Table of Contents

## 1 FunctionDef count\_vowels(text)

## 2 Function count\_vowels

**count\_vowels**: The function of count\_vowels is to count the total number of vowels within a given string in a case-insensitive manner.

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

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

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

**Output Example**:

# Calling count\_vowels("Hello World") returns:  
3

## 3 FunctionDef pairwise\_sum(numbers)

## 4 Function pairwise\_sum

**pairwise\_sum**: 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 collection of floating-point or integer values to be summed.

**Code Description**: The function initializes two floating-point variables, total and compensation, to 0.0. 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. This step adjusts the current number by the error from the previous addition. 2. A temporary sum t is computed by adding the corrected value y to the running total. 3. The compensation variable is updated to capture the numerical error (the low-order bits) lost in the previous step. This is calculated as (t - total) - y. 4. The main total is updated to 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 the accumulation of floating-point errors. It is more numerically stable than a simple sum() when dealing with a large set of numbers or numbers with widely varying magnitudes.

**Output Example**:

# Calling the function with a list of floats  
result = pairwise\_sum([100000000.0, 3.14159, -100000000.0])  
  
# The expected output would be approximately 3.14159  
# result: 3.14159

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

## 6 Function split\_into\_chunks

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

**parameters**: The parameters of this Function. · text (str): The input string that will be divided into chunks. · size (int): The desired maximum length for each chunk. This value must be a positive integer.

**Code Description**: The function first validates the size parameter. It checks if size is less than or equal to zero. If it is, the function raises a ValueError with the message “size must be positive” to prevent invalid operations.

If size is a positive number, the function proceeds to split the text. It uses a generator expression within a tuple() constructor. The range() function generates a sequence of starting indices for each chunk, starting from 0, up to the length of the text, incrementing by size in each step. For each index i generated by the range, a slice of the text is taken from i to i + size. This creates a substring of length size. The final substring may be shorter if the total length of the text is not evenly divisible by size. All the generated substrings are collected into a tuple, which is then returned.

**Note**: The size parameter must be a positive integer; otherwise, a ValueError will be raised. The last element in the returned tuple may have a length less than the specified size.

**Output Example**:

# Calling split\_into\_chunks("HelloWorld", 3) returns:  
('Hel', 'loW', 'orl', 'd')