Table of Contents

## 1 FunctionDef count\_vowels(text)

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

**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 uses a generator expression to iterate through each character (ch) in the input text. For every character, it checks if that character is present in the vowels set. If a character is found in the set, the generator yields the number 1. Finally, the built-in sum() function is used to add up all the 1s produced by the generator, effectively counting the total number of vowels. This final sum is then returned.

**Note**: The function performs a case-insensitive count because the vowels set explicitly includes both uppercase and lowercase vowel characters. Using a set for vowel lookup provides efficient character checking.

**Output Example**: Calling count\_vowels("Hello World") returns 3. ## FunctionDef pairwise\_sum(numbers) **pairwise\_sum**: The function of pairwise\_sum is to compute the sum of an iterable of numbers using the Kahan summation algorithm for improved numerical precision.

**parameters**: The parameters of this Function. · numbers: An iterable containing float or int 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, it first calculates a corrected value y by subtracting the compensation from the current value. This compensation term carries the error from the previous iteration’s summation. Next, it calculates a temporary sum t by adding the corrected value y to the running total. The compensation for the next iteration is then calculated by finding the difference between the new sum t and the original total, and then subtracting y. This step effectively isolates the numerical error introduced by the addition. Finally, the total is updated to the value of t. After the loop has processed all values, the function returns the final total.

**Note**: This function implements the Kahan summation algorithm, which is designed to reduce the accumulation of floating-point errors that can occur when summing a sequence of numbers. It is more numerically stable than a simple iterative sum, especially for datasets with a large range of values or many elements.

**Output Example**: Calling pairwise\_sum([1e10, 1.0, -1e10]) would return 1.0. ## 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 smaller, fixed-size substrings.

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

**Code Description**: The function first checks if the provided size is a positive number. If size is less than or equal to 0, it raises a ValueError with the message “size must be positive”. If the size is valid, the function proceeds to slice the input text. It uses a generator expression that iterates through the text with a step equal to size, creating substrings from the current index i up to i + size. These substrings are then collected into a tuple and returned. The final substring in the tuple may be shorter than the specified size if the length of the input text is not perfectly divisible by size.

**Note**: A ValueError will be raised if the size argument is not a positive integer. The last chunk in the output tuple can be shorter than the specified size.

**Output Example**: ('abc', 'def', 'g')