Table of Contents

## 1 FunctionDef count\_vowels(text)

Function - count\_vowels:

Purpose: Counts the total number of vowels (a, e, i, o, u) within a given string in a case-insensitive manner.

Parameters: • text (str): The input string to be scanned for vowel characters.

Detailed Description: The function calculates the number of vowels present in the input text string. It first initializes a set containing all lowercase and uppercase vowels (“a”, “e”, “i”, “o”, “u”, “A”, “E”, “I”, “O”, “U”). Using a set allows for efficient, constant-time lookups.

The function then iterates through each character of the input text string. For every character, it checks if that character exists within the predefined vowels set. A generator expression (1 for ch in text if ch in vowels) yields the number 1 for each character that is confirmed to be a vowel. Finally, the built-in sum() function is used to total all the generated 1s, producing the final count of vowels, which is then returned.

Important Notes: • The vowel check is case-insensitive because the vowels set contains both lowercase and uppercase letters. • Non-alphabetic characters, consonants, numbers, and whitespace within the input string are ignored and do not affect the count. • If an empty string is passed as the text parameter, the function will correctly return 0. • The function has no external dependencies and does not raise any custom exceptions.

**Output Example**:

7

Usage Example:

# Example 1: Counting vowels in a mixed-case sentence  
sentence = "This is an Example String."  
vowel\_count = count\_vowels(sentence)  
print(f"The number of vowels is: {vowel\_count}")  
# Expected Output: The number of vowels is: 7  
  
# Example 2: Counting vowels in a string with no vowels  
no\_vowels\_text = "Rhythm fly by."  
vowel\_count = count\_vowels(no\_vowels\_text)  
print(f"The number of vowels is: {vowel\_count}")  
# Expected Output: The number of vowels is: 0  
  
# Example 3: Counting vowels in an empty string  
empty\_text = ""  
vowel\_count = count\_vowels(empty\_text)  
print(f"The number of vowels is: {vowel\_count}")  
# Expected Output: The number of vowels is: 0

## 2 FunctionDef pairwise\_sum(numbers)

Function - pairwise\_sum:

Purpose: Computes the arithmetic sum of an iterable of numbers using the Kahan summation algorithm for improved numerical precision.

Parameters: • numbers (Iterable[float]): An iterable collection of numbers, such as a list or tuple. The elements can be floats or integers, as each value is internally converted to a float before being added to the sum.

Detailed Description: This function calculates the sum of a sequence of numbers while minimizing the floating-point errors that can accumulate during summation. It implements the Kahan summation algorithm, a technique that provides greater accuracy than a simple iterative sum, especially when dealing with numbers of widely varying magnitudes.

The function initializes two variables: total to 0.0 to store the running sum, and compensation to 0.0 to track the accumulated rounding error. It then iterates through each value in the input numbers iterable.

For each value, the following steps are performed: 1. A corrected value y is calculated by subtracting the compensation (the error from the previous step) from the current value. 2. This corrected value y is added to the running total, and the result is stored in a temporary variable t. 3. The new error introduced by the addition is calculated as (t - total) - y. This captures the low-order bits that were lost due to finite precision and stores them in the compensation variable for the next iteration. 4. The total is updated with the value of t.

After iterating through all the numbers, the function returns the final total, which is a more precise representation of the true sum.

Important Notes: • This function is particularly useful when summing a large set of floating-point numbers or when the numbers have significantly different magnitudes, as it mitigates the loss of precision inherent in standard floating-point arithmetic. • The function expects an iterable of numeric types. If an element in the iterable cannot be converted to a float (e.g., a string), a ValueError or TypeError will be raised at runtime. • No external libraries or modules are required.

**Output Example**:

2.0

Usage Example:

# A list of floats where standard summation might lose precision.  
# The correct sum is 2.0.  
data = [1.0, 1e100, 1.0, -1e100]  
  
# Calculate the sum using the numerically stable pairwise\_sum function.  
stable\_result = pairwise\_sum(data)  
print(f"The stable sum is: {stable\_result}")  
  
# Example with a simple list of numbers.  
numbers\_list = [15.5, 20.0, -5.25, 100.0]  
simple\_sum = pairwise\_sum(numbers\_list)  
print(f"The sum is: {simple\_sum}")

**Expected Output:**

The stable sum is: 2.0  
The sum is: 130.25

## 3 FunctionDef split\_into\_chunks(text, size)

Function - split\_into\_chunks:

Purpose: Splits a given string into a tuple of smaller, fixed-size substrings.

Parameters: • text (str): The input string that will be divided into chunks. • size (int): The desired maximum length for each chunk. This value is required to be a positive integer.

Detailed Description: This function divides a string into multiple segments of a specified length. The function first performs a validation check on the size parameter. If size is zero or a negative number, it raises a ValueError to indicate that the chunk length must be positive.

If the size is valid, the function proceeds with the splitting logic. It uses a generator expression that iterates over the indices of the input text with a step equal to size. The range function is configured to start at index 0, go up to the length of the string, and increment by the size value at each step. For each generated index i, the function uses string slicing (text[i : i + size]) to extract a substring. This process continues until the entire string has been processed. The resulting substrings are collected into a tuple, which is then returned.

Important Notes: • The function will raise a ValueError if the provided size is not a positive integer (i.e., if size <= 0). • The final substring in the returned tuple may be shorter than the specified size if the total length of the input text is not an even multiple of size. • This function does not require any external libraries.

**Output Example**:

('Hel', 'loW', 'orl', 'd')

Usage Example:

# Example 1: Splitting a string where the length is not a multiple of the size  
text\_to\_split = "This is a sample string for chunking."  
chunk\_size = 10  
chunks = split\_into\_chunks(text\_to\_split, chunk\_size)  
print(chunks)  
# Expected Output: ('This is a ', 'sample str', 'ing for ch', 'unking.')  
  
# Example 2: Splitting a string where the length is a multiple of the size  
text\_to\_split\_even = "abcdefghij"  
chunk\_size\_even = 5  
chunks\_even = split\_into\_chunks(text\_to\_split\_even, chunk\_size\_even)  
print(chunks\_even)  
# Expected Output: ('abcde', 'fghij')  
  
# Example 3: Attempting to use an invalid size  
try:  
 invalid\_chunks = split\_into\_chunks("some text", 0)  
except ValueError as e:  
 print(f"Error: {e}")  
# Expected Output: Error: size must be positive