Table of Contents

## 1 FunctionDef count\_vowels(text)

## 2 Function: count\_vowels

The function of **count\_vowels** is to count and return the total number of vowels within a given input 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 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 each character, it checks if the character is present in the vowels set. If a character is a vowel, the generator yields the number 1. Finally, the sum() function is used to add up all the 1s produced by the generator, effectively counting the total number of vowels. This final count is returned as an integer.

**Note**: The vowel check is case-insensitive because the vowels set explicitly includes both lowercase and uppercase letters. Any characters in the input text that are not vowels (including consonants, numbers, and symbols) are ignored.

**Output Example**:

# Calling the function with the string "Hello World"  
count\_vowels("Hello World")  
  
# The function returns the integer 3  
3

## 3 FunctionDef pairwise\_sum(numbers)

## 4 Function: pairwise\_sum

The function of **pairwise\_sum** is to compute the arithmetic sum of an iterable of numbers using a numerically stable approach to improve precision.

**parameters**: The parameters of this Function. · numbers: An iterable containing float or integer values that will be summed.

**Code Description**: This function implements the Kahan summation algorithm to minimize floating-point errors. It initializes two floating-point variables, total and compensation, to 0.0. The function 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 holds the error from the previous iteration’s addition. Next, it calculates a temporary sum t by adding the corrected value y to the running total. The new compensation value is then calculated by finding the difference (t - total) - y, which effectively captures the low-order bits (the error) lost in the t = total + y operation. Finally, the total is updated to the value of t.

After the loop has processed all values in the iterable, the function returns the final total.

**Note**: This function is particularly useful for summing a large number of floating-point values where standard summation might accumulate significant precision errors. It provides a more accurate result than Python’s built-in sum() for such cases.

**Output Example**:

# Calling pairwise\_sum([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])  
1.0

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

## 6 Function: split\_into\_chunks

The function of **split\_into\_chunks** is to split a given string into a series 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 parameter is less than or equal to zero. If it is, a ValueError is raised with the message “size must be positive”. If size is a positive number, the function proceeds to slice the input text. It uses a generator expression that iterates from the start of the string to its end, with a step equal to size. In each step, it creates a substring of length size starting from the current index. These substrings are then collected into a tuple, which is returned. The last substring in the tuple may be shorter than size if the length of the original text is not a multiple of size.

**Note**: The function will raise a ValueError if the size argument is zero or a negative number. The last chunk in the returned tuple can be shorter than the specified size.

**Output Example**:

('sub', 'str', 'ing')