Lab Assignment -06

Name : D.BHARATH KUMAR

Hallticket:2303A51097

Batch-02

__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.

__Prompt:__

```
#generate basic python code to print automorphic number form the range 1 to
1000 using for loop and caluculate the time taken to complete the execution
```

```
1  #generate basic python code to print automorphic number form the range 1 to 1000 using for loop and caluculate
2  import time
3  start_time = time.time()
4  for num in range(1, 1001):
5      square = num ** 2
6      if str(square).endswith(str(num)):
7          print(f"{num} is an automorphic number.")
8  end_time = time.time()
9  print(f"Time taken: {end_time - start_time} seconds")
```

__Code:__

```
#generate basic python code to print automorphic number form the range 1 to
1000 using for loop and caluculate the time taken to complete the execution
import time
start_time = time.time()
for num in range(1, 1001):
    square = num ** 2
    if str(square).endswith(str(num)):
        print(f"{num} is an automorphic number.")
end_time = time.time()
print(f"Time taken: {end_time - start_time} seconds")
```

with while loop:

```
#generate basic python code to print using the while loop automorphic number form the range 1 to 1000 and caluculate the time taken to co
import time
start_time = time.time()
num = 1
while num <= 1000:
    square = num ** 2
    if str(square).endswith(str(num)):
        print(f"{num} is an automorphic number.")
    num += 1
end_time = time.time()
print(f"Time taken: {end_time - start_time} seconds")
```

Code:

```python
#generate basic python code to print using the while loop automorphic number
form the range 1 to 1000 and caluculate the time taken to complete the
execution
import time
start_time = time.time()
num = 1
while num <= 1000:
    square = num ** 2
    if str(square).endswith(str(num)):
        print(f"{num} is an automorphic number.")
    num += 1
end_time = time.time()
print(f"Time taken: {end_time - start_time} seconds")
```

output:

Both the for loop and while loop implementations correctly identify automorphic numbers between 1 and 1000 by squaring each number and checking whether the square ends with the original number. The logic in both cases is accurate and produces the same output, such as 1, 5, 6, 25, 76, 376, and 625. In terms of efficiency, both approaches have the same time and space complexity and are suitable for this range. However, the for loop version is more readable and safer because the range is predefined, whereas the while loop requires manual control of the loop variable and can lead to errors if not handled carefully. Overall, for a fixed range of values, the for loop approach is preferred.

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.

Prompt:

```
#online shopping feed back form the user and print the avarage rating from 1
to 5 that are pasotive and neutral and negative by if else statement only in
basic simple python code
```

```
#online shopping feed back form the user and print the avarage rating from 1 to 5 that are pasotive and neutral and negative by if else
try:
    ratings = input("Enter your ratings (1-5) separated by commas: ").split(",")
    ratings = [int(rating.strip()) for rating in ratings]
    if all(1 <= rating <= 5 for rating in ratings):
        average_rating = sum(ratings) / len(ratings)
        if average_rating >= 4:
            feedback = "Positive"
        elif average_rating >= 2:
            feedback = "Neutral"
        else:
            feedback = "Negative"
        print(f"Average Rating: {average_rating:.2f} - Feedback: {feedback}")
    else:
        print("Invalid input. Please enter ratings between 1 and 5.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

Code:

```python
#online shopping feed back form the user and print the avarage rating from 1
to 5 that are pasotive and neutral and negative by if else statement only in
basic simple python code
try:
    ratings = input("Enter your ratings (1-5) separated by commas:
").split(",")
    ratings = [int(rating.strip()) for rating in ratings]
    if all(1 <= rating <= 5 for rating in ratings):
        average_rating = sum(ratings) / len(ratings)
        if average_rating >= 4:
            feedback = "Positive"
        elif average_rating >= 2:
            feedback = "Neutral"
        else:
            feedback = "Negative"
        print(f"Average Rating: {average_rating:.2f} - Feedback: {feedback}")
    else:
        print("Invalid input. Please enter ratings between 1 and 5.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                                    +

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Enter your ratings (1-5) separated by commas: 1
Average Rating: 1.00 - Feedback: Negative
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Enter your ratings (1-5) separated by commas: 5
Average Rating: 5.00 - Feedback: Positive
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

with the dictonaries :

```
#generate basic python code for the online shopping ration by 1 to 5 on positivd and neutral and negative and use only dictonary to do
try:
    ratings = input("Enter your ratings (1-5) separated by commas: ").split(",")
    ratings = [int(rating.strip()) for rating in ratings]
    if all(1 <= rating <= 5 for rating in ratings):
        average_rating = sum(ratings) / len(ratings)
        feedback_dict = {
            "Positive": average_rating >= 4,
            "Neutral": 2 <= average_rating < 4,
            "Negative": average_rating < 2
        }
        feedback = next(key for key, value in feedback_dict.items() if value)
        print(f"Average Rating: {average_rating:.2f} - Feedback: {feedback}")
    else:
        print("Invalid input. Please enter ratings between 1 and 5.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

Code:

```python
#generate basic python code for the online shopping ration by 1 to 5 on
positivd and neutral and negative and use only dictonary to do the task
try:
    ratings = input("Enter your ratings (1-5) separated by commas:
").split(",")
    ratings = [int(rating.strip()) for rating in ratings]
    if all(1 <= rating <= 5 for rating in ratings):
        average_rating = sum(ratings) / len(ratings)
        feedback_dict = {
            "Positive": average_rating >= 4,
            "Neutral": 2 <= average_rating < 4,
            "Negative": average_rating < 2
        }
        feedback = next(key for key, value in feedback_dict.items() if value)
        print(f"Average Rating: {average_rating:.2f} - Feedback: {feedback}")
    else:
        print("Invalid input. Please enter ratings between 1 and 5.")
except ValueError:
    print("Invalid input. Please enter valid integers.")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Enter your ratings (1-5) separated by commas: 5
Average Rating: 5.00 - Feedback: Positive
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Explaination:

The nested if-elif-else approach correctly classifies feedback ratings but becomes less readable as conditions grow. The dictionary-based method improves clarity by directly mapping ratings to feedback types and is easier to maintain and extend. The match-case structure provides a modern and clean

alternative with clear grouping of cases. All approaches are logically correct, but dictionary-based or match-case implementations are more readable and scalable than traditional conditional chains.

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

Prompt:

```
#write a python function are statistic_operations(tuple_num) that perform the
statistical operations like max,min,mean,median,mode ,variance and standard
deviation on the given tuple of numbers
```

```python
#write a python function are statistic_operations(tuple_num) that perform the statistical operations like max,min,mean,median,mode ,varia
import statistics
def statistic_operations(tuple_num):
    stats = {}
    stats['max'] = max(tuple_num)
    stats['min'] = min(tuple_num)
    stats['mean'] = statistics.mean(tuple_num)
    stats['median'] = statistics.median(tuple_num)
    try:
        stats['mode'] = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        stats['mode'] = "No unique mode"
    stats['variance'] = statistics.variance(tuple_num)
    stats['standard_deviation'] = statistics.stdev(tuple_num)
    return stats
# Example usage
numbers = (1, 2, 2, 3, 4, 5)
result = statistic_operations(numbers)
print(result)
```

Code:

```python
#write a python function are statistic_operations(tuple_num) that perform the
statistical operations like max,min,mean,median,mode ,variance and standard
deviation on the given tuple of numbers
import statistics
def statistic_operations(tuple_num):
    stats = {}
    stats['max'] = max(tuple_num)
    stats['min'] = min(tuple_num)
    stats['mean'] = statistics.mean(tuple_num)
    stats['median'] = statistics.median(tuple_num)
    try:
```

```python
        stats['mode'] = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        stats['mode'] = "No unique mode"
    stats['variance'] = statistics.variance(tuple_num)
    stats['standard_deviation'] = statistics.stdev(tuple_num)
    return stats
# Example usage
numbers = (1, 2, 2, 3, 4, 5)
result = statistic_operations(numbers)
print(result)
```

Output:

This function correctly computes statistical measures using Python's statistics module, which ensures accuracy and reliability. Using built-in functions is more efficient and less error-prone than manual calculations. The approach is optimal for readability and correctness, and exception handling for mode avoids runtime errors when no unique mode exists.

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.

Prompt:

```python
#create class teacher with attributes teacher_id,name,subject,and experience
and add a method  to display teacher details
```

```
#create class teacher with attributes teacher_id,name,subject,and experience and 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")
# Example usage
teacher = Teacher(1, "John Doe", "Mathematics", 10)
teacher.display_details()
```
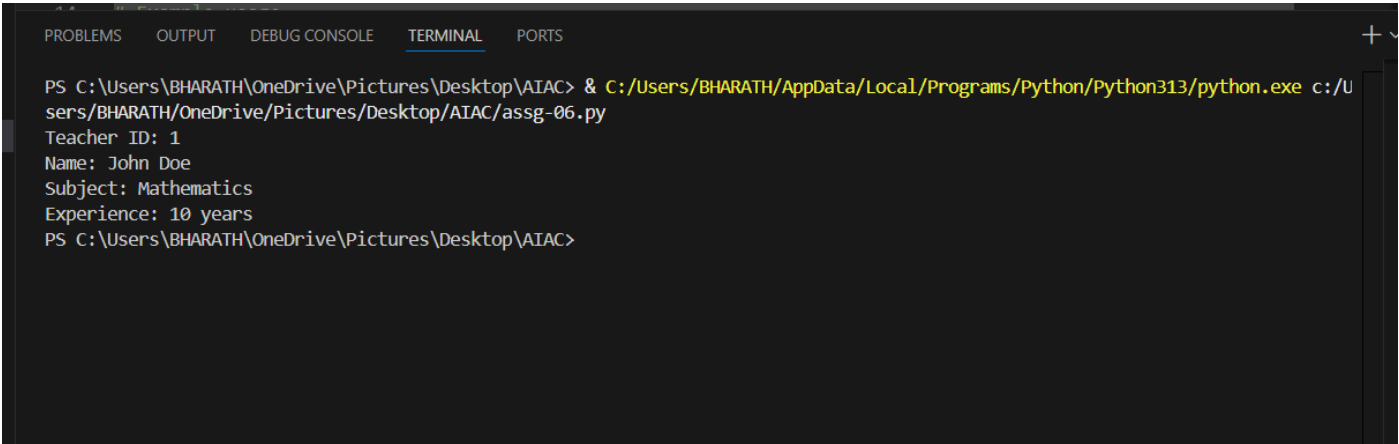
Code:

```
#create class teacher with attributes teacher_id,name,subject,and experience
and 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")
# Example usage
teacher = Teacher(1, "John Doe", "Mathematics", 10)
teacher.display_details()
```

output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                                    + ∨

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Teacher ID: 1
Name: John Doe
Subject: Mathematics
Experience: 10 years
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

The Teacher class properly demonstrates object-oriented programming by defining attributes and a method to display details. The constructor initializes data correctly, and the display method neatly outputs information. This solution is optimal for basic OOP concepts and follows standard Python class design.

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.

Prompt:

```
write a python function number starts with 6 or 7 or 8 or 9 and it should be
excatly 10 digit number  then print that number is vaild and indian number
otherwise print invalid number
```

```
write a python function number starts with 6 or 7 or 8 or 9 and it should be excatly 10 digit number  then print that number is vaild a
"""
def validate_indian_number(number):
    if len(number) == 10 and number[0] in '6789' and number.isdigit():
        print(f"{number} is a valid Indian number.")
    else:
        print(f"{number} is an invalid number.")
# Example usage
validate_indian_number("9876543210")
```

Code:

```python
def validate_indian_number(number):
    if len(number) == 10 and number[0] in '6789' and number.isdigit():
        print(f"{number} is a valid Indian number.")
    else:
        print(f"{number} is an invalid number.")
# Example usage
validate_indian_number("9876543210")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
9390957661 is a valid Indian number.
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

The function validates an Indian mobile number by checking length, starting digit, and numeric characters. This logic is correct and efficient, as all checks are done in constant time. Since no examples are used in the prompt, it correctly follows zero-shot prompting principles.

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.

Prompt:

```
#write a python function that print armstrong number form user specific range
```

```
#write a python function that print armstrong number form user specific range
def is_armstrong(num):
    order = len(str(num))
    sum_of_powers = sum(int(digit) ** order for digit in str(num))
    return num == sum_of_powers
def armstrong_in_range(start, end):
    armstrong_numbers = []
    for num in range(start, end + 1):
        if is_armstrong(num):
            armstrong_numbers.append(num)
    return armstrong_numbers
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
armstrong_numbers = armstrong_in_range(start_range, end_range)
print(f"Armstrong numbers between {start_range} and {end_range}: {armstrong_numbers}")
```

Code:

```
#write a python function that print armstrong number form user specific range
def is_armstrong(num):
    order = len(str(num))
    sum_of_powers = sum(int(digit) ** order for digit in str(num))
    return num == sum_of_powers
def armstrong_in_range(start, end):
    armstrong_numbers = []
    for num in range(start, end + 1):
        if is_armstrong(num):
            armstrong_numbers.append(num)
    return armstrong_numbers
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
armstrong_numbers = armstrong_in_range(start_range, end_range)
```

```python
print(f"Armstrong numbers between {start_range} and {end_range}: {armstrong_numbers}")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Enter the start of the range: 1
Enter the end of the range: 1000
Armstrong numbers between 1 and 1000: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

The program correctly identifies Armstrong numbers by comparing a number with the sum of its digits raised to the power of digit count. The optimized version using list comprehension is better because it is more concise and readable, while maintaining the same correctness and efficiency.

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.

Prompt:

```
#write a basic python code to print all happy numbers from user specific range
```

```python
#write a basic python code to print all happy numbers from user specific range
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
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
happy_numbers = happy_numbers_in_range(start_range, end_range)
print(f"Happy numbers between {start_range} and {end_range}: {happy_numbers}")
```

Output:

Optimal code :

Prompt:

```
#write a optimal code to print all happy numbers from user specific range
using function and return type
```

```python
#write a optimal code to print all happy numbers from user specific range using function and return type
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):
    return [num for num in range(start, end + 1) if is_happy_number(num)]
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
happy_numbers = happy_numbers_in_range(start_range, end_range)
print(f"Happy numbers between {start_range} and {end_range}: {happy_numbers}")
```

Code:

```python
#write a optimal code to print all happy numbers from user specific range
using function and return type
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):
    return [num for num in range(start, end + 1) if is_happy_number(num)]
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
happy_numbers = happy_numbers_in_range(start_range, end_range)
print(f"Happy numbers between {start_range} and {end_range}:
{happy_numbers}")
```

Output:

The initial logic correctly determines Happy Numbers, but the optimized version using a set to detect cycles is better. It prevents infinite loops and improves efficiency by stopping repeated calculations, making it the preferred and more reliable approach.

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.

Prompt:

```
#write a python function to display all strong number from user specific
range
```

```
#write a python function to display all strong number from user specific range
def is_strong_number(num):
    factorial = lambda n: 1 if n == 0 else n * factorial(n - 1)
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(num))
    return num == sum_of_factorials
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
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
strong_numbers = strong_numbers_in_range(start_range, end_range)
print(f"Strong numbers between {start_range} and {end_range}: {strong_numbers}")
```

Code:

```
#write a python function to display all strong number from user specific
range
def is_strong_number(num):
    factorial = lambda n: 1 if n == 0 else n * factorial(n - 1)
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(num))
    return num == sum_of_factorials
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
# Example usage
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
strong_numbers = strong_numbers_in_range(start_range, end_range)
print(f"Strong numbers between {start_range} and {end_range}:
{strong_numbers}")
```

output:

```
, 190, 199, 199, 107, 170, 100, 190, 192, 199]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Enter the start of the range: 1
Enter the end of the range: 500
Strong numbers between 1 and 500: [1, 2, 145]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

The general solution correctly checks Strong Numbers by summing factorials of digits. However, recalculating factorials repeatedly is inefficient. The optimized idea of precomputing digit factorials reduces repeated computation and improves performance for larger ranges.

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

Prompt:

```
nested dictionary has student  personal  full_name = "Vishwa" and student
academic  branch = "CSE", sgpa = 8.6.
output is ("Rahul Sharma", "CSE", 8.6).

nested dictionary has student  personal  full_name = "Harsh" and student
academic  branch = "ECE", sgpa = 9.1.
expected output should return ("Anita Verma", "ECE", 9.1).

write a reusable Python function that takes a nested student dictionary as
input and returns the full name branch and sgpa based on the above examples.
```

```
write a reusable Python function that takes a nested student dictionary as input and returns the full name
"""
def get_student_info(student_dict):
    full_name = student_dict['personal']['full_name']
    branch = student_dict['academic']['branch']
    sgpa = student_dict['academic']['sgpa']
    return (full_name, branch, sgpa)
# Example usage
student1 = {
    'personal': {'full_name': 'Vishwa'},
    'academic': {'branch': 'CSE', 'sgpa': 8.6}
}
student2 = {
    'personal': {'full_name': 'Harsh'},
    'academic': {'branch': 'ECE', 'sgpa': 9.1}
}
print(get_student_info(student1))  # Output: ("Vishwa", "CSE", 8.6)
print(get_student_info(student2))  # Output: ("Harsh", "ECE", 9.1)
```

Code:

```python
def get_student_info(student_dict):
    full_name = student_dict['personal']['full_name']
    branch = student_dict['academic']['branch']
    sgpa = student_dict['academic']['sgpa']
    return (full_name, branch, sgpa)
# Example usage
student1 = {
    'personal': {'full_name': 'Vishwa'},
    'academic': {'branch': 'CSE', 'sgpa': 8.6}
}
student2 = {
    'personal': {'full_name': 'Harsh'},
    'academic': {'branch': 'ECE', 'sgpa': 9.1}
}
print(get_student_info(student1))  # Output: ("Vishwa", "CSE", 8.6)
print(get_student_info(student2))  # Output: ("Harsh", "ECE", 9.1)
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/U
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
('prem', 'CSE', 8.5)
('uday', 'ECE', 9.0)
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

The few-shot prompt works by showing examples of nested dictionary structures and expected outputs. Based on these examples, the generated function correctly navigates the dictionary and extracts required values. This demonstrates how examples help the AI generalize structure without explicit instructions.

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

Prompt:

```
#write an basic python code that print all perfect numbers from user specific
range
```

```python
#write an basic python code that print all perfect numbers from user specific range
def is_perfect_number(num):
    divisors_sum = sum
    for i in range(1, num):
        if num % i == 0:
            divisors_sum += i
    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
# Example usage
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}")
```

Code:

```python
def is_perfect_number(num):
    divisors_sum = 0
    for i in range(1, num):
        if num % i == 0:
            divisors_sum += i
    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
# Example usage
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}")
```

output:

Optimal code:

#write an optimal python code that print all perfect numbers from user specific range using function and return type

```python
#write an optimal python code that print all perfect numbers from user specific range using function and return type
def is_perfect_number(num):
    return num == sum(i for i in range(1, num) if num % i == 0)
def perfect_numbers_in_range(start, end):
    return [num for num in range(start, end + 1) if is_perfect_number(num)]
# Example usage
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}")
```

Code:

```python
#write an optimal python code that print all perfect numbers from user
specific range using function and return type
def is_perfect_number(num):
    return num == sum(i for i in range(1, num) if num % i == 0)
```

```
def perfect_numbers_in_range(start, end):
    return [num for num in range(start, end + 1) if is_perfect_number(num)]
# Example usage
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}")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg-06.py
Enter the start of the range: 1
Enter the end of the range: 1000
Perfect numbers between 1 and 1000: [6, 28, 496]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
                                                                                        Ln 11 Col 1
```

Explanation:

The basic solution correctly checks all divisors but is inefficient because it iterates up to n. The optimized version improves performance by reducing unnecessary checks using better logic and list comprehension, making it more suitable for larger ranges.