Table of Contents

## 1 FunctionDef count\_vowels(text)

The function of **count\_vowels** is to calculate the total number of vowels within a given string, treating uppercase and lowercase vowels the same.

**parameters**: The parameters of this Function. · text: The input string that will be scanned for vowels.

**Code Description**: The function begins by defining a set named vowels which contains all lowercase and uppercase English vowels (a, e, i, o, u, A, E, I, O, U). It then iterates through each character (ch) in the input text. For every character, it checks if the character is present in the vowels set. A generator expression yields the number 1 for each character that is found in the set. Finally, the built-in sum() function calculates the total of all the 1s generated, effectively counting the vowels, and returns this sum.

**Note**: The vowel check is case-insensitive because the vowels set explicitly includes both uppercase and lowercase letters. Using a set for vowel lookup is an efficient method for membership testing.

**Output Example**:

# Calling count\_vowels("Hello World")  
10

## 2 FunctionDef pairwise\_sum(numbers)

The function of **pairwise\_sum** is to compute the arithmetic sum of an iterable of numbers using the Kahan summation algorithm for improved numerical precision.

**parameters**: The parameters of this Function. · numbers: An iterable (e.g., a list or tuple) containing floating-point numbers or integers that will be summed.

**Code Description**: The function initializes two floating-point variables: total to 0.0 to store the running sum, and compensation to 0.0 to accumulate rounding errors. It then iterates through each value in the input numbers iterable.

Inside the loop, for each value: 1. A corrected value y is calculated by subtracting the current compensation from the input value (which is first cast to a float). 2. A temporary sum t is computed by adding the current total to the corrected value y. 3. The compensation for the next iteration is recalculated. This new compensation value captures the numerical error (the low-order part) that was lost in the t = total + y operation. 4. The main total is updated with the value of the temporary sum t.

After the loop has processed all the numbers, the function returns the final total.

**Note**: This function implements the Kahan summation algorithm, which is designed to minimize floating-point errors that can accumulate when summing a sequence of numbers. It is more numerically stable than a simple iterative addition, especially for datasets with a large number of values or values of widely varying magnitudes.

**Output Example**:

# Calling the function with a list of floats  
pairwise\_sum([0.1, 0.2, 0.3, 0.4, 0.5])

1.5

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

The function of **split\_into\_chunks** is to divide a string into a tuple of smaller, fixed-size substrings.

**parameters**: The parameters of this Function. · text: The string to be split into chunks. · size: The desired length for each chunk. This value must be a positive integer.

**Code Description**: The function first checks if the provided size is a positive number. If size is less than or equal to zero, it raises a ValueError with the message “size must be positive”. If the size is valid, the function uses a generator expression to iterate through the text string. The iteration starts at index 0 and advances by size steps until the end of the string is reached. In each step, it slices the text from the current index i to i + size, creating a chunk. Finally, all generated chunks are collected into a tuple and returned.

**Note**: The last chunk in the returned tuple may be shorter than the specified size if the total length of the input text is not evenly divisible by size.

**Output Example**:

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