Table of Contents

## 1 FunctionDef count\_vowels(text)

**count\_vowels**: The function of count\_vowels is to count the number of vowels in a given string, ignoring case.

**parameters**: The parameters of this Function. · text: Input string to scan.

**Code Description**: The function count\_vowels takes a string text as input. It initializes a set called vowels containing both lowercase and uppercase vowels. It then iterates through each character ch in the input text. For each character, it checks if the character is present in the vowels set. If it is, the function increments a counter (implicitly done using sum with a generator expression). Finally, the function returns the total count of vowels found in the input string.

**Note**: The function uses a set for efficient vowel checking. The case-insensitive counting is achieved by including both lowercase and uppercase vowels in the vowels set.

**Output Example**: Calling count\_vowels(“Hello, World!”) returns 3.

## 2 FunctionDef pairwise\_sum(numbers)

**pairwise\_sum**: The function of pairwise\_sum is to compute the sum of numbers in an iterable using Kahan summation for improved numerical stability.

**parameters**: The parameters of this Function. · numbers: Any iterable of floats or ints.

**Code Description**: The function pairwise\_sum takes an iterable of numbers (floats or ints) as input. It initializes total and compensation to 0.0. It then iterates through each value in the input numbers. Inside the loop, it converts the current value to a float and subtracts the compensation from it, storing the result in y. It then adds y to the current total and stores the result in t. The compensation is updated to correct for floating-point errors. Finally, the updated total is assigned back to total. After iterating through all the numbers, the function returns the final total, which represents the sum of the input numbers, computed with reduced floating-point error.

**Note**: This function uses Kahan summation to minimize floating point errors, which can be significant when summing a large number of values or values with widely varying magnitudes.

**Output Example**: Calling pairwise\_sum([1.0, 1e100, 1.0, -1e100]) returns 2.0 (approximately, due to floating-point representation).

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

**split\_into\_chunks**: The function of split\_into\_chunks is to split a given string into a tuple of fixed-size substrings (chunks).

**parameters**: The parameters of this Function. · text: The string that will be split into chunks. · size: The desired length of each chunk.

**Code Description**: The function split\_into\_chunks takes a string text and an integer size as input. It first checks if the size is a positive number. If size is not positive, it raises a ValueError. If the size is valid, it uses a generator expression with string slicing to create substrings of length size. The generator expression iterates through the string text with a step of size, creating substrings from index i to i + size. Finally, it converts the generated sequence of substrings into a tuple and returns it. The last chunk may be shorter than size if the length of the input string is not a multiple of size.

**Note**: The function raises a ValueError if the provided size is not positive. The last chunk in the returned tuple may be shorter than the specified size if the input string’s length is not divisible by the size.

**Output Example**: Calling split\_into\_chunks(“abcdefgh”, 3) returns (‘abc’, ‘def’, ‘gh’).