

## Assignment-6

Name : U.VIGNESH  
Ht.no : 2303A51964  
Batch 24

### 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.

#### Code:

```
C:\Users> PC > Downloads > aiac > assign_6.py > ...
1  # generate a python code to display all automorphic numbers between the range 1 to 1000 using for loop
2  import time as t
3  def is_automorphic(num):
4      square = num * num
5      return str(square).endswith(str(num))
6  def automorphic_numbers_in_range(start, end):
7      automorphic_numbers = []
8      for num in range(start, end + 1):
9          if is_automorphic(num):
10              automorphic_numbers.append(num)
11      return automorphic_numbers
12 start_time = t.time()
13 start_range = 1
14 end_range = 1000
15 automorphic_numbers = automorphic_numbers_in_range(start_range, end_range)
16 print(f"Automorphic numbers between {start_range} and {end_range} are: {automorphic_numbers}")
17 end_time = t.time()
18 print(f"Execution time: {end_time - start_time} seconds")
```

```
# generate a python code to display all automorphic numbers between the range 1 to 1000 using while loop
import time as t
def is_automorphic(num):
    square = num * num
    return str(square).endswith(str(num))
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 = 1
end_range = 1000
automorphic_numbers = automorphic_numbers_in_range(start_range, end_range)
print(f"Automorphic numbers between {start_range} and {end_range} are: {automorphic_numbers}")
end_time = t.time()
print(f"Execution time: {end_time - start_time} seconds")
```

#### Output:

```
PS C:\Users\PC & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/assign_6.py
Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]
Execution time: 0.000714063644091797 seconds
Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]
Execution time: 0.0006761550903320312 seconds
PS C:\Users\PC>
```

**Explanation :** The for loop is more Pythonic and cleaner than the manual while loop. Prefer for loops when iterating over a known range. Both implementations produce identical results and similar execution times.

### 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

#### CODE :

```
C:\> Users> PC > Downloads> aiac > assign_6(2).py > ...
1  #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 to 5
2  def classify_feedback(rating):
3      if rating >= 4:
4          return "Positive"
5      else:
6          if rating == 3:
7              return "Neutral"
8          else:
9              return "Negative"
10
11 # Example usage
12 rating = int(input("Enter your rating (1-5): "))
13 feedback = classify_feedback(rating)
14 print(f"Feedback classification: {feedback}")
15
16 # rewrite the above code using dictionaries
17 def classify_feedback_dict(rating):
18     feedback_dict = {
19         5: "positive",
20         4: "positive",
21         3: "neutral",
22         2: "negative",
23         1: "negative"
24     }
25     return feedback_dict.get(rating, "Invalid rating")
26 # Example usage
27 rating = int(input("Enter your rating (1-5): "))
28 feedback = classify_feedback_dict(rating)
29 print(f"Feedback classification: {feedback}")
30
```

#### OUTPUT :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(2).py"
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.

#### Task 3: Statistical\_operations

Define a function named statistical\_operations(tuple\_num) that performs the following statistical operations on a tuple of numbers:

- Minimum, Maximum
- Mean, Median, Mode
- Variance, Standard Deviation

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

#### Code :

```
C:\> Users > PC > Downloads > aiac > assign_6(3).py > statistical_operations
1  def statistical_operations(data):
2      """
3          Perform basic statistical operations on a list of numbers.
4
5          Parameters:
6          data (list of float): A list of numerical values.
7
8          Returns:
9          dict: A dictionary containing the mean, median, and standard deviation.
10         """
11         import statistics
12
13         if not data:
14             return {"mean": None, "median": None, "std_dev": None}
15
16         mean = statistics.mean(data)
17         median = statistics.median(data)
18         std_dev = statistics.stdev(data) if len(data) > 1 else 0.0
19
20         return {
21             "mean": mean,
22             "median": median,
23             "std_dev": std_dev
24         }
25
26 # Example usage
27 data = [10, 20, 30, 40, 50]
28 results = statistical_operations(data)
29 print("Statistical Operations Results:")
30 print(f"Mean: {results['mean']}")
31 print(f"Median: {results['median']}")
32 print(f"Standard Deviation: {results['std_dev']}")
```

### Output :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(3).py"
Statistical Operations Results:
Mean: 30
Median: 30
Standard Deviation: 15.811388300841896
```

### Task 4: Teacher Profile

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

#### Code :

```
C: > Users > PC > Downloads > aiac > assign_6(4).py > ...
1  class Teacher:
2      def __init__(self, name, id, subject,experience):
3          self.name = name
4          self.id = id
5          self.subject = subject
6          self.experience = experience
7      def display_info(self):
8          print(f"Teacher Name: {self.name}")
9          print(f"Teacher ID: {self.id}")
10         print(f"Subject: {self.subject}")
11         print(f"Years of Experience: {self.experience}")
12 # Example usage
13 teacher1 = Teacher("Alice Johnson", "T123", "Mathematics", 10)
14 teacher1.display_info()
15
```

### Output :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(4).py"
Teacher Name: Alice Johnson
Teacher ID: T123
Subject: Mathematics
Years of Experience: 10
```

### Task #5 – Zero-Shot Prompting with Conditional Validation

Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.

#### Requirements

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

#### Expected Output

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

#### Code :

```
C: > Users > PC > Downloads > aiac > assign_6(5).py > ...
1  #generate a python code that validates an Indian mobile number.
2  import re
3
4  def validate_indian_mobile_number(mobile_number):
5      """
6          Validates an Indian mobile number.
7
8          Parameters:
9          mobile_number (str): The mobile number to be validated.
10
11         Returns:
12         bool: True if the mobile number is valid, False otherwise.
13
14         # Indian mobile numbers are 10 digits long and start with 6, 7, 8, or
15         pattern = r'^[6-9]\d{9}$'
16
17         if re.match(pattern, mobile_number):
18             return True
19         else:
20             return False
21
22     # Example usage
23     mobile_number = "9876543210"
24     if validate_indian_mobile_number(mobile_number):
25         print(f"\{mobile_number}\ is a valid Indian mobile number.")
26     else:
27         print(f"\{mobile_number}\ is not a valid Indian mobile number.")
```

### Output :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(5).py"
9876543210 is a valid 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.

Code:

```
C:> Users > PC > Downloads > aiac > assign_6(6).py > ...
1  # generate a python code to display the armstrong numbers in a given range using loops
2  start = int(input("Enter the start of the range: "))
3  end = int(input("Enter the end of the range: "))
4
5  def is_armstrong(num):
6      num_str = str(num)
7      num_digits = len(num_str)
8      sum_of_powers = 0
9      for digit in num_str:
10          sum_of_powers += int(digit) ** num_digits
11      return sum_of_powers == num
12
13  print(f"Armstrong numbers in the range {start} to {end}:")
14  for i in range(start, end + 1):
15      if is_armstrong(i):
16          print(i)
```

Output :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(6).py"
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
PS C:\Users\PC>
```

#### Task Description #7 (Loops – Happy Numbers in a Range)

Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).

Instructions:

- Implement the logic using a loop: repeatedly replace a number with the sum of the squares of its digits until the result is either 1 (Happy Number) or enters a cycle (Not Happy).
- Validate correctness by checking known Happy Numbers (e.g., 1, 7, 10, 13, 19, 23, 28...).
- Ask AI to regenerate an optimized version (e.g., by using a set to detect cycles instead of infinite loops).

Expected Output #8:

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

**Code:**

```
C:\ > Users > PC > Downloads > aiac > assign_6(7).py > ...
1 # generate a python code to display all happy numbers within the range
2 def is_happy_number(n):
3     seen = set()
4     while n != 1 and n not in seen:
5         seen.add(n)
6         n = sum(int(digit) ** 2 for digit in str(n))
7     return n == 1
8 def happy_numbers_in_range(start, end):
9     happy_numbers = []
10    for num in range(start, end + 1):
11        if is_happy_number(num):
12            happy_numbers.append(num)
13    return happy_numbers
14 if __name__ == "__main__":
15    start = int(input("Enter the start of the range: "))
16    end = int(input("Enter the end of the range: "))
17    happy_numbers = happy_numbers_in_range(start, end)
18    print(f"Happy numbers between {start} and {end}: {happy_numbers}")
19

#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 :**

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(7).py"
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, 379, 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, 913, 931, 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, 379, 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, 913, 931, 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.

**Task Description #8 (Loops – Strong Numbers in a Range)**

Task: Generate a function using AI that displays all Strong Numbers (sum of factorial of digits equals the number, e.g.,  $145 = 1!+4!+5!$ ) within a given range.

**Instructions:**

- Use loops to extract digits and calculate factorials.
- Validate with examples (1, 2, 145).
- Ask AI to regenerate an optimized version (precompute digit factorials).

**Expected Output #9:**

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

**Code :**

```

C:\Users\PC>Downloads>aiac>assign_6(8).py>...
1  #generate a python code to display all the strong numbers within a given range using loops
2  def factorial(n):
3      if n == 0 or n == 1:
4          return 1
5      fact = 1
6      for i in range(2, n + 1):
7          fact *= i
8      return fact
9  def isStrongNumber(num):
10     sum_of_factorials = 0
11     temp = num
12     while temp > 0:
13         digit = temp % 10
14         sum_of_factorials += factorial(digit)
15         temp //= 10
16     return sum_of_factorials == num
17  def findStrongNumbersInRange(start, end):
18      strong_numbers = []
19      for num in range(start, end + 1):
20          if isStrongNumber(num):
21              strong_numbers.append(num)
22      return strong_numbers
23
24 if __name__ == "__main__":
25     start_range = int(input("Enter the start of the range: "))
26     end_range = int(input("Enter the end of the range: "))
27     strong_numbers = findStrongNumbersInRange(start_range, end_range)
28     print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers}")

```

```

#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 isStrongNumber(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 findStrongNumbersInRange(start, end):
    strong_numbers = []
    for num in range(start, end + 1):
        if isStrongNumber(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 = findStrongNumbersInRange(start_range, end_range)
    print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers}")

```

## Output :

```

Strong numbers between 1 and 1000 are: [1, 2, 145]
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(8).py"
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 start of the range: 1
Enter the end of the range: 1000
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.

## Task #9 – Few-Shot Prompting for Nested Dictionary Extraction

### Objective

Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.

### Requirements

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

### Expected Output

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

### Code :

```

C:\Users\PC>Downloads>aiac>assign_6(9).py>...
1
2     Ravi Kumar, CSE, 8.7
3     Anita Sharma, ECE, 9.1
4     Suresh Reddy, ME, 7.9
5
6     """
7     def parse_student_data(data):
8         students = []
9         for line in data.strip().split('\n'):
10             name, dept, gpa = line.split(',')
11             students.append({
12                 'name': name,
13                 'department': dept,
14                 'gpa': float(gpa)
15             })
16
17     return students
18 if __name__ == "__main__":
19     data = """Ravi Kumar, CSE, 8.7
20     Anita Sharma, ECE, 9.1
21     Suresh Reddy, ME, 7.9"""
22     student_list = parse_student_data(data)
23     for student in student_list:
24         print(student)

```

## Output :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(9).py"
{'name': 'Ravi Kumar', 'department': 'CSE', 'gpa': 8.7}
{'name': 'Anita Sharma', 'department': 'ECE', 'gpa': 9.1}
{'name': 'Suresh Reddy', 'department': 'ME', 'gpa': 7.9}
```

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

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

Instructions:

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

Code :

```
C: > Users > PC > Downloads > aiac > assign_6(10).py > ...
1  # Generate a python code to display all perfect numbers within a given range using loops
2  def is_perfect_number(num):
3      if num < 2:
4          return False
5      divisors_sum = sum(i for i in range(1, num) if num % i == 0)
6      return divisors_sum == num
7  def perfect_numbers_in_range(start, end):
8      perfect_numbers = []
9      for num in range(start, end + 1):
10         if is_perfect_number(num):
11             perfect_numbers.append(num)
12     return perfect_numbers
13 if __name__ == "__main__":
14     start_range = int(input("Enter the start of the range: "))
15     end_range = int(input("Enter the end of the range: "))
16     perfect_numbers = perfect_numbers_in_range(start_range, end_range)
17     print(f"Perfect numbers between {start_range} and {end_range}: {perfect_numbers}")
18
```

```
# 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 :

```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/assign_6(10).py"
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]
```