ASSIGNMENT-6.3

Name: k.prudhvi raj

Batch:24

Ht.No:2303A51963

Task Description #1 (Loops – Automorphic Numbers in a Range)

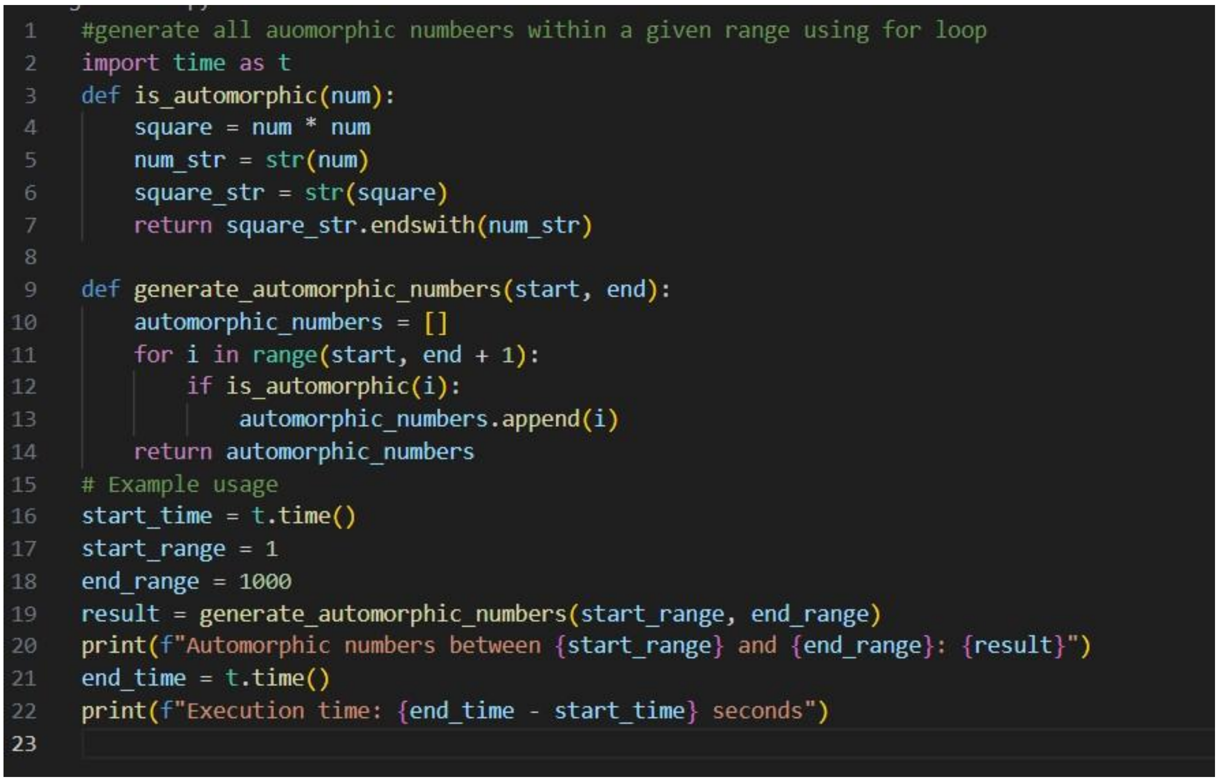
• Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

• 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.

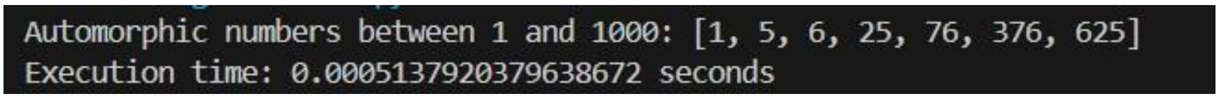
Expected Output #1:

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



```python
#generate all auomorphic numbeers within a 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 generate_automorphic_numbers(start, end):
    automorphic_numbers = []
    for i in range(start, end + 1):
        if is_automorphic(i):
            automorphic_numbers.append(i)
    return automorphic_numbers
# Example usage
start_time = t.time()
start_range = 1
end_range = 1000
result = generate_automorphic_numbers(start_range, end_range)
print(f"Automorphic numbers between {start_range} and {end_range}: {result}")
end_time = t.time()
print(f"Execution time: {end_time - start_time} seconds")
```



```
Automorphic numbers between 1 and 1000: [1, 5, 6, 25, 76, 376, 625]
Execution time: 0.0005137920379638672 seconds
```

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

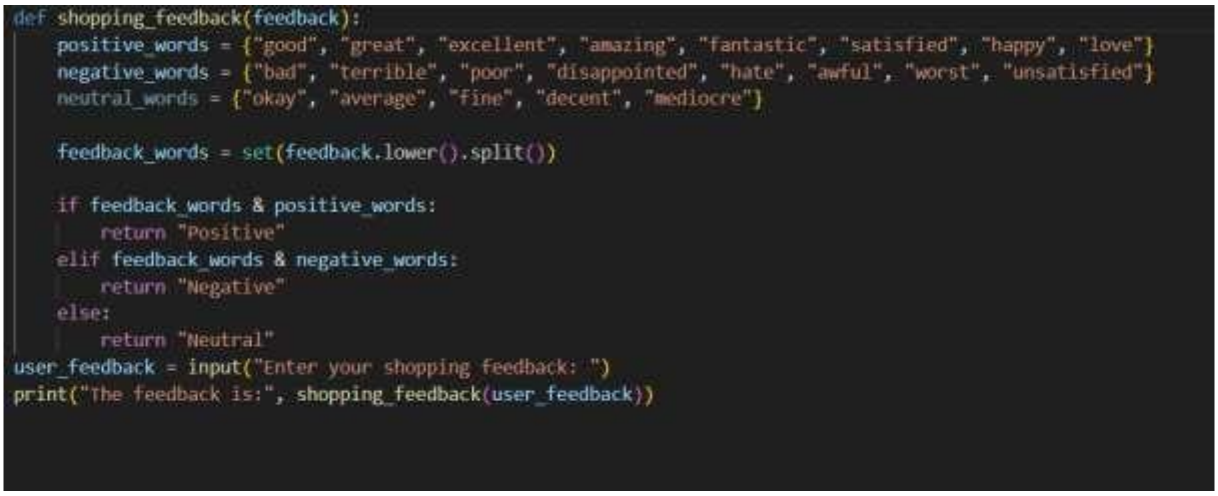
Analyze correctness and readability.

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

Expected Output #2:

• Feedback classification function with explanation and an alternative

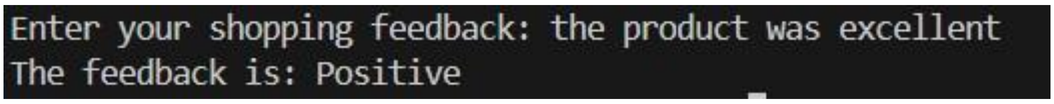
approach.



```python
def shopping_feedback(feedback):
    positive_words = {"good", "great", "excellent", "amazing", "fantastic", "satisfied", "happy", "love"}
    negative_words = {"bad", "terrible", "poor", "disappointed", "hate", "awful", "worst", "unsatisfied"}
    neutral_words = {"okay", "average", "fine", "decent", "mediocre"}

    feedback_words = set(feedback.lower().split())

    if feedback_words & positive_words:
        return "Positive"
    elif feedback_words & negative_words:
        return "Negative"
    else:
        return "Neutral"
user_feedback = input("Enter your shopping feedback: ")
print("The feedback is:", shopping_feedback(user_feedback))
```



```
Enter your shopping feedback: the product was excellent
The feedback is: Positive
```

Task 3: Statistical_operations

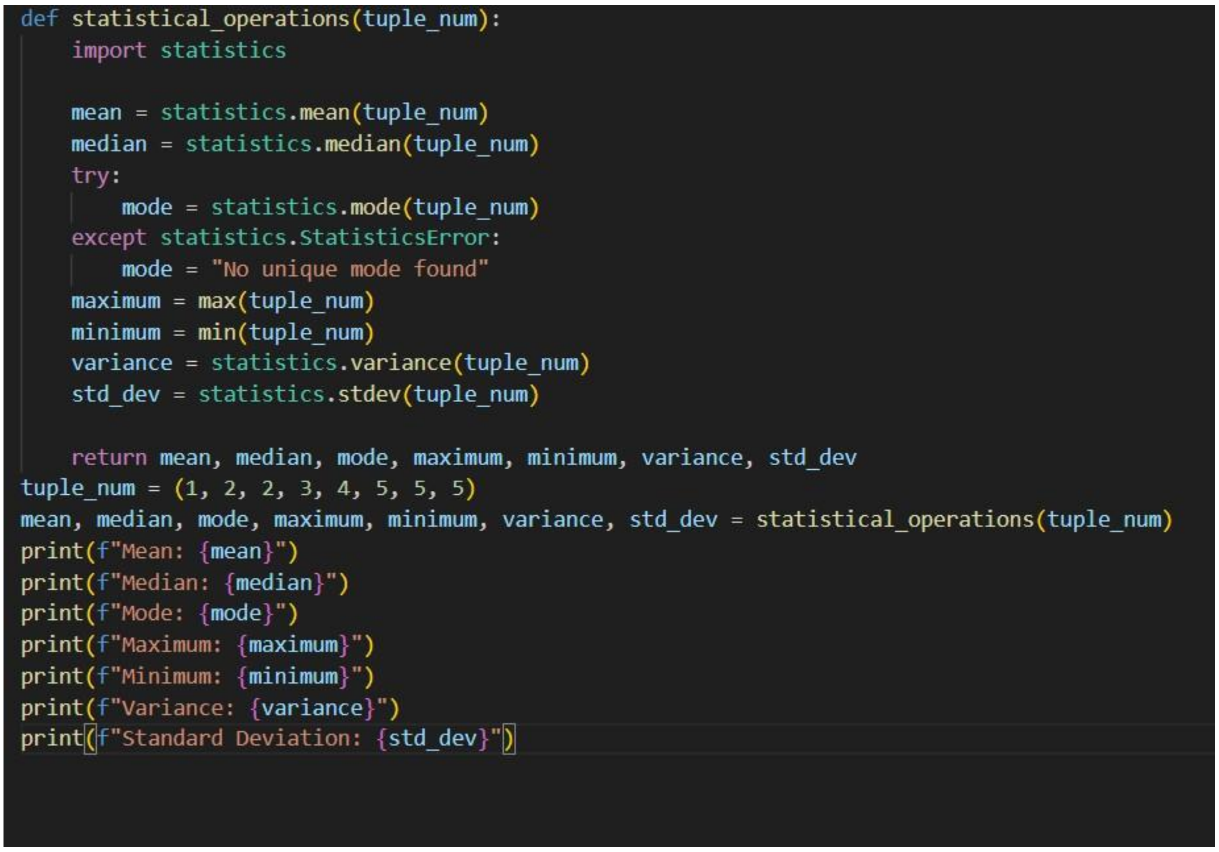
Define a function named statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers:

• Minimum, Maximum

• Mean, Median, Mode

• Variance, Standard Deviation

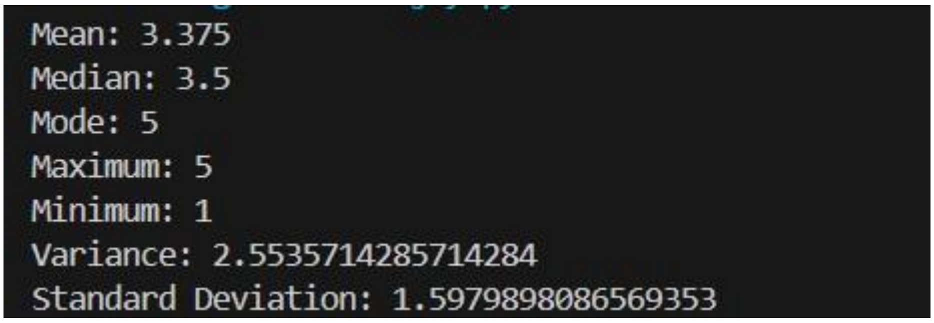
While writing the function, observe the code suggestions provided by GitHub Copilot.Make decisions to accept, reject, or modify the suggestions based on their relevance and correctness



```python
def statistical_operations(tuple_num):
    import statistics

    mean = statistics.mean(tuple_num)
    median = statistics.median(tuple_num)
    try:
        mode = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        mode = "No unique mode found"
    maximum = max(tuple_num)
    minimum = min(tuple_num)
    variance = statistics.variance(tuple_num)
    std_dev = statistics.stdev(tuple_num)

    return mean, median, mode, maximum, minimum, variance, std_dev
tuple_num = (1, 2, 2, 3, 4, 5, 5, 5)
mean, median, mode, maximum, minimum, variance, std_dev = statistical_operations(tuple_num)
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")
print(f"Maximum: {maximum}")
print(f"Minimum: {minimum}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_dev}")
```

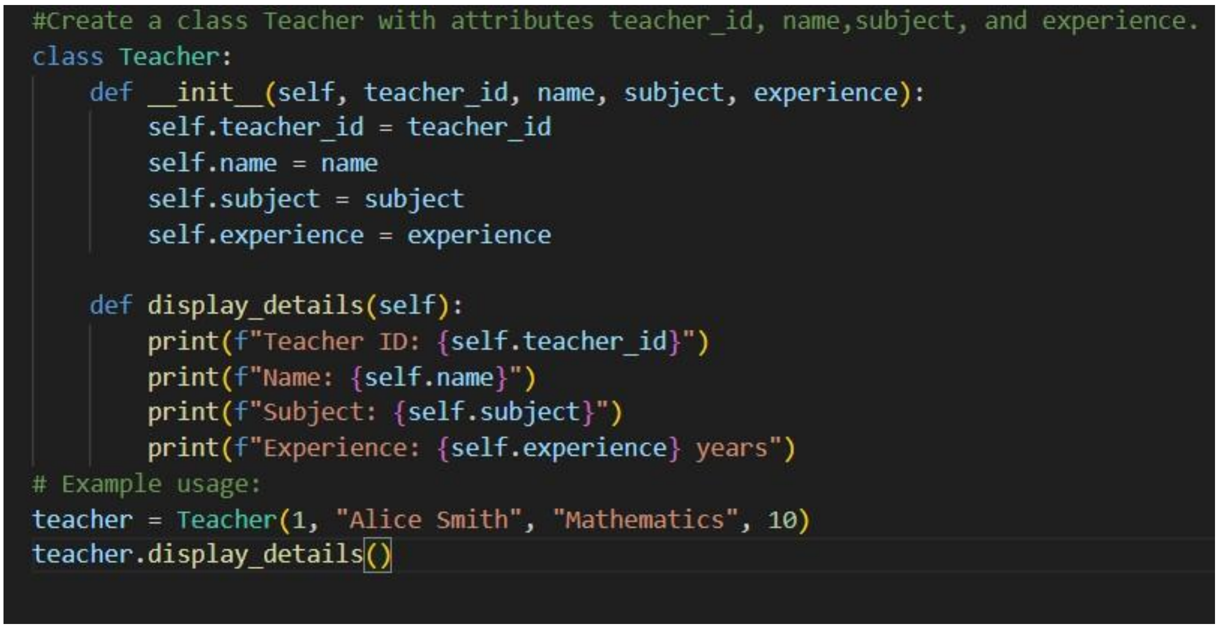


```
Mean: 3.375
Median: 3.5
Mode: 5
Maximum: 5
Minimum: 1
Variance: 2.5535714285714284
Standard Deviation: 1.5979898086569353
```

Task 4: Teacher Profile

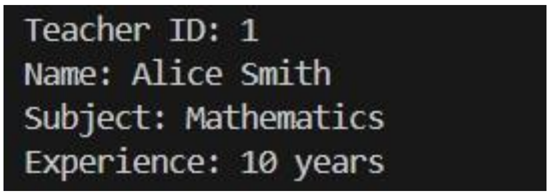
• Prompt: Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.

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



```python
#Create a class Teacher with attributes teacher_id, name,subject, and experience.
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()
```



```
Teacher ID: 1
Name: Alice Smith
Subject: Mathematics
Experience: 10 years
```

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

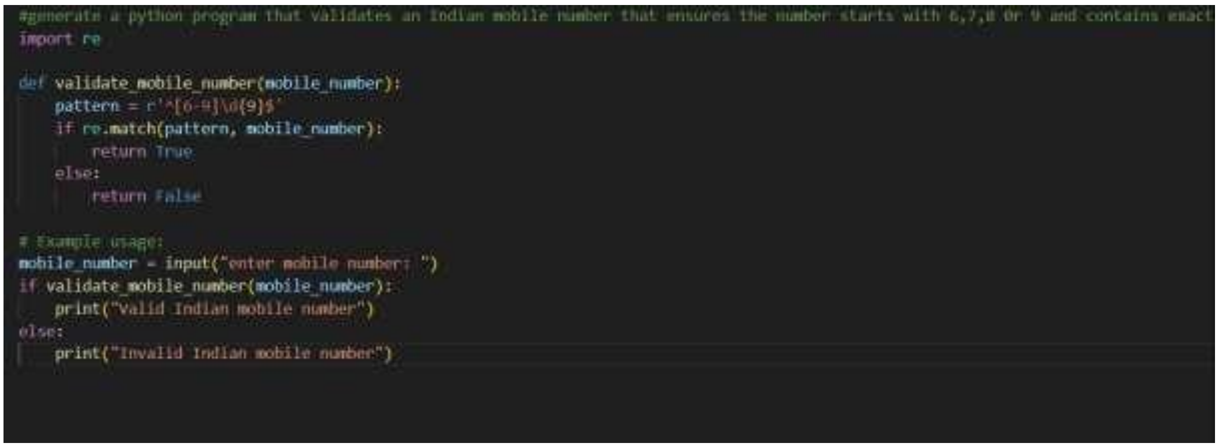
• The function must ensure the mobile number:

o Starts with 6, 7, 8, or 9 o

Contains exactly 10 digits

Expected Output

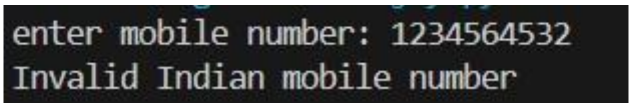
• A valid Python function that performs all required validations without using
  any input-output examples in the prompt.



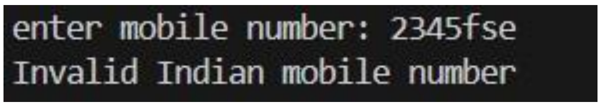
```python
#generate a python program that validates an Indian mobile number that ensures the number starts with 6,7,8 0r 9 and contains exact
import re

def validate_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 mobile number: ")
if validate_mobile_number(mobile_number):
    print("Valid Indian mobile number")
else:
    print("Invalid Indian mobile number")
```



```
enter mobile number: 1234564532
Invalid Indian mobile number
```



```
enter mobile number: 2345fse
Invalid Indian mobile number
```

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.

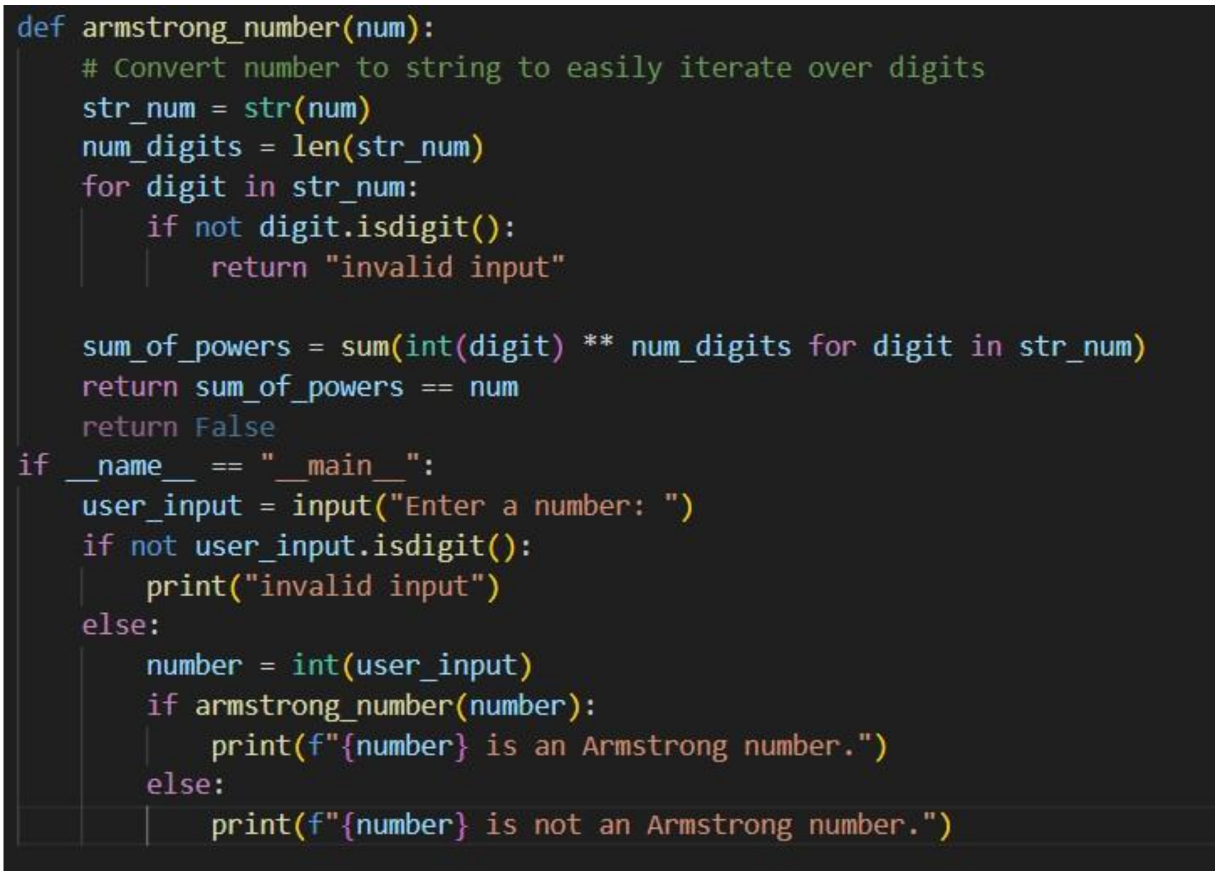
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).

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

Expected Output #7:

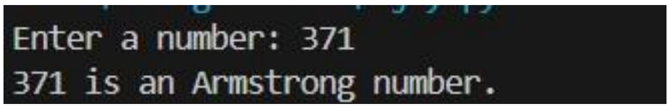
• Python program listing Armstrong numbers in the range.

• Optimized version with explanation.



```python
def armstrong_number(num):
    # Convert number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)
    for digit in str_num:
        if not digit.isdigit():
            return "invalid input"

    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    return sum_of_powers == num
    return False
if __name__ == "__main__":
    user_input = input("Enter a number: ")
    if not user_input.isdigit():
        print("invalid input")
    else:
        number = int(user_input)
        if armstrong_number(number):
            print(f"{number} is an Armstrong number.")
        else:
            print(f"{number} is not an Armstrong number.")
```



```
Enter a number: 371
371 is an Armstrong number.
```

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).

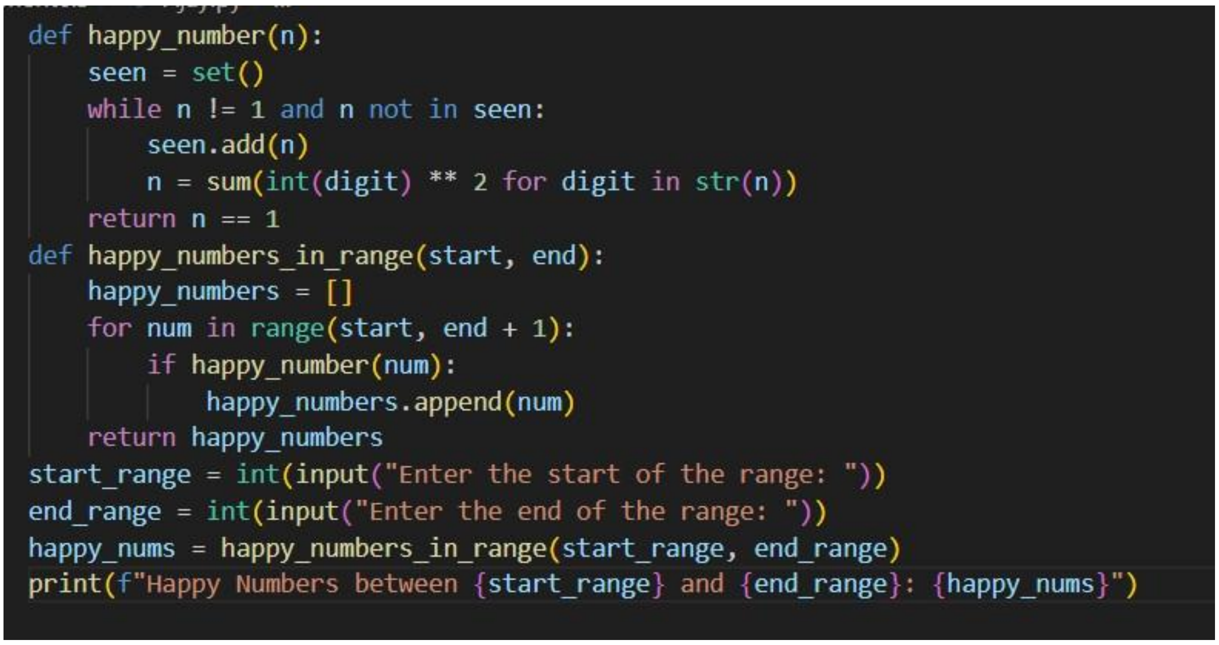
• Validate correctness by checking known Happy Numbers (e.g., 1, 7, 10, 13, 19, 23, 28…).

• Ask AI to regenerate an optimized version (e.g., by using a set to detect cycles instead of infinite loops).

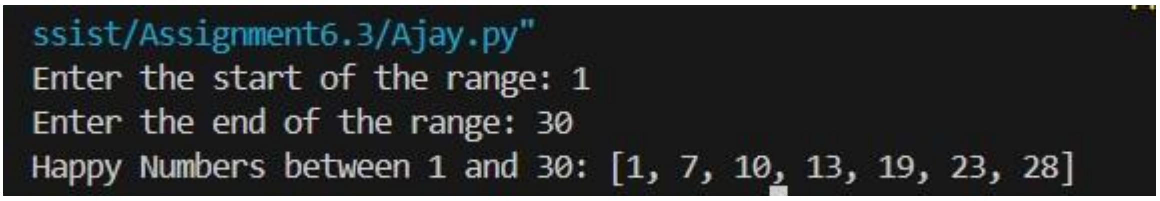
Expected Output #8:

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

• Optimized version using cycle detection with explanation.



```python
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 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_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
happy_nums = happy_numbers_in_range(start_range, end_range)
print(f"Happy Numbers between {start_range} and {end_range}: {happy_nums}")
```



```
ssist/Assignment6.3/Ajay.py"
Enter the start of the range: 1
Enter the end of the range: 30
Happy Numbers between 1 and 30: [1, 7, 10, 13, 19, 23, 28]
```

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.

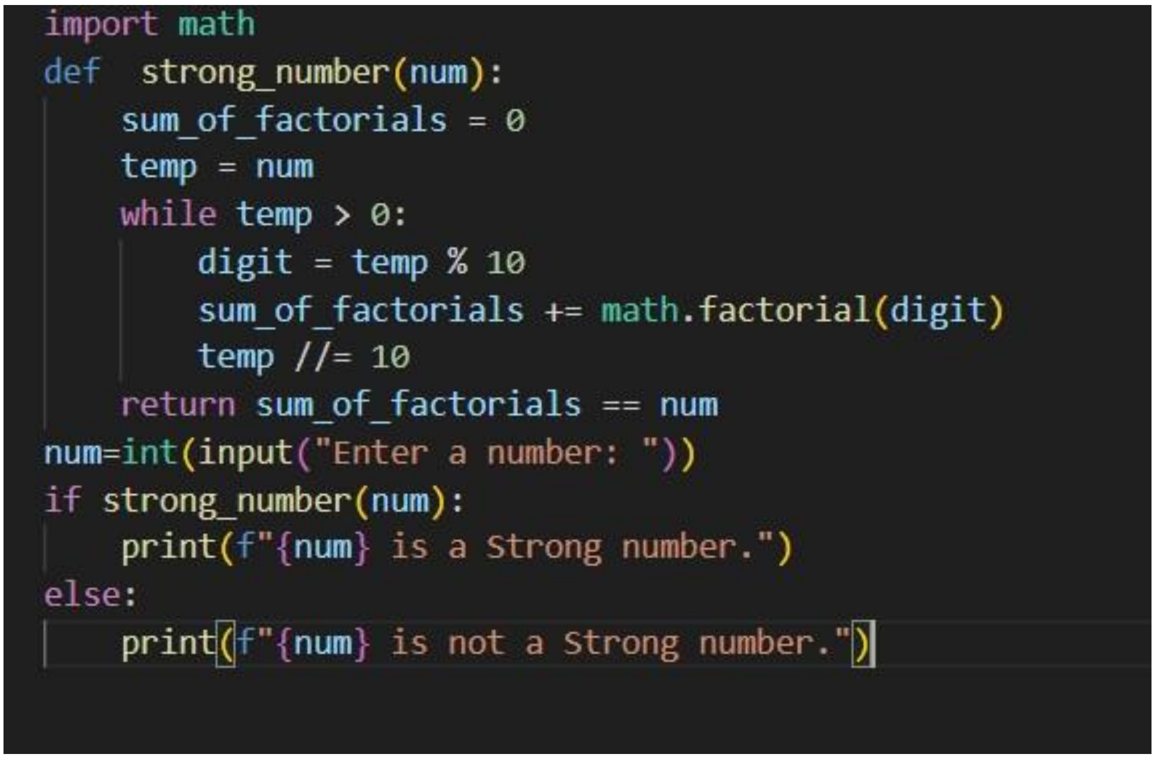
• Validate with examples (1, 2, 145).

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

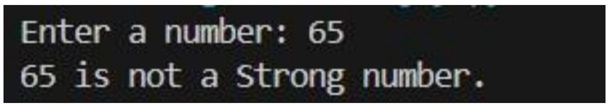
Expected Output #9:

• Python program that lists Strong Numbers.

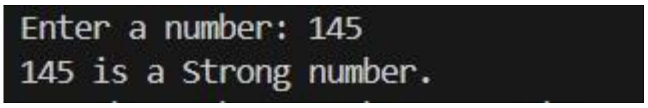
• Optimized version with explanation.



```python
import math
def  strong_number(num):
    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
num=int(input("Enter a number: "))
if strong_number(num):
    print(f"{num} is a Strong number.")
else:
    print(f"{num} is not a Strong number.")
```



```
Enter a number: 65
65 is not a Strong number.
```



```
Enter a number: 145
145 is a Strong number.
```

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

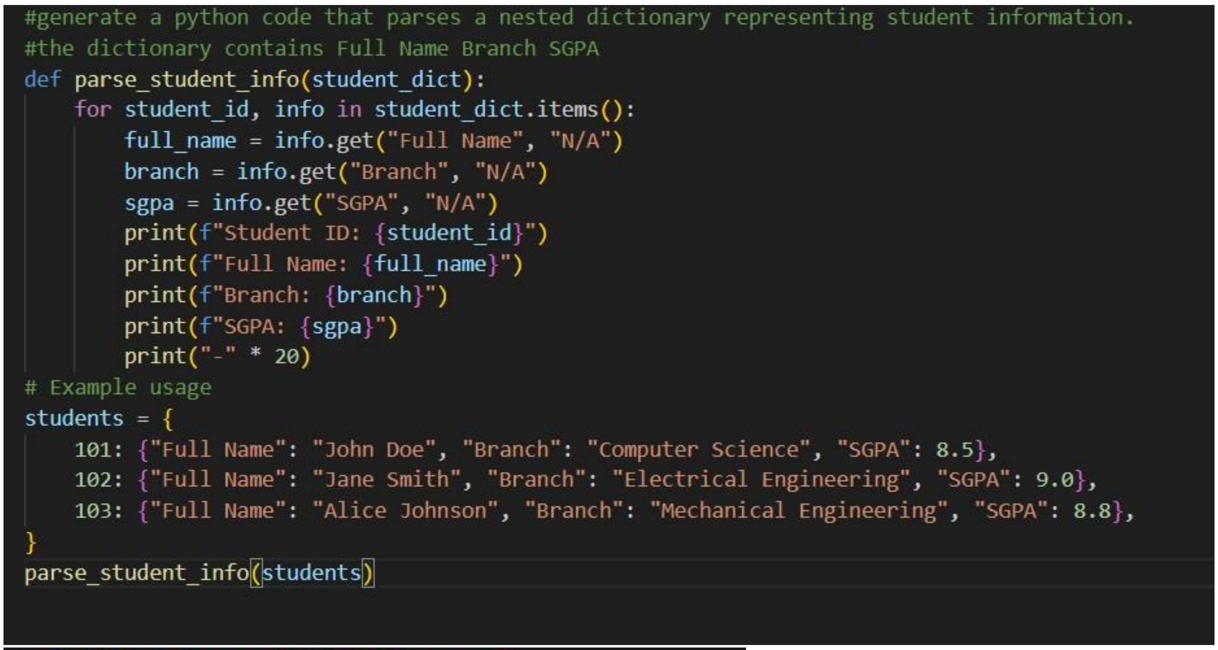
o Branch o

SGPA

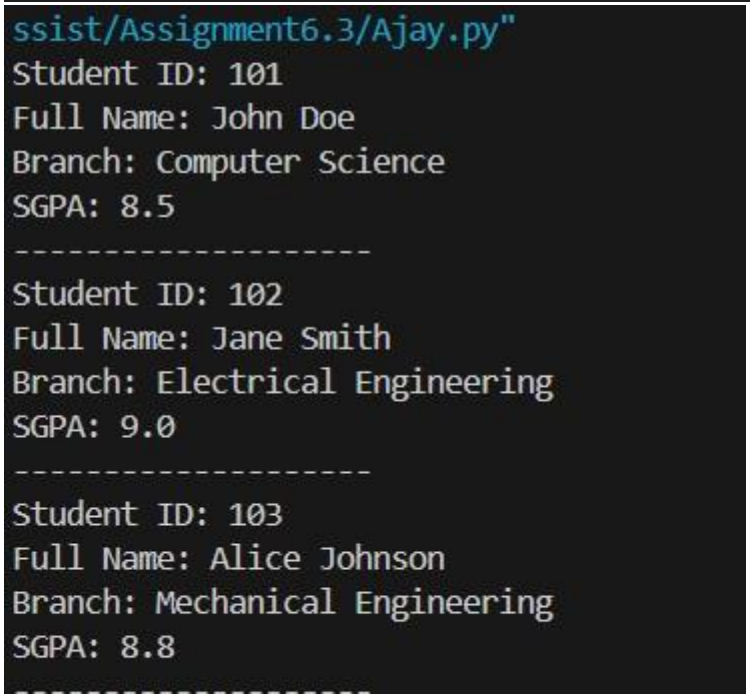
Expected Output

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)
```



```
ssist/Assignment6.3/Ajay.py"
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
--------------------
```

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…).

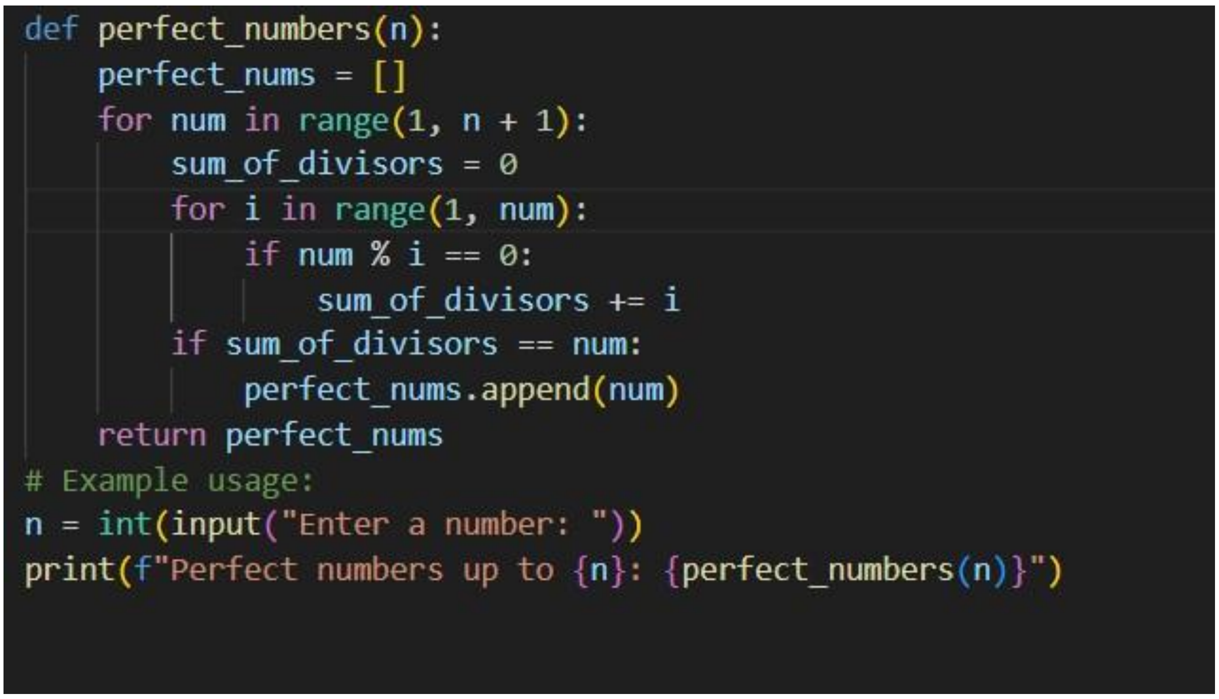
• Ask AI to regenerate an optimized version (using divisor check only up to

√n).

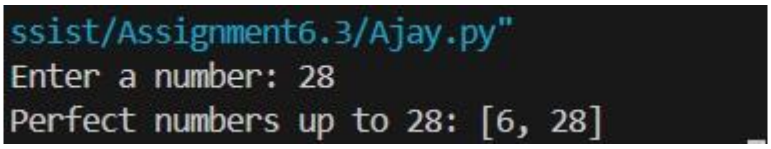
Expected Output #12:

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

• Optimized version with explanation.



```python
def perfect_numbers(n):
    perfect_nums = []
    for num in range(1, n + 1):
        sum_of_divisors = 0
        for i in range(1, num):
            if num % i == 0:
                sum_of_divisors += i
        if sum_of_divisors == num:
            perfect_nums.append(num)
    return perfect_nums
# Example usage:
n = int(input("Enter a number: "))
print(f"Perfect numbers up to {n}: {perfect_numbers(n)}")
```



```
ssist/Assignment6.3/Ajay.py"
Enter a number: 28
Perfect numbers up to 28: [6, 28]
```