

# Ai-Lab Assignment-2.2

K.SAI ROHAN

Batch 38  
2303A52025

## Task 1: Cleaning Sensor Data

Prompt (Gemini) Generate Python code to remove negative values from a list

```
[ ]: # Original list with negative values
my_list = [1, -2, 3, -4, 5, 0, -10, 100]

# Using a list comprehension to filter out negative values
filtered_list = [x for x in my_list if x >= 0]

print("Original list:", my_list)
print("List after removing negative values:", filtered_list)

Original list: [1, -2, 3, -4, 5, 0, -10, 100]
List after removing negative values: [1, 3, 5, 0, 100]
```

### 0.0.1 Explanation:

1. `my_list = [1, -2, 3, -4, 5, 0, -10, 100]` line initializes a list named `my_list` containing both positive, negative, and zero values.
2. `filtered_list = [x for x in my_list if x >= 0]` is a list comprehension that iterates through each element `x` in `my_list`.
  - The `if x >= 0` condition checks if the current element `x` is greater than or equal to 0 (i.e., non-negative).
  - If the condition is true, `x` is included in the new list `filtered_list`.
3. The `print` statements display both the original and the filtered lists.

## Task 2: String Character Analysis

Prompt (Gemini) Write a Python function to count vowels, consonants, and digits in a string

```
[ ]: 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:
text = "Hello World 123!"
v, c, d = count_chars(text)

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

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

## 0.0.2 Explanation of the `count_chars` function:

1. `def count_chars(input_string)`: defines a function named `count_chars` that takes one argument, `input_string`.
2. `vowels = "aeiouAEIOU"`: string containing all lowercase and uppercase vowels for easy checking.
3. `vowel_count, consonant_count, digit_count = 0`: initialize counters to zero.
4. `for char in input_string`: the function iterates through each character in the input string.
5. `if char.isalpha():`: checks if the current character is an alphabet (letter).
  - `if char in vowels:`: if it's an alphabet, it further checks if it's present in the string. If so, `vowel_count` is incremented.
  - `else:`: If it's an alphabet but not a vowel, it's a consonant, so `consonant_count` is incremented.

6. **elif char.isdigit():** If the character is not an alphabet, it checks if it's a digit. If it is, digit\_count is incremented.
7. **return vowel\_count, consonant\_count, digit\_count:** The function returns the three counts as a tuple.
8. The example usage demonstrates how to call the function and print the results.

### Task 3: Palindrome Check - Tool Comparison

Prompt (Gemini) i] Create a Python function to check whether a string is a palindrome.

```
[ ]: import re

def is_palindrome(input_string):

    # Convert to lowercase and remove non-alphanumeric characters
    cleaned_string = re.sub(r'[^a-zA-Z0-9]', '', input_string).lower()

    # Compare the cleaned string with its reverse
    return cleaned_string == cleaned_string[::-1]

# Example usage:
string1 = "madam"
string2 = "A man, a plan, a canal: Panama"
string3 = "hello" string4 = "Racecar"

print(f"'{string1}' is a palindrome: {is_palindrome(string1)}")
print(f"'{string2}' is a palindrome: {is_palindrome(string2)}")
print(f"'{string3}' is a palindrome: {is_palindrome(string3)}")
print(f"'{string4}' is a palindrome: {is_palindrome(string4)}")
```

'madam' is a palindrome: True  
'A man, a plan, a canal: Panama' is a palindrome: True  
'hello' is a palindrome: False  
'Racecar' is a palindrome: True

#### 0.0.3 Explanation of the `is_palindrome` function:

1. **import re:** Imports the regular expression module, which is used for cleaning the string.
2. **def is\_palindrome(input\_string):** Defines a function named `is_palindrome` that takes one argument, `input_string`.
3. **cleaned\_string = re.sub(r'[^a-zA-Z0-9]', '', input\_string).lower()**
  - `re.sub(r'[^a-zA-Z0-9]', '', input_string)` part uses a regular expression to remove any character that is NOT an alphanumeric character (letters A-Z, numbers 0-9) from the `input_string`. It replaces them with an empty string.
  - `.lower()` Converts the resulting string to lowercase. This ensures that the comparison is case-insensitive (e.g., 'Racecar' and 'racecar' are treated the same).
4. **return cleaned\_string == cleaned\_string[::-1]**
  - `cleaned_string[::-1]` creates a reversed version of the `cleaned_string` using string slicing. `[:: -1]` is a Python idiom for reversing a sequence.

- The function then compares the cleaned string with its reversed counterpart. If they are identical, the string is a palindrome, and the function returns True; otherwise, it returns False.

5. The example usage demonstrates how to use the function with different strings and prints the results.

### Task 3: Palindrome Check - Tool Comparison

Prompt (Copilot) ii] Write a Python palindrome checking function

```
[ ]: def is_palindrome(text: str) -> bool:
    """
    Returns True if 'text' is a palindrome, False otherwise.
    """
```

```
Palindrome check ignores case and non-alphanumeric characters.
"""

# Normalize: lowercase and keep only letters/numbers
cleaned = ''.join(c.lower() for c in text if c.isalnum())
# Compare with its reverse
return cleaned == cleaned[::-1]

# Examples
print(is_palindrome("madam"))           # True
print(is_palindrome("RaceCar"))          # True
print(is_palindrome("A man, a plan, a canal: Panama")) # True
print(is_palindrome("hello"))            # False
```

#### 0.0.4 Explanation of the is\_palindrome function:

- def is\_palindrome(text: str) -> bool: This line defines a function named is\_palindrome that takes one argument, text, which is type-hinted as a string (str), and is expected to return a boolean value (bool).
- """ Returns True if 'text' is a palindrome, False otherwise. Palindrome check ignores case and non-alphanumeric characters. """ This is a docstring. It provides a concise summary of the function's purpose, explaining that it checks if a string is a palindrome while ignoring case and non-alphanumeric characters.
- cleaned = ''.join(c.lower() for c in text if c.isalnum()) This line creates a 'cleaned' version of the input string. It uses a generator expression within a join method:
  - for c in text: It iterates through each character in the input text.
  - if c.isalnum(): It filters characters, keeping only those that are alphanumeric (letters or numbers). Non-alphanumeric characters (like spaces, punctuation, etc.) are discarded.
  - c.lower(): For each retained character, it converts it to its lowercase equivalent. This makes the palindrome check case-insensitive.
  - '''.join(...): All the processed (alphanumeric and lowercase) characters are then joined together to form a new string, which is assigned to the variable cleaned.
- return cleaned == cleaned[::-1] This line performs the actual palindrome check and returns the result:

- `cleaned[::-1]`: This creates a reversed version of the string using string slicing. `[::-1]` is a common Python idiom to reverse sequences.
- `cleaned == cleaned[::-1]`: compares the cleaned string with its reversed counterpart. If they are identical, the string is a palindrome, and this comparison evaluates to `True`. Otherwise, it evaluates to `False`. This boolean result is then returned by the function.

#### Task 4: Code Explanation Using AI

Prompt (Gemini) Explain the given Python function line by line and check

```
[ ]: import math

def is_prime(number):
    """
    Checks if a given number is prime.
    A prime number is a natural number greater than 1 that has no positive_
    divisors other than 1 and itself.
    """
    if number <= 1:
        return False
    if number == 2:
        return True
    if number % 2 == 0:
        return False

    # Check for divisors from 3 up to the square root of the number
    # only considering odd numbers
    for i in range(3, int(math.sqrt(number)) + 1, 2):
        if number % i == 0:
            return False

    return True

# Example usage:
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 0 prime? {is_prime(0)}")
print(f"Is 17 prime? {is_prime(17)}")
print(f"Is 25 prime? {is_prime(25)}")
```

#### 0.0.5 Explanation of the `is_prime` function line by line:

1. `import math`: imports the `math` module, which is necessary for using the `sqrt()` function.

2. `def is_prime(number):`: Defines a function named `is_prime` that takes one argument, `number`.
3. `"""Checks if a given number is prime .....` T"his" is a docstring, which provides a brief explanation of what the function does.
4. `if number <= 1:` : Prime numbers are defined as natural numbers greater than 1. If the number is 1 or less, it cannot be prime.
5. `return False`: If the condition in the previous line is true, the function immediately returns `False`.
6. `if number == 2:` : The number 2 is the only even prime number.
7. `return True`: If the number is 2, the function immediately returns `True`.
8. `if number % 2 == 0:` : After checking for 2, any other even number (which would have a remainder of 0 when divided by 2) cannot be prime.
9. `return False`: If the number is an even number greater than 2, the function immediately returns `False`.
10. `for i in range(3, int(math.sqrt(number)) + 1):` : The loop checks for odd divisors starting from 3 up to the square root of the number.
  - `range(3, ..., 2)` Starts checking from 3 and increments by 2 in each step, efficiently skipping even numbers since we already handled them.
  - `int(math.sqrt(number)) + 1` We only need to check for divisors up to the square root of the number. If a number has a divisor larger than its square root, it must also have a divisor smaller than its square root. `+ 1` ensures that the square root itself is included in the range if it's an integer.
11. `if number % i == 0:` : Inside the loop, this checks if the number is perfectly divisible by (i.e., the remainder is 0).
12. `return False`: If any such divisor is found, the number is not prime, and the function returns `False`.
13. `return True`: If the loop completes without finding any divisors, it means the number has no divisors other than 1 and itself, so it is prime, and the function returns `True`.

#### Task 4: Code Explanation Using AI

Prompt (Gemini) ii] Explain the given Python function line by line.

```
[ ]: def is_palindrome(text: str) -> bool:
    """
    Returns True if 'text' is a palindrome, False otherwise.
    Palindrome check ignores case and non-alphanumeric characters.
    """
    # Normalize: lowercase and keep only letters/numbers cleaned
    = ''.join(c.lower() for c in text if c.isalnum())
    # Compare with its reverse
    cleaned == cleaned[::-1]

# Test cases for the is_palindrome function
print(f"'madam' is a palindrome: {is_palindrome('madam')}")
print(f"'RaceCar' is a palindrome: {is_palindrome('RaceCar')}
```

```
{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"'Was it a car or a cat I saw?' is a
palindrome: {is_palindrome('Was it_
·a car or a cat I saw?')}") 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"'Python' is a palindrome:
{is_palindrome('Python')}")
```

#### 0.0.6 Explanation of the test code:

This code block redefines `is_palindrome` function (Copilot's version) to ensure it's available for testing. Then, it calls the function with several example strings\*: 'madam', 'RaceCar', 'A man, a plan, a canal: Panama', 'Was it a car or a cat I saw?', 'No lemon, no melon', and '12321'. These strings are expected to be palindromes, so their results should be True. '\*' 'hello' and 'Python' are not palindromes, so their results should be False.

The `print` statements display each test string along with the boolean result from the `is_palindrome` function, making it easy to verify its correctness.