ASSIGNMENT:03

HALLTICKET:2303A51598

BATCH:29

SHAIK NEHA FARIYAL

Task 1: generate a function that filters out all negative

numbers from a list.

PROMPT:

generate a python code for function that filters out all negative

numbers from a list.

```python
def filter_negative_numbers(input_list):

    return [num for num in input_list if num >= 0]

# Example Usage:
my_list = [10, -5, 20, 0, -15, 30, -2.5]
print("Original list:", my_list)

filtered_list = filter_negative_numbers(my_list)
print("Filtered list (non-negative numbers only):", filtered_list)

# Using existing data from the kernel
print("\nUsing 'sensor_readings' from previous execution:", sensor_readings)
filtered_sensor_readings = filter_negative_numbers(sensor_readings)
print("Filtered sensor readings:", filtered_sensor_readings)
```

OUTPUT:

```
Original list: [10, -5, 20, 0, -15, 30, -2.5]
Filtered list (non-negative numbers only): [10, 20, 0, 30]

Using 'sensor_readings' from previous execution: [23.5, 24.1, -5.0, 22.9, 25.0, -1.2, 23.8]
Filtered sensor readings: [23.5, 24.1, 22.9, 25.0, 23.8]
```

JUSTIFICATION:

The first task involved removing negative values from IoT sensor data. Since real-world sensor data often contains invalid or noisy values, data cleaning is an essential preprocessing step. Gemini generated an efficient function using list comprehension to filter negative numbers. This shows that AI tools can quickly produce optimized and readable code for data preprocessing, reducing manual effort and minimizing errors.

Task 2: String Character Analysis

PROMPT:

generate a Python function that counts vowels, consonants,

and digits in a string..

```
[8]
✓ 0s
def count_chars(input_string):

    vowels = "aeiouAEIOU"
    consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    digits = "0123456789"

    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        if char in vowels:
            vowel_count += 1
        elif char in consonants:
            consonant_count += 1
        elif char in digits:
            digit_count += 1

    return {
        "vowels": vowel_count,
        "consonants": consonant_count,
        "digits": digit_count
    }
```

```
# Example Usage:
my_string1 = "Hello World! 123"
counts1 = count_chars(my_string1)
print(f"String: '{my_string1}'\nCounts: {counts1}")

my_string2 = "Python Programming 2023"
counts2 = count_chars(my_string2)
print(f"\nString: '{my_string2}'\nCounts: {counts2}")

my_string3 = "AI & ML"
counts3 = count_chars(my_string3)
print(f"\nString: '{my_string3}'\nCounts: {counts3}")
```

OUTPUT:

```
...  String: 'Hello World! 123'
     Counts: {'vowels': 3, 'consonants': 7, 'digits': 3}

     String: 'Python Programming 2023'
     Counts: {'vowels': 4, 'consonants': 13, 'digits': 4}

     String: 'AI & ML'
     Counts: {'vowels': 2, 'consonants': 2, 'digits': 0}
```

JUSTIFICATION:

This task focused on counting vowels, consonants, and digits in a string, which is important in text analysis and input validation. Gemini generated a structured function using built-in string methods. This demonstrates the ability of AI tools to handle conditional logic correctly and generate reusable functions. The solution highlights how AI can support both learning and practical text-processing tasks.

Task 3: Palindrome Check – Tool Comparison

PROMPT:

Generate a palindrome-checking function using Gemini and Copilot, then compare the results.

Gemini:

```
[9]
✓ 0s
def is_palindrome(input_string):
    # 2. Convert the input_string to lowercase
    cleaned_string = ""
    for char in input_string:
        # 3. Create a new string containing only the alphanumeric characters
        #    from the lowercase string.
        if char.isalnum():
            cleaned_string += char.lower()

    # 4. Compare this cleaned string with its reversed version.
    # 5. Return True if they are identical, and False otherwise.
    return cleaned_string == cleaned_string[::-1]

# Example Usage:
print(f"'Racecar' is a palindrome: {is_palindrome('Racecar')}")
print(f"'Madam' is a palindrome: {is_palindrome('Madam')}")
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"'No lemon, no melon' is a palindrome: {is_palindrome('No lemon, no melon')}")
print(f"'12321' is a palindrome: {is_palindrome('12321')}")
print(f"'12345' is a palindrome: {is_palindrome('12345')}")
```

Copilot:

```
1  #Generate a palindrome-checking function the compare the results.
2  def is_palindrome(s):
3      # Remove spaces and convert to lowercase for uniformity
4      s = s.replace(" ", "").lower()
5      # Check if the string is equal to its reverse
6      return s == s[::-1]
7  # Test cases
8  test_strings = ["racecar", "hello", "A man a plan a canal Panama", "Python"]
9  for string in test_strings:
10     result = is_palindrome(string)
11     print(f'"{string}" is a palindrome: {result}')
12
```

OUTPUT:

Gemini:

```
...  'Racecar' is a palindrome: True
     'Madam' is a palindrome: True
     'A man, a plan, a canal: Panama' is a palindrome: True
     'Hello' is a palindrome: False
     'No lemon, no melon' is a palindrome: True
     '12321' is a palindrome: True
     '12345' is a palindrome: False
```

Copilot:

```
PS C:\Users\nehaf\OneDrive\Documents\ps lab> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Documents/ps lab/assignment02.py"
"racecar" is a palindrome: True
"hello" is a palindrome: False
"A man a plan a canal Panama" is a palindrome: True
"Python" is a palindrome: False
PS C:\Users\nehaf\OneDrive\Documents\ps lab>
```

JUSTIFICATION:

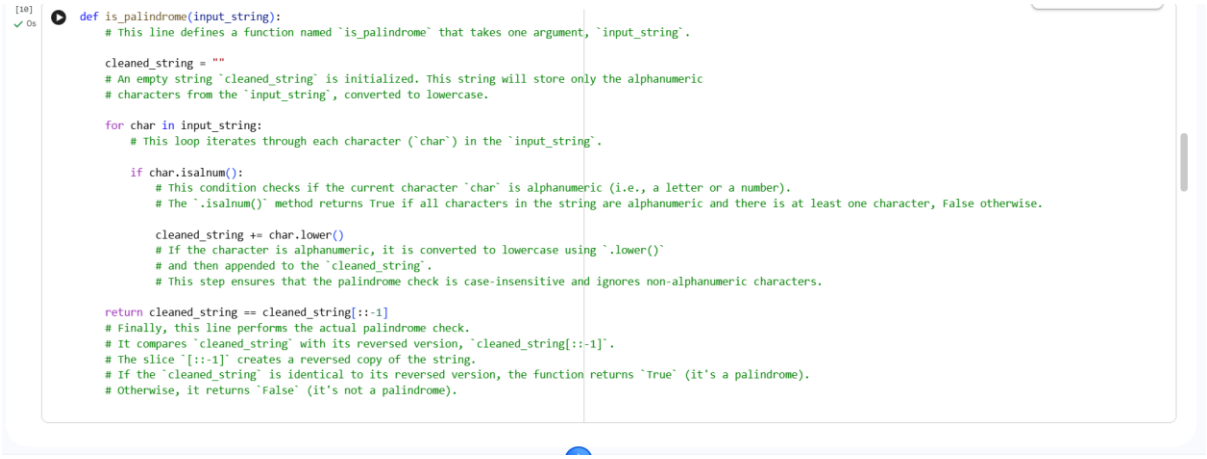
In this task, palindrome-checking functions were generated using Gemini and Copilot. Gemini provided a short and optimized solution, while Copilot produced a more detailed, step-by-step approach. This comparison justified that different AI tools offer different strengths: Gemini is effective for concise professional code, whereas Copilot is helpful for understanding logic clearly. This task emphasized the importance of comparing AI tools to choose the most suitable one.

Task 4: Code Explanation Using AI

PROMPT:

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

CODE:



```
[10]
✓ 0s
def is_palindrome(input_string):
    # This line defines a function named `is_palindrome` that takes one argument, `input_string`.

    cleaned_string = ""
    # An empty string `cleaned_string` is initialized. This string will store only the alphanumeric
    # characters from the `input_string`, converted to lowercase.

    for char in input_string:
        # This loop iterates through each character (`char`) in the `input_string`.

        if char.isalnum():
            # This condition checks if the current character `char` is alphanumeric (i.e., a letter or a number).
            # The `.isalnum()` method returns True if all characters in the string are alphanumeric and there is at least one character, False otherwise.

            cleaned_string += char.lower()
            # If the character is alphanumeric, it is converted to lowercase using `.lower()`
            # and then appended to the `cleaned_string`.
            # This step ensures that the palindrome check is case-insensitive and ignores non-alphanumeric characters.

    return cleaned_string == cleaned_string[::-1]
    # Finally, this line performs the actual palindrome check.
    # It compares `cleaned_string` with its reversed version, `cleaned_string[::-1]`.
    # The slice `[::-1]` creates a reversed copy of the string.
    # If the `cleaned_string` is identical to its reversed version, the function returns `True` (it's a palindrome).
    # Otherwise, it returns `False` (it's not a palindrome).
```

JUSTIFICATION:

This task demonstrated Gemini's ability to explain unfamiliar code line by line. The explanation of the prime-checking function helped in understanding both logic and optimization. This justifies the role of AI not only as a coding assistant but also as a learning support tool.