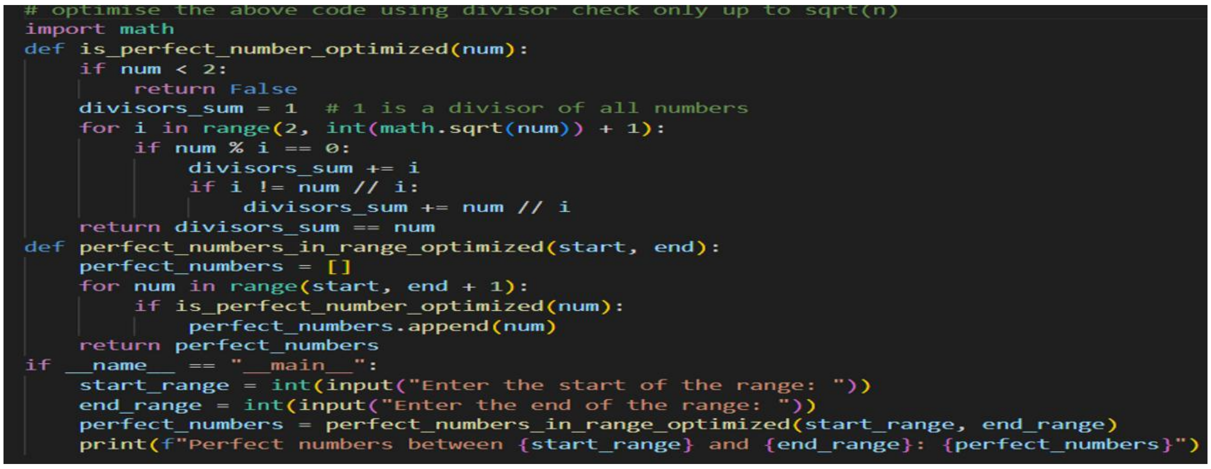
Expected Output #12:

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

• Optimized version with explanation.

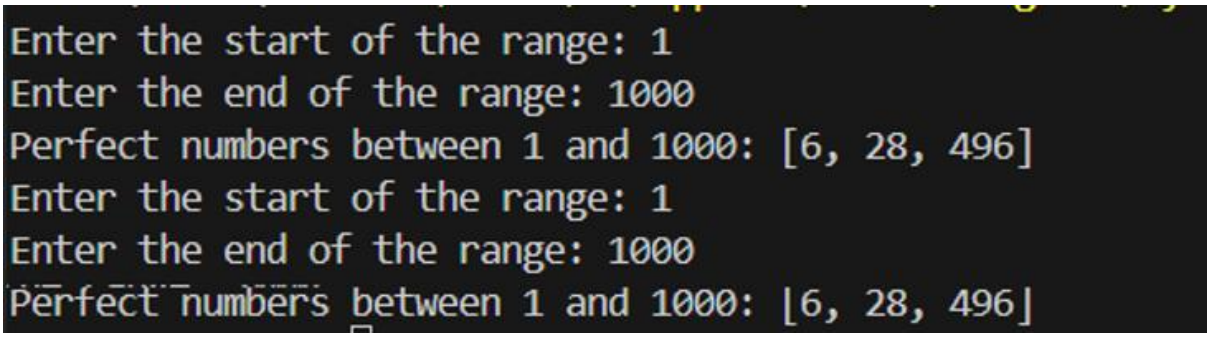


```python
# Generate a python code to display all perfect numbers within a given range using loops
def is_perfect_number(num):
    if num < 2:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum == num
def perfect_numbers_in_range(start, end):
    perfect_numbers = []
    for num in range(start, end + 1):
        if is_perfect_number(num):
            perfect_numbers.append(num)
    return perfect_numbers
if __name__ == "__main__":
    start_range = int(input("Enter the start of the range: "))
    end_range = int(input("Enter the end of the range: "))
    perfect_numbers = perfect_numbers_in_range(start_range, end_range)
    print(f"Perfect numbers between {start_range} and {end_range}: {perfect_numbers}")
```



```python
# optimise the above code using divisor check only up to sqrt(n)
import math
def is_perfect_number_optimized(num):
    if num < 2:
        return False
    divisors_sum = 1  # 1 is a divisor of all numbers
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            divisors_sum += i
            if i != num // i:
                divisors_sum += num // i
    return divisors_sum == num
def perfect_numbers_in_range_optimized(start, end):
    perfect_numbers = []
    for num in range(start, end + 1):
        if is_perfect_number_optimized(num):
            perfect_numbers.append(num)
    return perfect_numbers
if __name__ == "__main__":
    start_range = int(input("Enter the start of the range: "))
    end_range = int(input("Enter the end of the range: "))
    perfect_numbers = perfect_numbers_in_range_optimized(start_range, end_range)
    print(f"Perfect numbers between {start_range} and {end_range}: {perfect_numbers}")
```

OUTPUT :



```
Enter the start of the range: 1
Enter the end of the range: 1000
Perfect numbers between 1 and 1000: [6, 28, 496]
Enter the start of the range: 1
Enter the end of the range: 1000
Perfect numbers between 1 and 1000: [6, 28, 496]
```