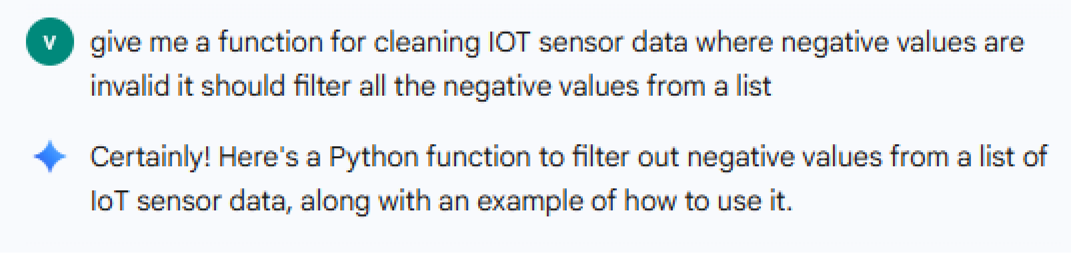
2303A51449
Batch 03

ASSIGNMENT 02 (LAB 2.2)

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.

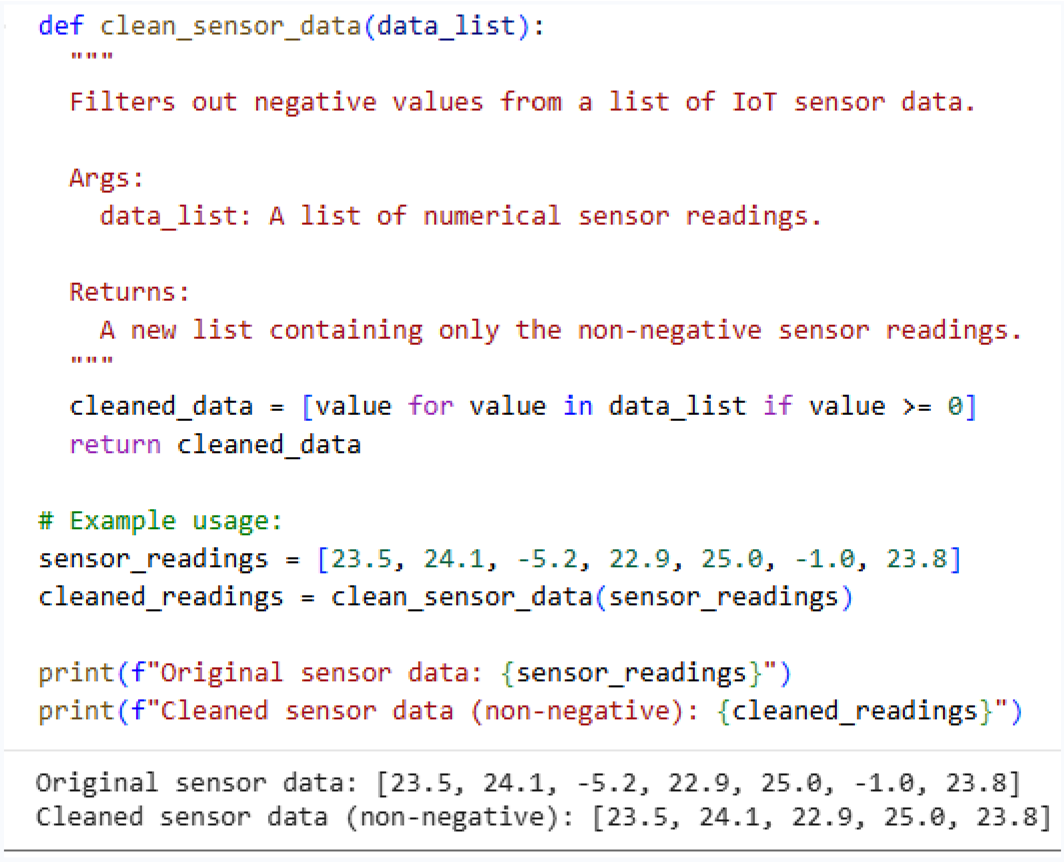
Prompt : give me a function for cleaning IOT sensor data where negative values are invalid it should filter all the negative values from a list



give me a function for cleaning IOT sensor data where negative values are invalid it should filter all the negative values from a list

Certainly! Here's a Python function to filter out negative values from a list of IoT sensor data, along with an example of how to use it.

Code and Output:

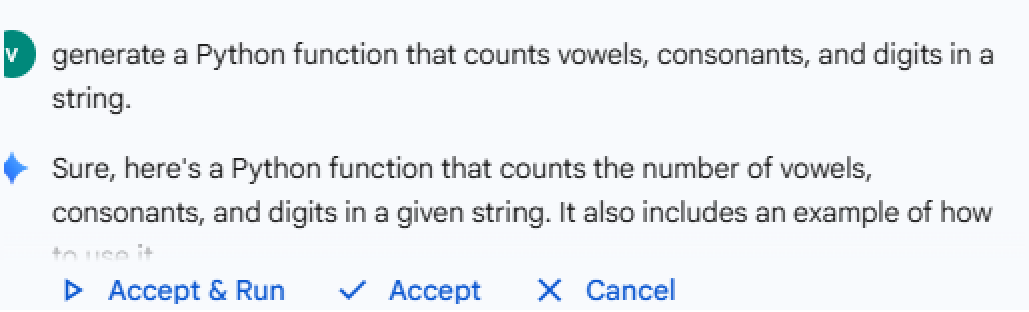


```
def clean_sensor_data(data_list):
  """
  Filters out negative values from a list of IoT sensor data.

  Args:
    data_list: A list of numerical sensor readings.

  Returns:
    A new list containing only the non-negative sensor readings.
  """
  cleaned_data = [value for value in data_list if value >= 0]
  return cleaned_data

# Example usage:
sensor_readings = [23.5, 24.1, -5.2, 22.9, 25.0, -1.0, 23.8]
cleaned_readings = clean_sensor_data(sensor_readings)

print(f"Original sensor data: {sensor_readings}")
print(f"Cleaned sensor data (non-negative): {cleaned_readings}")
```

```
Original sensor data: [23.5, 24.1, -5.2, 22.9, 25.0, -1.0, 23.8]
Cleaned sensor data (non-negative): [23.5, 24.1, 22.9, 25.0, 23.8]
```

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.

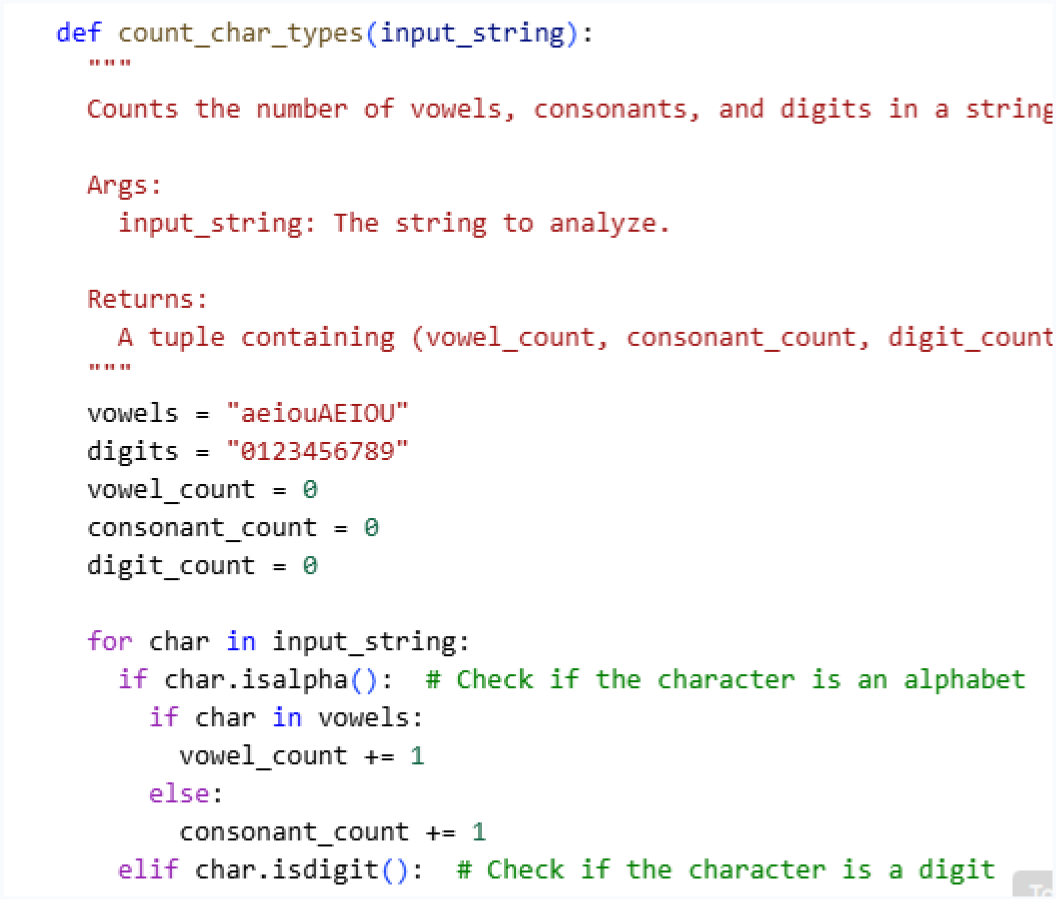
Prompt : generate a Python function that counts vowels, consonants, and digits in a string.



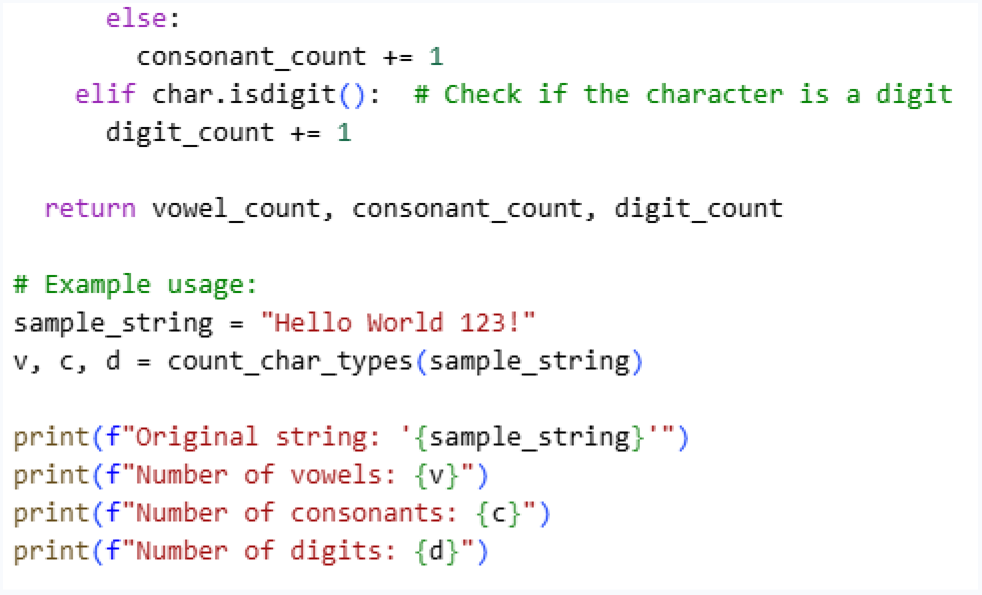
generate a Python function that counts vowels, consonants, and digits in a string.

Sure, here's a Python function that counts the number of vowels, consonants, and digits in a given string. It also includes an example of how to use it

CODE & OUTPUT:

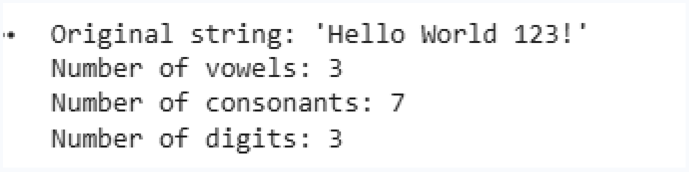


```
def count_char_types(input_string):
  """
  Counts the number of vowels, consonants, and digits in a string

  Args:
    input_string: The string to analyze.

  Returns:
    A tuple containing (vowel_count, consonant_count, digit_count
  """
  vowels = "aeiouAEIOU"
  digits = "0123456789"
  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
```



```
      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:
sample_string = "Hello World 123!"
v, c, d = count_char_types(sample_string)

print(f"Original string: '{sample_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
```

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.

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

CODE:

```
import re


def is_palindrome_gemini(input_string):
  """
  Checks if a string is a palindrome, ignoring case and
non-alphanumeric characters.
  This function is designed to mimic a style that might be generated by
a large language model like Gemini.

  Args:
    input_string: The string to check.

  Returns:
    True if the string is a palindrome, False otherwise.
  """
  # Convert to lowercase and remove non-alphanumeric characters
  cleaned_string = re.sub(r'[^a-zA-Z0-9]', '', input_string).lower()

  # Check if the cleaned string is equal to its reverse
  return cleaned_string == cleaned_string[::-1]

# Example usage:
print("Palindrome Checker (Gemini Style)")
print("---------------------------------")

test_strings = [
    "Racecar",
    "A man, a plan, a canal: Panama",
    "Hello World",
    "Madam, in Eden, I'm Adam",
    "12321",
    "No lemon, no melon"
]

for s in test_strings:
    if is_palindrome_gemini(s):
        print(f"'{s}' is a palindrome.")
    else:
        print(f"'{s}' is NOT a palindrome.")
```

OUTPUT:

```
Palindrome Checker (Gemini Style)
---------------------------------
'Racecar' is a palindrome.
'A man, a plan, a canal: Panama' is a palindrome.
'Hello World' is NOT a palindrome.
'Madam, in Eden, I'm Adam' is a palindrome.
'12321' is a palindrome.
'No lemon, no melon' is a palindrome.
```

CODE:

```
"""
palindrome_compare.py

Contains two palindrome checkers:
- gemini_is_palindrome: normalizes Unicode (NFKD), strips
non-alphanumerics, compares reversed string
```

```python
- copilot_is_palindrome: two-pointer scan, compares characters
case-insensitively without Unicode normalization

A main() runs several test cases and reports where the functions
disagree.
"""
from __future__ import annotations

import re
import unicodedata
from typing import List, Tuple


def gemini_is_palindrome(s: str) -> bool:
    """Gemini-style palindrome checker.

    - Normalizes Unicode using NFKD (so accented characters compare to
their base letters)
    - Keeps only alphanumeric characters
    - Compares cleaned string to its reverse
    """
    if s is None:
        return False
    # Unicode normalization separates base characters from diacritics
    normalized = unicodedata.normalize("NFKD", s)
    cleaned = "".join(ch for ch in normalized if ch.isalnum()).lower()
    return cleaned == cleaned[::-1]


def copilot_is_palindrome(s: str) -> bool:
    """Copilot-style palindrome checker.

    - Uses two-pointer technique
    - Skips non-alphanumeric characters
    - Case-insensitive comparison
    - Does NOT perform Unicode decomposition/normalization
    """
    if s is None:
        return False
    i, j = 0, len(s) - 1
    while i < j:
        while i < j and not s[i].isalnum():
            i += 1
        while i < j and not s[j].isalnum():
            j -= 1
        if s[i].lower() != s[j].lower():
            return False
        i += 1
        j -= 1
    return True
```

```
TEST_CASES: List[Tuple[str, str]] = [
    ("", "empty string"),
    ("a", "single char"),
    ("Abba", "case-insensitive palindrome"),
    ("A man, a plan, a canal: Panama", "classic phrase"),
    ("No 'x' in Nixon", "punctuation and spaces"),
    ("Not a palindrome", "non-palindrome"),
    ("été", "accented palindrome (" + "NFKD-normalized -> 'ete'" +
")"),
    ("ÅbbaA", "accented A vs A"),
    ("😊a😊", "emoji around single letter"),
]


def main() -> None:
    print("Comparing Gemini vs Copilot palindrome checkers\n")

    mismatches = []
    for s, note in TEST_CASES:
        g = gemini_is_palindrome(s)
        c = copilot_is_palindrome(s)
        ok = g == c
        status = "MATCH" if ok else "DIFFER"
        print(f"{status:7} | Gemini: {g!s:5} | Copilot: {c!s:5} |
{note:30} | {s!r}")
        if not ok:
            mismatches.append((s, note, g, c))

    print("\nSummary:")
    if not mismatches:
        print("✅ All outputs match for the test cases.")
    else:
        print(f"⚠️ Found {len(mismatches)} disagreement(s):")
        for s, note, g, c in mismatches:
            print(f" - {note}: Gemini={g} Copilot={c} for input {s!r}")


if __name__ == "__main__":
    main()
```

OUTPUT:

```
MATCH   | Gemini: True  | Copilot: True  | empty string
| ''
MATCH   | Gemini: True  | Copilot: True  | single char
| 'a'
MATCH   | Gemini: True  | Copilot: True  | case-insensitive palindrome
| 'Abba'
MATCH   | Gemini: True  | Copilot: True  | classic phrase
| 'A man, a plan, a canal: Panama'
```

```
MATCH    | Gemini: True  | Copilot: True  | punctuation and spaces
| "No 'x' in Nixon"
MATCH    | Gemini: False | Copilot: False | non-palindrome
| 'Not a palindrome'
MATCH    | Gemini: True  | Copilot: True  | accented palindrome
(NFKD-normalized -> 'ete') | 'été'
MATCH    | Gemini: False | Copilot: False | accented A vs A
| 'ÅbbaA'
MATCH    | Gemini: True  | Copilot: True  | emoji around single letter
| '😊a😊'

Summary:
✅ All outputs match for the test cases.
```

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

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

CODE:

```python
def is_prime(number):
  """
  Checks if a given number is prime.

  Args:
    number: An integer to check for primality.

  Returns:
    True if the number is prime, False otherwise.
  """
  if number <= 1:
    return False
  if number <= 3:
    return True
  if number % 2 == 0 or number % 3 == 0:
    return False
  i = 5
  while i * i <= number:
    if number % i == 0 or number % (i + 2) == 0:
      return False
    i += 6
  return True

# Example usage:
print("Prime Number Checker")
print("---------------------")

test_numbers = [2, 3, 4, 5, 7, 10, 11, 13, 17, 20, 29, 97, 100]

for num in test_numbers:
    if is_prime(num):
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is NOT a prime number.")
```

OUTPUT:

```
Prime Number Checker
---------------------
2 is a prime number.
3 is a prime number.
4 is NOT a prime number.
5 is a prime number.
7 is a prime number.
10 is NOT a prime number.
11 is a prime number.
13 is a prime number.
17 is a prime number.
20 is NOT a prime number.
29 is a prime number.
97 is a prime number.
100 is NOT a prime number.
```