Ai Assisted Coding

Name:K.Rupaas shyni

Hall.No:2303A52417

Batch:32

1.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:

```
'''
Write a python code that display all automorphic numbers between 1 to 1000
write using for loop
'''
import time as t
start_time = t.time()
for num in range(1, 1001):
    square = num ** 2
    if str(square).endswith(str(num)):
        print(num)
end_time = t.time()
print("Execution time:", end_time - start_time)

# Using while loop
import time as t
start_time = t.time()
num = 1
while num <= 1000:
    square = num ** 2
    if str(square).endswith(str(num)):
        print(num)
    num += 1
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
PS C:\Users\rupaa\OneDrive\Desktop\AIAC> &
1
5
6
25
76
376
625
Execution time: 0.0014467239379882812
1
5
6
25
76
376
625
Execution time: 0.0010151863098144531
PS C:\Users\rupaa\OneDrive\Desktop\AIAC>
```

2.Task Description #2 (Conditional Statements – Online Shopping Feedback

Classification)

• Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).

• Instructions:

o Generate initial code using nested if-elif-else.

o Analyze correctness and readability.

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

Expected Output #2:

• Feedback classification function with explanation and an alternative approach.

Code:

```
'''
write a python code to classify the feedback as Positive,Negative or Neutral using nestwd if elif else based on rating provided by user from 1 to 5
write the time taken to execute the code

'''
import time as t
start_time = t.time()
rating = int(input("Enter your rating (1-5): "))
if 1 <= rating <= 5:
    if rating >= 4:
        print("Positive Feedback")
    elif rating == 3:
        print("Neutral Feedback")
    else:
        print("Negative Feedback")
else:
    print("Invalid rating. Please enter a number between 1 and 5.")
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
Enter your rating (1-5): 4
Positive Feedback
Execution time: 3.6358706951141357
PS C:\Users\rupaa\OneDrive\Desktop\AIAC>
```

3.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:

```
'''
write a python code that performs Minimum, Maximum,Mean,Median,Mode,Variance,Standard Deviation on a
'''
import time as t
import statistics as stats
start_time = t.time()
numbers = (12, 15, 12, 18, 20, 15, 22, 18, 15)
minimum = min(numbers)
maximum = max(numbers)
mean = stats.mean(numbers)
median = stats.median(numbers)
mode = stats.mode(numbers)
variance = stats.variance(numbers)
std_deviation = stats.stdev(numbers)
print("Minimum:", minimum)
print("Maximum:", maximum)
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
print("Variance:", variance)
print("Standard Deviation:", std_deviation)
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output :

```
Minimum: 12
Maximum: 22
Mean: 16.333333333333332
Median: 15
Mode: 15
Variance: 11.75
Standard Deviation: 3.427827300200522
Execution time: 0.0011961460113525391
PS C:\Users\rupaa\OneDrive\Desktop\AIAC>
```

4.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:

```python
#write a pyhton code with attributes teacher_id,name,subject,and experience and add method to display
import time as t
start_time = t.time()
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")
teacher1 = Teacher(1, "Alice Smith", "Mathematics", 5)
teacher1.display_details()
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
Teacher ID: 1
Name: Alice Smith
Subject: Mathematics
Experience: 5 years
Execution time: 0.0010240077972412 11
```

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.

Code:

```python
#write a python code to generate a function that validates Indian Mobile Number that number should st
import time as t
import re
start_time = t.time()
def validate_mobile_number(mobile_number):
    pattern = r'^[6-9]\d{9}$'
    if re.match(pattern, mobile_number):
        return True
    else:
        return False
mobile_number = input("Enter your mobile number: ")
if validate_mobile_number(mobile_number):
    print("Valid Indian Mobile Number")
else:
    print("Invalid Indian Mobile Number")
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
Enter your mobile number: 9014972103
Valid Indian Mobile Number
Execution time: 13.038331031799316
```

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:

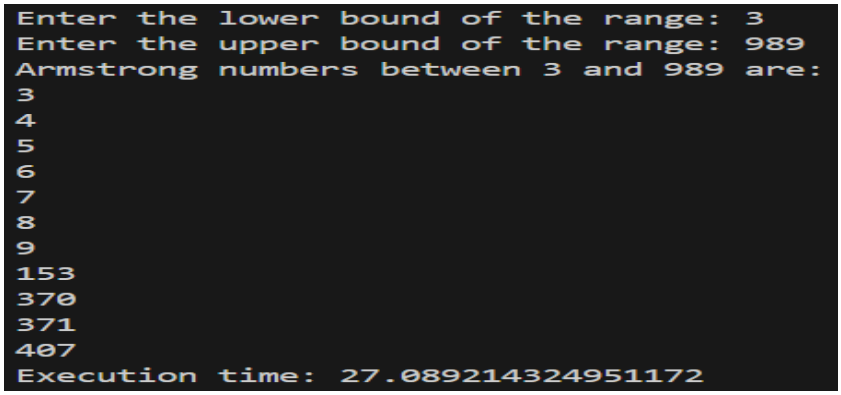
• Python program listing Armstrong numbers in the range.

• Optimized version with explanation.

Code:

```python
# write a python code that finds all Armstrong numbers in a user specified range
#Use for loop and digit power logic and validate correctness by checking known armstrong numbers
import time as t
start_time = t.time()
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number
lower_bound = int(input("Enter the lower bound of the range: "))
upper_bound = int(input("Enter the upper bound of the range: "))
print(f"Armstrong numbers between {lower_bound} and {upper_bound} are:")
for num in range(lower_bound, upper_bound + 1):
    if is_armstrong(num):
        print(num)
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:



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:

```python
#write a python code that displays all happy numbers between sepcified range
#write using loop and function and validate correctness by checking known happy numbers
import time as t
start_time = t.time()
def is_happy(number):
    seen = set()
    while number != 1 and number not in seen:
        seen.add(number)
        number = sum(int(digit) ** 2 for digit in str(number))
    return number == 1
lower_bound = int(input("Enter the lower bound of the range: "))
upper_bound = int(input("Enter the upper bound of the range: "))
print(f"Happy numbers between {lower_bound} and {upper_bound} are:")
for num in range(lower_bound, upper_bound + 1):
    if is_happy(num):
        print(num)
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
Enter the lower bound of the range: 9
Enter the upper bound of the range: 15
Happy numbers between 9 and 15 are:
10
13
```

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:

```python
#Write a Python program to print all Strong Numbers in a given range using loops.
#A Strong Number is a number whose sum of factorials of digits equals the number (example: 145).
#Validate with 1, 2, and 145.
import time as t
import math
start_time = t.time()
def is_strong(number):
    sum_of_factorials = sum(math.factorial(int(digit)) for digit in str(number))
    return sum_of_factorials == number
lower_bound = int(input("Enter the lower bound of the range: "))
upper_bound = int(input("Enter the upper bound of the range: "))
print(f"Strong numbers between {lower_bound} and {upper_bound} are:")
for num in range(lower_bound, upper_bound + 1):
    if is_strong(num):
        print(num)
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
Enter the lower bound of the range: 2
Enter the upper bound of the range: 20
Strong numbers between 2 and 20 are:
2
```

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

code:

```python
#Create a Python function that extracts Full Name, Branch, and SGPA from a nested student dictionary.
#Learn the structure using 2-3 example input dictionaries with their expected outputs, then write a reusable function that works for
#The function should correctly navigate nested dictionaries and return the required values.
import time as t
start_time = t.time()
def extract_student_info(student_dict):
    full_name = student_dict.get('personal_info', {}).get('full_name', 'N/A')
    branch = student_dict.get('academic_info', {}).get('branch', 'N/A')
    sgpa = student_dict.get('academic_info', {}).get('sgpa', 'N/A')
    return full_name, branch, sgpa
# Example student dictionaries
student1 = {
    'personal_info': {
        'full_name': 'John Doe',
        'age': 20
    },
    'academic_info': {
        'branch': 'Computer Science',
        'sgpa': 8.5
    }
}
student2 = {
    'personal_info': {
        'full_name': 'Jane Smith',
        'age': 22
    },
    'academic_info': {
        'branch': 'Mechanical Engineering',
        'sgpa': 9.0
    }
}
# Extracting information
for student in [student1, student2]:
    name, branch, sgpa = extract_student_info(student)
    print(f"Full Name: {name}, Branch: {branch}, SGPA: {sgpa}")
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:

```
Full Name: John Doe, Branch: Computer Science, SGPA: 8.5
Full Name: Jane Smith, Branch: Mechanical Engineering, SGPA: 9.0
Execution time: 0.0
```

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:

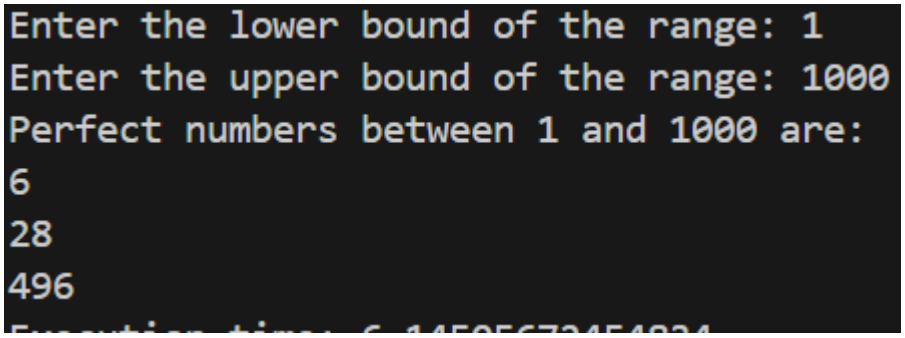
• Python program that lists Perfect Numbers in the given range.

• Optimized version with explanation.

Code:

```
#Write a Python program to print all Perfect Numbers in a given range using loops.
#A Perfect Number equals the sum of its proper divisors (example: 6, 28).
#Validate using 6, 28, 496.
#Also generate an optimized version by checking divisors only up to √n and explain briefly.
import time as t
start_time = t.time()
def is_perfect(number):
    if number < 2:
        return False
    sum_of_divisors = 1  # 1 is a proper divisor of all numbers
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            sum_of_divisors += i
            if i != number // i:
                sum_of_divisors += number // i
    return sum_of_divisors == number
lower_bound = int(input("Enter the lower bound of the range: "))
upper_bound = int(input("Enter the upper bound of the range: "))
print(f"Perfect numbers between {lower_bound} and {upper_bound} are:")
for num in range(lower_bound, upper_bound + 1):
    if is_perfect(num):
        print(num)
end_time = t.time()
print("Execution time:", end_time - start_time)
```

Output:



Explanations:

1. The program finds Automorphic numbers by checking whether the square of a number ends with the number itself using loop logic.
2. The feedback classification uses conditional statements to correctly label ratings as Negative, Neutral, or Positive based on given values.
3. The function uses Python built-in statistics methods to compute minimum, maximum, mean, median, mode, variance, and standard deviation accurately.
4. The Teacher class demonstrates object oriented programming by initializing attributes through a constructor and displaying details using a class method.
5. The function validates an Indian mobile number by checking that it has exactly ten digits and starts with six, seven, eight, or nine.
6. The program identifies Armstrong numbers by comparing each number with the sum of its digits raised to the power of total digits.

7. Happy numbers are detected by repeatedly summing the squares of digits and using a set to prevent infinite loops.
8. The function checks whether a number equals the sum of factorials of its digits to identify Strong numbers.
9. The function navigates a nested dictionary structure to correctly extract student full name, branch, and SGPA.
10. Perfect numbers are identified by summing proper divisors efficiently by checking only up to the square root of the number.