AI ASSISTED CODING

ASSIGNMENT-6.3

D.Sandeep

2303A51665

Batch-23

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 = 100
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")
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.00037550926208496094 seconds
PS D:\AI>
```

```python
#generate all auomorphic numbeers within a 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 generate_automorphic_numbers(start, end):
    automorphic_numbers = []
    i = start
    while i <= end:
        if is_automorphic(i):
            automorphic_numbers.append(i)
        i += 1
    return automorphic_numbers
# Example usage
start_time = t.time()
start_range = 1
end_range = 100
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")
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.00037550926208496094 seconds
PS D:\AI>
```

1. They use very little memory since only a small list of automorphic numbers is stored.
2. A for loop is faster because it is optimized inside Python.
3. A for loop makes the code cleaner and easier to understand because no manual counter is needed.
4. A for loop is safer and less error-prone since it avoids skipping numbers or running into infinite loops.

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 dictionarybased or match-case structure.

Expected Output #2:

• Feedback classification function with explanation and an alternative approach.

```python
#generate a nested if-elif-else to classify shopping feedback as positive negative or neutral based
# on rating (1-5)
def classify_feedback(rating):
    if rating >= 4:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    else:
        return "Negative"
# Example usage
ratings = [5, 4, 3, 2, 1]
for rating in ratings:
    classification = classify_feedback(rating)
    print(f"Rating: {rating}, Feedback: {classification}")
#rewrite the above code using dictionary mapping
def classify_feedback_dict(rating):
    feedback_map = {
        5: "Positive",
        4: "Positive",
        3: "Neutral",
        2: "Negative",
        1: "Negative"
    }
    return feedback_map.get(rating, "Invalid rating")
# Example usage
ratings = [5, 4, 3, 2, 1]
for rating in ratings:
    classification = classify_feedback_dict(rating)
    print(f"Rating: {rating}, Feedback: {classification}")
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
Rating: 5, Feedback: Positive
Rating: 4, Feedback: Positive
Rating: 3, Feedback: Neutral
Rating: 2, Feedback: Negative
Rating: 1, Feedback: Negative
Rating: 5, Feedback: Positive
Rating: 4, Feedback: Positive
Rating: 3, Feedback: Neutral
Rating: 2, Feedback: Negative
Rating: 1, Feedback: Negative
```

1. A dictionary makes things clear because all rating-to-feedback pairs are visible in one place.
2. It is easy to maintain since you can add or change ratings without changing the main logic.
3. It handles invalid ratings more safely by letting you check or give a default message.
4. It follows Python's standard and clean way of mapping values, so the code looks natural.
5. It stays clean and readable even if the number of rating categories increases.

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 ,mode ,standard deviation,variance of the list
import statistics as stats
def statistical_operations(numbers):
    minimum = min(numbers)
    maximum = max(numbers)
    mean = stats.mean(numbers)
    median = stats.median(numbers)
    try:
        mode = stats.mode(numbers)
    except stats.StatisticsError:
        mode = "No unique mode"
    variance = stats.variance(numbers)
    std_deviation = stats.stdev(numbers)

    return {
        "Minimum": minimum,
        "Maximum": maximum,
        "Mean": mean,
        "Median": median,
        "Mode": mode,
        "Variance": variance,
        "Standard Deviation": std_deviation
    }
# Example usage
data = [10, 20, 20, 30, 40, 50]
results = statistical_operations(data)
for key, value in results.items():
    print(f"{key}: {value}")
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
Minimum: 10
Maximum: 50
Mean: 28.333333333333332
Median: 25.0
Mode: 20
Variance: 216.66666666666666
Standard Deviation: 14.719601443879744
```

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. (variable) experience: Any
        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

Contains exactly 10 digits

```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 D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
Enter an Indian mobile number: 8074762231
Valid Indian mobile number.
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
Enter an Indian mobile number: 8520883781
```

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.

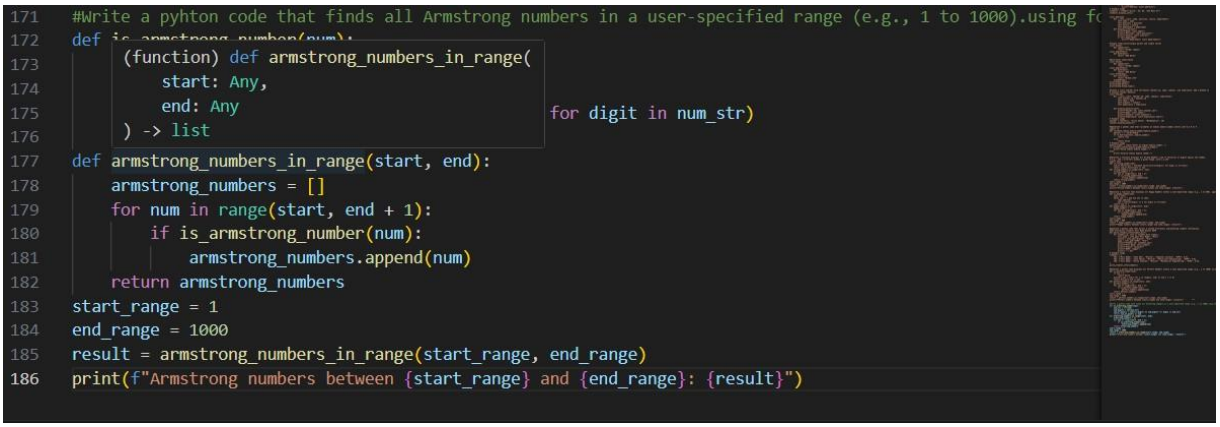
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).

• 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.



```
#Write a pyhton code that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).using fo
def is_armstrong_number(num):
    (function) def armstrong_numbers_in_range(
        start: Any,
        end: Any
    ) -> list                                   for digit in num_str)
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}")
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
d:\AI\Assignment-6.3.py:106: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
Armstrong numbers between 1 and 1000: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
```

1. It does the same work in fewer lines, so there is less overhead and faster execution.
2. It calculates the digit powers directly instead of using extra variables, reducing unnecessary steps.
3. List comprehension is faster than manually appending values in a loop.
4. It avoids storing extra temporary values, so memory usage stays low.
5. The code is cleaner and easier to read while giving the same correct result.

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 function that displays all Happy Numbers within a user-specified range (e.g., 1 to 500). reger
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 D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
d:\AI\Assignment-6.3.py:106: 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, 368, 376, 379,
383, 386, 391, 392, 397, 404, 409, 440, 446, 464, 469, 478, 487, 490, 496]
```

1. It saves results of numbers already checked, so the same calculations are not done again and again.
2. This avoids repeated loops for the same digit sequences, which makes it much faster for large ranges.
3. Using a set still prevents infinite loops, but caching adds an extra speed boost.

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 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 = 1000
result = strong_numbers_in_range(start_range, end_range)
print(f"Strong numbers between {start_range} and {end_range}: {result}")
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
d:\AI\Assignment-6.3.py:106: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
Strong numbers between 1 and 500: [1, 2, 145]
```

Time Complexity Comparison:

For larger ranges, this makes a significant performance difference since you're trading a tiny bit of extra memory (10 dictionary entries) for substantial computation savings across the entire loop.

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)
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
d:\AI\Assignment-6.3.py:106: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
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 root n

```python
#Generate a python code displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000). usir
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}")
```

```
--------------------
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
d:\AI\Assignment-6.3.py:106: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
Perfect numbers between 1 and 1000: [6, 28, 496]
```