

School of Computer Science and Artificial Intelligence

Lab Assignment 2.2

Program	: B. Tech (CSE)
Course Title	: AI Assisted Coding
Course Code	: 23CS002PC304
Semester	: III
Academic Session	: 2025-2026
Name of Student	: Sruthi
Enrollment No.	: 2403A51L10
Batch No.	: 51
Date	: 13-01-2026

Task 1

Prompt: Write a python program to that filters out all negative numbers from a list using function

Code:

```
def filter_negative_numbers(numbers):  
    """  
    Filters out negative numbers from a list.  
  
    Args:  
        numbers: A list of numbers.  
  
    Returns:  
        A new list containing only non-negative numbers.  
    """  
    non_negative_numbers = [num for num in numbers if num >= 0]  
    return non_negative_numbers  
my_list = [1, -2, 3, -4, 0, 5, -6]  
filtered_list = filter_negative_numbers(my_list)  
  
print(f"Original list: {my_list}")  
print(f"Filtered list (non-negative numbers): {filtered_list}")
```

Output:

```
Original list: [1, -2, 3, -4, 0, 5, -6]  
Filtered list (non-negative numbers): [1, 3, 0, 5]
```

Explanation:

This code defines a function `filter_negative_numbers` that takes a list of numbers as input and returns a new list containing only the non-negative numbers.

1. **`def filter_negative_numbers(numbers):`**: This line defines a function named `filter_negative_numbers` that accepts one argument, `numbers`, which is expected to be a list.

2. **non_negative_numbers = [num for num in numbers if num >= 0]**: This is a list comprehension. It iterates through each num in the input numbers list. For every num that is greater than or equal to 0 (i.e., non-negative), it includes that num in a new list called non_negative_numbers.
3. **return non_negative_numbers**: The function then returns this newly created list containing only the non-negative numbers.

Task 2:

Prompt: Write a python program that counts vowels, consonants, and digits in a string.

Code:

```
def count_characters(input_string):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0
    for char in input_string:
        if char.isalpha(): # Check if it's an alphabet character
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char.isdigit(): # Check if it's a digit
            digit_count += 1
    return vowel_count, consonant_count, digit_count

my_string = "Hello World 123!"
v, c, d = count_characters(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}")
```

Output:

```
... Original string: 'Hello World 123!'
    Number of vowels: 3
    Number of consonants: 7
    Number of digits: 3
```

Explanation:

It defines a function count_characters(input_string) that takes a string as input.

1. **Initialization:** It sets up vowels (a string of all lowercase and uppercase vowels), and initializes vowel_count, consonant_count, and digit_count to zero.
2. **Iteration:** It loops through each character in the input_string.
3. **Character Classification:** For each character:
 - o It checks if the character is an alphabet letter using char.isalpha().
 - If it's an alphabet letter, it then checks if it's present in the vowels string. If yes, vowel_count increments; otherwise, consonant_count increments.

- If it's not an alphabet letter, it checks if the character is a digit using `char.isdigit()`. If yes, `digit_count` increments.
- 4. **Return Value:** Finally, the function returns a tuple containing the total `vowel_count`, `consonant_count`, and `digit_count`.

Task 3:

Prompt: Write a python program that checks the string is palindrome are not using functions.

Colab Gemini Code:

```
def is_palindrome(s):
    """
    Checks if a given string is a palindrome.
    A palindrome is a word, phrase, or sequence that reads
    the same forwards and backward.
    Args:
        s: The input string.
    Returns:
        True if the string is a palindrome, False otherwise.
    """
    # Convert to lowercase and remove non-alphanumeric characters
    # to handle cases like 'Madam' or 'A man, a plan, a canal: Panama.'
    processed_s = ''.join(char.lower() for char in s if char.isalnum())
    # Compare the processed string with its reverse
    return processed_s == processed_s[::-1]

# Example usage:
string1 = "madam"
string2 = "hello"
string3 = "A man, a plan, a canal: Panama"
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)}')
```

Copilot Code

```
def is_palindrome(s):
    # Remove spaces and convert to lowercase for uniformity
    s = s.replace(" ", "").lower()
    # Check if the string is equal to its reverse
    return s == s[::-1]

# Test the function
input_string = "A man a plan a canal Panama"
if is_palindrome(input_string):
    print(f'"{input_string}" is a palindrome.')
else:
    print(f'"{input_string}" is not a palindrome.')
```

Colab Gemini Output:

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

Copilot Output:

```
"A man a plan a canal Panama" is a palindrome.
```

Explanation:

The main difference between the two programs is how they preprocess the input string. The first program removes all non-alphanumeric characters, making it more accurate for checking palindromes with punctuation and special characters. The second program removes only spaces, which makes it simpler but less reliable for complex strings.

Task 4:

Prompt: Explain a python function palindrome check line by line.

Let's break down the `is_palindrome` function line by line:

```
def is_palindrome(text):
```

This line defines a function named `is_palindrome` that takes one argument, `text`, which is expected to be a string. This function will return `True` if the `text` is a palindrome and `False` otherwise.

```
    cleaned_text = text.lower()
```

This line converts the input `text` to all lowercase letters. This is important because palindromes are typically case-insensitive (e.g., "Madam" should be considered the same as "madam"). The result is stored in a new variable `cleaned_text`.

```
    cleaned_text = ''.join(char for char in cleaned_text if char.isalnum())
```

This line further processes `cleaned_text`. It uses a generator expression within `join()` to iterate through each `char` in the `cleaned_text`. The `if char.isalnum()` condition checks if the character is alphanumeric (a letter or a number). Only alphanumeric characters are included, effectively removing spaces, punctuation, and other special characters. This ensures that phrases like "A man, a plan, a canal: Panama" are correctly identified as palindromes. The filtered characters are then joined back into a single string, overwriting the `cleaned_text` variable.

```
    return cleaned_text == cleaned_text[::-1]
```

This is the core of the palindrome check. It compares the `cleaned_text` with its reverse.

- `cleaned_text[::-1]` creates a reversed copy of the `cleaned_text` using slicing. The `[::-1]` slice notation means: start from the beginning, go to the end, and step backwards by 1.
- The `==` operator then compares the original `cleaned_text` with its reversed version. If they are identical, the expression evaluates to `True`, meaning the text is a palindrome; otherwise, it evaluates to `False`. This boolean result is then returned by the function.

```
# Example usage:
print(f"'madam' is a palindrome: {is_palindrome('madam')}")
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')}")
```

These lines demonstrate how to call the `is_palindrome` function with different strings and print the results using f-strings for clear output.

Comments:

The explanation clearly describes how the `is_palindrome` function works step by step. Each line of code is explained in an easy and understandable way. The reasons for converting the text to lowercase and removing special characters are clearly mentioned. The explanation of reversing the string using slicing is simple and clear. Overall, the explanation shows a good understanding of how the palindrome logic works in Python.