AI ASSISSTANT CODING

ASSIGNMENT -2

B. Sri Laxmi Gayathri

2303A52033

BATCH- 38

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

Task 1: Cleaning Sensor Data

❖ Scenario:

❖ You are cleaning IoT sensor data where negative values are invalid.

❖ **Task:**

Use Gemini in Colab to generate a function that filters out all negative numbers from a list.

❖ Expected Output:

➢ Before/after list

➢ Screenshot of Colab execution

PROMPT:

write a python code to extract all the negative numbers from a user given list of numbers and show the new list which contains no negative numbers

Code & output:

```python
# Get input from the user for a list of numbers
numbers_str = input("Enter a list of numbers separated by spaces: ")

# Convert the input string into a list of integers or floats
try:
    numbers = [float(num) for num in numbers_str.split()]
except ValueError:
    print("Invalid input. Please enter numbers separated by spaces.")
    numbers = [] # Set an empty list in case of error

# Extract negative numbers
negative_numbers = [num for num in numbers if num < 0]

# Create a new list without negative numbers
non_negative_numbers = [num for num in numbers if num >= 0]

print(f"Original list: {numbers}")
print(f"Negative numbers: {negative_numbers}")
print(f"List without negative numbers: {non_negative_numbers}")
```

```
Enter a list of numbers separated by spaces: 1 2 -1 3 4 -4
Original list: [1.0, 2.0, -1.0, 3.0, 4.0, -4.0]
Negative numbers: [-1.0, -4.0]
List without negative numbers: [1.0, 2.0, 3.0, 4.0]
```

Explanation :

This program takes a list of numbers from the user and checks which of them are negative.

The user first enters several numbers separated by spaces. This input is taken as a string and then converted into a list of decimal numbers using the split() and float() functions inside a try block. If the user enters anything other than numbers, the program shows an error message and uses an empty list.

After that, the program uses list comprehension to find all the negative numbers in the list. It also creates another list that contains only non-negative numbers, meaning zero and positive values.

At the end, the program displays the original list, the negative numbers, and the list after removing all negative values**.**

Task 2: String Character Analysis

❖ **Scenario:**

You are building a text-analysis feature.

❖ **Task:**

Use Gemini to generate a Python function that counts vowels, consonants,

and digits in a string.

❖ Expected Output:

➢ Working function

➢ Sample inputs and outputs

PROMPT: write a python code that counts vowels, consonants and digits in a string

Code & output:

```
def count_chars(input_string):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0
    for char in input_string:
        if char.isalpha():  # Check if the character is an alphabet
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char.isdigit():  # Check if the character is a digit
            digit_count += 1

    return vowel_count, consonant_count, digit_count
# Example usage:
my_string = "Hello World 123!"-
v, c, d = count_chars(my_string)

print(f"Original String: '{my_string}'")
print(f"Number of Vowels: {v}")
print(f"Number of Consonants: {c}")
print(f"Number of Digits: {d}")

my_string_2 = "Python is fun 4 everyone."
v2, c2, d2 = count_chars(my_string_2)
print(f"\nOriginal String: '{my_string_2}'")
print(f"Number of Vowels: {v2}")
print(f"Number of Consonants: {c2}")
print(f"Number of Digits: {d2}")
```

```
Original String: 'Hello World 123!'
Number of Vowels: 3
Number of Consonants: 7
Number of Digits: 3

Original String: 'Python is fun 4 everyone.'
Number of Vowels: 7
Number of Consonants: 12
Number of Digits: 1
```

Explanation:

This program is used to count vowels, consonants, and digits in a given string.

A function called count_chars is created, which takes a string as input. Inside the function, the program checks each character one by one. If the character is a letter, it checks whether it is a vowel or a consonant and increases the count accordingly. If the character is a number, the digit count is increased.

After checking all the characters, the function returns the total number of vowels, consonants, and digits.

In the program, two example strings are given. The function is called for each string and the results are printed.

Task 3: Palindrome Check – Tool Comparison

❖ **Scenario**:

You must decide which AI tool is clearer for string logic.

❖ **Task**: Generate a palindrome-checking function using Gemini and Copilot, then

compare the results.

❖ **Expected Output:**

➢ Side-by-side code comparison

➢ Observations on clarity and structure

PROMPT: write a python code for palindrome checker by user input

Code & output:

```
def is_palindrome(s):
    # Remove spaces and convert to lowercase for case-insensitive comparison
    s = s.replace(" ", "").lower()
    # Compare the string with its reverse
    return s == s[::-1]

# Example usage:
print(f"'madam' is a palindrome: {is_palindrome('madam')}")
print(f"'racecar' is a palindrome: {is_palindrome('racecar')}")
print(f"'A man a plan a canal Panama' is a palindrome: {is_palindrome('A man a plan a canal Panama')}")
print(f"'hello' is a palindrome: {is_palindrome('hello')}")
print(f"'Python' is a palindrome: {is_palindrome('Python')}")
```

```
'madam' is a palindrome: True
'racecar' is a palindrome: True
'A man a plan a canal Panama' is a palindrome: True
'hello' is a palindrome: False
'Python' is a palindrome: False
```

Explanation:

This program checks whether a given string is a palindrome.

The function removes spaces and converts the string to lowercase, then compares it with its reverse. If both are the same, it returns True; otherwise, it returns False.

Task 4: Code Explanation Using AI

❖ **Scenario:**

You are reviewing unfamiliar code written by another developer.

❖ **Task:**

Ask Gemini to explain a Python function (prime check OR palindrome check) line by line.

❖ **Expected Output:**

➢ Code snippet

➢ AI explanation

➢ Student comments on understanding

PROMPT: write a python code for Python function (prime check) line by line.

Code & Output:

```
def is_prime(number):
    # 1. Prime numbers are natural numbers greater than 1.
    #    Numbers less than or equal to 1 are not prime.
    if number <= 1:
        return False

    # 2. 2 is the only even prime number.
    if number == 2:
        return True

    # 3. All other even numbers greater than 2 are not prime.
    if number % 2 == 0:
        return False

    # 4. For odd numbers, we check for divisibility by odd numbers
    #    starting from 3 up to the square root of the number.
    #    If a number has a divisor, it must have one less than or equal to its square root.
    for i in range(3, int(math.sqrt(number)) + 1, 2):
        if number % i == 0:
            return False

    # 5. If no divisors are found, the number is prime.
    return True

# Example usage:
print(f"Is 1 prime? {is_prime(1)}")      # Expected: False
print(f"Is 2 prime? {is_prime(2)}")      # Expected: True
print(f"Is 7 prime? {is_prime(7)}")      # Expected: True
print(f"Is 10 prime? {is_prime(10)}")    # Expected: False
print(f"Is 11 prime? {is_prime(11)}")    # Expected: True
print(f"Is 29 prime? {is_prime(29)}")    # Expected: True
print(f"Is 30 prime? {is_prime(30)}")    # Expected: False
print(f"Is 97 prime? {is_prime(97)}")    # Expected: True
```

```
Is 1 prime? False
Is 2 prime? True
Is 7 prime? True
Is 10 prime? False
Is 11 prime? True
Is 29 prime? True
Is 30 prime? False
Is 97 prime? True
```

Explanation (Line by Line)

Let's break down the `is_prime` function:

1. `import math`:
   - This line imports the `math` module, which provides access to mathematical functions like `sqrt` (square root) that we'll use for optimization.
2. `def is_prime(number):`:
   - This defines a function named `is_prime` that takes one argument, `number`. This function will return `True` if `number` is prime, and `False` otherwise.
3. `if number <= 1:`:
   - Prime numbers are defined as natural numbers greater than 1. So, if the input `number` is 1 or less, it cannot be prime. This is the first quick check.
4. `return False`:
   - If the condition `number <= 1` is true, the function immediately returns `False` because such numbers are not prime.
5. `if number == 2:`:
   - The number 2 is the smallest and the only even prime number. It's an important special case.
6. `return True`:
   - If the `number` is exactly 2, it is prime, so the function returns `True`.
7. `if number % 2 == 0:`:
   - This checks if the `number` is an even number (i.e., its remainder when divided by 2 is 0). Since we've already handled `number == 2`, any other even number must be greater than 2 and thus cannot be prime.
8. `return False`:
   - If the `number` is an even number greater than 2, it's not prime, and the function returns `False`.
9. `for i in range(3, int(math.sqrt(number)) + 1, 2):`:
   - This is the core loop for checking odd numbers. We only need to check for divisors up to the square root of the `number` because if a number `n` has a divisor `d` greater than `sqrt(n)`, it must also have a divisor `n/d` which is less than `sqrt(n)`.
   - `range(3, ..., 2)`: The loop starts from `3` (the next odd number after 2) and increments by `2` in each step (so it only checks odd numbers). This is an optimization as we've already handled even numbers.
   - `int(math.sqrt(number)) + 1`: Calculates the integer part of the square root of `number` and adds 1 to ensure the range includes the square root itself if it's an integer.
10. `if number % i == 0:`:
    - Inside the loop, this line checks if the `number` is perfectly divisible by the current odd number `i` (i.e., the remainder is 0).
11. `return False`:
    - If `number` is divisible by any `i` in the loop, it means `number` has a factor other than 1 and itself, so it's not prime. The function returns `False` immediately.
12. `return True`:
    - If the loop completes without finding any divisors, it means the `number` has no factors other than 1 and itself (excluding the checks for 2 and even numbers already done). Therefore, the `number` is prime, and the function returns `True`.

Explanation:

This program checks whether a given number is a prime number.

The function is_prime first removes numbers less than or equal to 1. It then checks special cases like 2 and even numbers. For the remaining odd numbers, it checks divisibility from 3 up to the square root of the number. If no divisor is found, the number is considered prime.