

ASSIGNMENT:2.2

NAME-ARSHA VARDHINI

ROLL:NO:2303A51600

BATCH-29

TASK-1

PROMPT: generate a function that filters out all negative numbers from a list .(first generate list)

CODE:

The screenshot shows a Jupyter Notebook interface with two code cells. Cell [1] contains code to generate a list of 10 random integers, including positive, negative, and zero. Cell [2] contains a function definition for filtering out negative numbers. A sidebar on the right shows an AI conversation where the user asks for a function to filter out negative numbers from a list, and the AI responds by generating a sample list and defining the function. The AI interface includes buttons for 'Accept & Run', 'Accept', and 'Cancel'.

```
# Generate a list of 10 random integers, including positive, negative, and zero
numbers = [random.randint(-10, 10) for _ in range(10)]
print("Original list:", numbers)

Original list: [-5, 9, 5, -10, -7, -7, 8, 1, -10, -4]

def filter_negative_numbers(input_list):
    """
    Filters out all negative numbers from a given list.

    Args:
        input_list (list): The list of numbers to filter.

    Returns:
        list: A new list containing only non-negative numbers.
    """
    return [num for num in input_list if num >= 0]

# Apply the function to the generated list
filtered_numbers = filter_negative_numbers(numbers)
print("Filtered list (non-negative numbers only):", filtered_numbers)
```

OBSERVATION:

The program uses Python random module to generate a list of ten integers ranging from minus ten to ten so the list may contain negative numbers zero and positive numbers

The original list is printed first showing all randomly generated values

A function named filter_negative_numbers is defined to remove negative numbers from the list

Inside the function list comprehension is used with the condition num greater than or equal to zero which keeps only zero and positive values

The function does not modify the original list instead it returns a new filtered list

The filtered list output contains only non negative numbers proving that the filtering logic works correctly

TASK-2

PROMPT: generate a Python function that counts vowels, consonants, and digits in a string

CODE:

```
Untitled28.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
RAM Disk
[3] 0s
def count_chars(my_string):
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in my_string:
        if char in 'AEIOUaeiou':
            vowel_count += 1
        elif char in 'BCDFGHbcdfgh':
            consonant_count += 1
        elif char.isdigit():
            digit_count += 1

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

# Example usage:
my_string = "Hello World 123!"
counts = count_chars(my_string)
print(f"The string '{my_string}' contains:")
print(f" Vowels: {counts['vowels']}")
print(f" Consonants: {counts['consonants']}")
print(f" Digits: {counts['digits']}")
```

generate a Python function that counts vowels, consonants, and digits in a string.

Certainly! Here's a Python function that counts vowels, consonants, and digits in a given string, along with an example of how to use it.

Accept & Run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

OBSERVATION:

The program defines a function named `count_chars` that analyzes a given string

It checks each character in the string one by one using a loop

If the character is an alphabet it further checks whether it is a vowel or a consonant

Vowels are identified using a predefined string containing both uppercase and lowercase vowels

If the character is a digit it increases the digit count

Special characters spaces and symbols are ignored

The function keeps separate counters for vowels consonants and digits

It returns the final counts in the form of a dictionary

The example string Hello World 123 is passed to the function

The output correctly displays the number of vowels consonants and digits present in the string

TASK-3

PROMPT: Generate a palindrome-checking function and compare the results.

CODE:

```
Untitled28.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[4] 0s
def is_palindrome(text):
    """
    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    # Remove non-alphanumeric characters and convert to lowercase
    processed_text = ''.join(char.lower() for char in text if char.isalnum())
    
    # Compare the processed string with its reverse
    return processed_text == processed_text[::-1]

# Test cases
print("Palindrome checker results:")
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"'racecar' is a palindrome: {is_palindrome('racecar')}")
print(f"'hello' is a palindrome: {is_palindrome('hello')}")
print(f"'Python' is a palindrome: {is_palindrome('Python')}")
print(f"'No lemon, no melon' is a palindrome: {is_palindrome('No lemon, no melon')}")

... Palindrome checker results:
'madam' is a palindrome: True
'A man, a plan, a canal: Panama' is a palindrome: True
'racecar' is a palindrome: True
'hello' is a palindrome: False
'Python' is a palindrome: False
'No lemon, no melon' is a palindrome: True
```

Generate a palindrome-checking function and compare the results.
Okay! I'll generate a Python function to check if a string is a palindrome. Then, I'll show you how it works with a few examples.

Accept & Run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

OBSERVATION:

The program defines a function named `is_palindrome` that checks whether a given string is a palindrome

It first removes all non-alphanumeric characters such as spaces punctuation and symbols

All remaining characters are converted to lowercase to ensure case insensitive comparison

The cleaned string is then compared with its reverse using slicing
If both strings are the same the function returns true otherwise it returns false

This approach allows phrases and sentences to be checked correctly not just single words

TASK-4

PROMPT: Explain a Python function (prime check OR palindrome check) line by line.

CODE:

The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar says "Untitled28.ipynb". The main area contains the following Python code:

```
def is_palindrome(text):
    """A function that takes a string and returns True if it is a palindrome, False otherwise.
    A palindrome is a word that reads the same backward as forward, e.g., madam or racecar.
    Args:
        text (str): The string to check.
    Returns:
        bool: True if text is a palindrome, False otherwise.
    """
    processed_text = ''.join(char.lower() for char in text if char.isalnum())
    return processed_text == processed_text[::-1]
```

Below the code, the Gemini AI interface provides an explanation:

- Line 1: `def is_palindrome(text):`
 - This line defines a function named `is_palindrome` that takes one argument: `text` (which is expected to be a string).
- Line 2: `"""..."""`
 - This is a docstring, which provides a brief description of what the function does, its arguments (`Args`), and what it returns (`Returns`). It's good practice for documenting code.
- Line 3: `processed_text = ''.join(char.lower() for char in text if char.isalnum())`
 - This is the core of the pre-processing step:
 - for char in text : It iterates through each character in the input `text` string.

At the bottom, there are buttons for "Explain the 'filter_negative_numbers' function line by line", "Test the 'is_palindrome' function with custom input", and "Generate a Python function to count words in a string".

OBSERVATION:

Explanation:

```
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 is a docstring, which provides a brief description of what the function does, its arguments (Args), and what it returns (Returns). It's good practice for documenting code.

```
processed_text = ''.join(char.lower() for char in text if  
char.isalnum())
```

This is the core of the pre-processing step:

`for char in text:` It iterates through each character in the input text string.

`if char.isalnum():` It checks if the character is alphanumeric (i.e., a letter or a number). This effectively filters out spaces, punctuation, and other symbols.

`char.lower():` For each alphanumeric character, it converts it to lowercase. This ensures that case doesn't affect the palindrome check (e.g., 'Racecar' should be considered a palindrome).

`'.join(...):` It concatenates all the filtered and lowercased characters back into a single string. The result is stored in the `processed_text` variable.

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

This line performs the actual palindrome check and returns the result:

`processed_text[::-1]`: This creates a reversed version of the `processed_text` string using Python's slicing notation. `[::-1]` means start from the end, go to the beginning, with a step of `-1`.

`processed_text == processed_text[::-1]`: It compares the `processed_text` with its reversed version. If they are identical, the expression evaluates to `True`, indicating it's a palindrome. Otherwise, it evaluates to `False`.

`return`: The Boolean `result` (`True` or `False`) is returned by the function.