

LAB 6.3

Hemavathi N

2303A51965

Batch 24

AIAC Lab

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.

```
C: > Users > ADMIN > OneDrive > Desktop > certificates > LAB 6.3 AIAC.py > is_automorphic
 1  #generate all automorphic numbers within the range using for loop with execution time measurement
 2  import time
 3  def is_automorphic(n):
 4      square = n * n
 5      return str(square).endswith(str(n))
 6  start_time_for = time.time()
 7  automorphic_numbers_for = []
 8  for i in range(1, 1001):
 9      if is_automorphic(i):
10          automorphic_numbers_for.append(i)
11  end_time_for = time.time()
12  print("Automorphic numbers between 1 and 1000 using for loop are:", automorphic_numbers_for)
13  end_time_for - start_time_for
14  start_time_while = time.time()
15  #generate all automorphic numbers within the range using while loop with execution time measurement
16  automorphic_numbers_while = []
17  i = 1
18  while i <= 1000:
19      if is_automorphic(i):
20          automorphic_numbers_while.append(i)
21      i += 1
22  end_time_while = time.time()
23  print("Automorphic numbers between 1 and 1000 using while loop are:", automorphic_numbers_while)
24  end_time_while - start_time_while
25  print("Execution time using for loop:", end_time_for - start_time_for)
26  print("Execution time using while loop:", end_time_while - start_time_while)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

py"

Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]

re: [1, 5, 6, 25, 76, 376, 625]

re: [1, 5, 6, 25, 76, 376, 625]

Automorphic numbers between 1 and 1000 using while loop are: [1, 5, 6, 25, 76, 376, 625]

PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "c:/Users/ADMIN/OneDrive/Desktop/c

Enter your rating (1-5): 5

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.

```
#generate a python code of shopping feedback as positive,neutral,negative based on numerical rating from 1 to 5 using nested if-elif-else statements
def get_feedback(rating):
    if rating >= 1 and rating <= 5:
        if rating == 5:
            return "Positive"
        elif rating == 4:
            return "Positive"
        elif rating == 3:
            return "Neutral"
        elif rating == 2:
            return "Negative"
        else: # rating == 1
            return "Negative"
    else:
        return "Invalid rating. Please provide a rating between 1 and 5."
# Example usage
rating = int(input("Enter your rating (1-5): "))
feedback = get_feedback(rating)
print("Your feedback is:", feedback)
#rewrite the above code using dictionary mapping
def get_feedback_dict(rating):
    feedback_map = {
        5: "Positive",
        4: "Positive",
        3: "Neutral",
        2: "Negative",
        1: "Negative"
    }
    return feedback_map.get(rating, "Invalid rating. Please provide a rating between 1 and 5.")
# Example usage
rating = int(input("Enter your rating (1-5): "))
feedback = get_feedback_dict(rating)
print("Your feedback is:", feedback)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Your feedback is: Positive
PS C:\Users\ADMIN> & "C:/Program Files/Python
Enter your rating (1-5): 3
Your feedback is: Neutral
Enter your rating (1-5): 4
Your feedback is: Positive
PS C:\Users\ADMIN> []
```

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

```
C: > Users > ADMIN > OneDrive > Desktop > certificates > import math.py > ...
1  def statistical_operations(tuple_num):
2      import statistics
3
4      mean = statistics.mean(tuple_num)
5      median = statistics.median(tuple_num)
6      minimum = min(tuple_num)
7      maximum = max(tuple_num)
8      variance = statistics.variance(tuple_num)
9      standard_deviation = statistics.stdev(tuple_num)
10     return mean, median, minimum, maximum, variance, standard_deviation
11 # Example usage:
12 numbers = (10, 20, 30, 40, 50)
13 mean, median, minimum, maximum, variance, standard_deviation = statistical_operations(numbers)
14 print("Mean:", mean)
15 print("Median:", median)
16 print("Minimum:", minimum)
17 print("Maximum:", maximum)
18 print("Variance:", variance)
19 print("Standard Deviation:", standard_deviation)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Mean: 30
Median: 30
Minimum: 10
Maximum: 50
Variance: 250
Standard Deviation: 15.811388300841896
PS C:\Users\ADMIN> []
gent Open Website
```

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 > ADMIN > OneDrive > Desktop > certificates > LAB 6.3 AIAC.py > ...
1  # Create a class Teacher with attributes teacher_id, name, subject, and experience.
2  # Add a method to display teacher details.
3  # Expected Output: Class with initializer, method, and object creation.
4  class Teacher:
5      def __init__(self, teacher_id, name, subject, experience):
6          self.teacher_id = teacher_id
7          self.name = name
8          self.subject = subject
9          self.experience = experience
10
11     def display_details(self):
12         print(f"Teacher ID: {self.teacher_id}")
13         print(f"Name: {self.name}")
14         print(f"Subject: {self.subject}")
15         print(f"Experience: {self.experience} years")
16
17     # Creating an object of the Teacher class
18     teacher1 = Teacher(101, "Alice Smith", "Mathematics", 10)
19     teacher1.display_details()
```

```
PS C:\Users\ADMIN> & "C:/Program Files/Python3  
Teacher ID: 101  
Name: Alice Smith  
Subject: Mathematics  
Experience: 10 years  
PS C:\Users\ADMIN> []  
Open Website
```

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.

```
C:\Users\ADMIN> OneDrive> Desktop> certificates> AIAC 6 (2).py > ...  
1  #generate a python code that validates an Indian mobile number.  
2  import re  
3  def validate_indian_mobile_number(mobile_number):  
4      # Indian mobile numbers start with 7, 8, or 9 and are 10 digits long  
5      pattern = r'^[789]\d{9}$'  
6      if re.match(pattern, mobile_number):  
7          return True  
8      else:  
9          return False  
10     # Example usage:  
11     mobile_number = input("Enter an Indian mobile number: ")  
12     if validate_indian_mobile_number(mobile_number):  
13         print("Valid Indian mobile number.")  
14     else:  
15         print("Invalid Indian mobile number.")  
16
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS  
PS C:\Users\ADMIN> & "C:/Program Files/Python313/p  
Enter an Indian mobile number: 8976543211  
Valid Indian mobile number.  
PS C:\Users\ADMIN> & "C:/Program Files/Python313/p  
Enter an Indian mobile number: 4567890  
Invalid Indian mobile number.  
PS C:\Users\ADMIN> []
```

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.

```
C:\> Users > ADMIN > OneDrive > Desktop > certificates > import math.py > ...
1  def find_armstrong_numbers(start, end):
2      armstrong_numbers = []
3      for num in range(start, end + 1):
4          order = len(str(num))
5          sum_of_powers = sum(int(digit) ** order for digit in str(num))
6          if num == sum_of_powers:
7              armstrong_numbers.append(num)
8      return armstrong_numbers
9  armstrong_numbers = find_armstrong_numbers(1, 1000)
10 print("Armstrong numbers between 1 and 1000 are:", armstrong_numbers)

PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "c:/Users/ADMIN/OneDrive/Desktop/certifi...
Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
PS C:\Users\ADMIN>
```

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.

```
1  #Generate a python code to display all happy numbers within the range
2  def is_happy_number(num):
3      seen = set()
4      while num != 1 and num not in seen:
5          seen.add(num)
6          num = sum(int(digit) ** 2 for digit in str(num))
7      return num == 1
8  def find_happy_numbers(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 happy_numbers = find_happy_numbers(1, 100)
15 print("Happy numbers between 1 and 100 are:", happy_numbers)
```

```

16 #optimise the code using cycle detection algorithm
17 def is_happy_number_optimized(num):
18     slow = num
19     fast = num
20     while True:
21         slow = sum(int(digit) ** 2 for digit in str(slow))
22         fast = sum(int(digit) ** 2 for digit in str(fast))
23         fast = sum(int(digit) ** 2 for digit in str(fast))
24         if slow == 1 or fast == 1:
25             return True
26         if slow == fast:
27             return False
28 def find_happy_numbers_optimized(start, end):
29     happy_numbers = []
30     for num in range(start, end + 1):
31         if is_happy_number_optimized(num):
32             happy_numbers.append(num)
33     return happy_numbers
34 happy_numbers_optimized = find_happy_numbers_optimized(1, 100)
35 print("Optimized happy numbers between 1 and 100 are:", happy_numbers_optimized)
36

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "c:/Users/ADMIN/OneDrive/Desktop/certificates/LAB 6.3 AIAC.py"
Happy numbers between 1 and 100 are: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "c:/Users/ADMIN/OneDrive/Desktop/certificates/LAB 6.3 AIAC.py"
Happy numbers between 1 and 100 are: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
optimized happy numbers between 1 and 100 are: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
PS C:\Users\ADMIN>

```

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.

```

#generate a python code to display all strong numbers within a given range
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
def is_strong_number(num):
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(num))
    return num == sum_of_factorials
def find_strong_numbers(start, end):
    strong_numbers = []
    for num in range(start, end + 1):
        if is_strong_number(num):
            strong_numbers.append(num)
    return strong_numbers
strong_numbers = find_strong_numbers(1, 100)
print("Strong numbers between 1 and 100 are:", strong_numbers)

```

```

#optimize the above code
def is_strong_number_optimized(num, factorial_cache={}):
    sum_of_factorials = 0
    for digit in str(num):
        d = int(digit)
        if d not in factorial_cache:
            factorial_cache[d] = factorial(d)
        sum_of_factorials += factorial_cache[d]
    return num == sum_of_factorials
def find_strong_numbers_optimized(start, end):
    strong_numbers = []
    for num in range(start, end + 1):
        if is_strong_number_optimized(num):
            strong_numbers.append(num)
    return strong_numbers
strong_numbers_optimized = find_strong_numbers_optimized(1, 100)
print("Optimized strong numbers between 1 and 100 are:", strong_numbers_optimized)

```

```

PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "C:/Users/ADMIN/Desktop/Python/assign_6(8).py"
Strong numbers between 1 and 100 are: [1, 2]
PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "C:/Users/ADMIN/Desktop/Python/assign_6(8).py"
Strong numbers between 1 and 100 are: [1, 2]
Optimized strong numbers between 1 and 100 are: [1, 2]
PS C:\Users\ADMIN>

```

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

```

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 is_strong_number(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 find_strong_numbers_in_range(start, end):
18      strong_numbers = []
19      for num in range(start, end + 1):
20          if is_strong_number(num):
21              strong_numbers.append(num)
22      return strong_numbers
23  if __name__ == "__main__":
24      start_range = int(input("Enter the start of the range: "))
25      end_range = int(input("Enter the end of the range: "))
26      strong_numbers = find_strong_numbers_in_range(start_range, end_range)
27      print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers}")
28

```

```
{'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}).

Expected Output #12:

- Python program that lists Perfect Numbers in the given range.
- Optimized version with explanation.

```
#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
    sum_of_divisors = 0
    for i in range(1, num):
        if num % i == 0:
            sum_of_divisors += i
    return sum_of_divisors == num
def find_perfect_numbers(start, end):
    perfect_numbers = []
    for num in range(start, end + 1):
        if is_perfect_number(num):
            perfect_numbers.append(num)
    return perfect_numbers
perfect_numbers = find_perfect_numbers(1, 1000)
print("Perfect numbers between 1 and 1000 are:", perfect_numbers)
```

```
#Optimize the above code
def is_perfect_number_optimized(num):
    if num < 2:
        return False
    sum_of_divisors = 1 # 1 is a divisor for all num > 1
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            sum_of_divisors += i
            if i != num // i:
                sum_of_divisors += num // i
    return sum_of_divisors == num
def find_perfect_numbers_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
perfect_numbers_optimized = find_perfect_numbers_optimized(1, 1000)
print("Optimized perfect numbers between 1 and 1000 are:", perfect_numbers_optimized)
```

```
Perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\ADMIN> & "C:/Program Files/Python313/python.exe" "c:/Users/
Perfect numbers between 1 and 1000 are: [6, 28, 496]
Optimized perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\ADMIN> []
```