Table of Contents

## 0.1 FunctionDef count\_vowels(text)

# 1 Function: count\_vowels(text: str)

## 1.1 Overview

The count\_vowels function calculates the total number of vowels within a given input string in a case-insensitive manner.

## 1.2 parameters

* text (str): The input string to be scanned for vowels.

## 1.3 Description

This function provides a straightforward way to count vowel occurrences in a string. The core logic is executed in two main steps:

1. A set named vowels is initialized with all lowercase and uppercase English vowels ("aeiouAEIOU"). Using a set allows for very fast membership checking, which is more efficient than searching through a list or string repeatedly.
2. The function then uses a generator expression, (1 for ch in text if ch in vowels), to iterate through each character (ch) of the input text. For every character that is found within the vowels set, the number 1 is generated.

Finally, the built-in sum() function is called on this generator. It consumes all the generated 1s and adds them together, producing the total count of vowels, which is then returned as an integer.

# Internal logic breakdown  
vowels = set("aeiouAEIOU")  
# For an input "Test", the generator would yield 1 for 'e'.  
# sum() would then calculate the total, returning 1.  
return sum(1 for ch in text if ch in vowels)

## 1.4 Usage Notes

* The function is case-insensitive because the vowels set contains both aeiou and AEIOU.
* Only the standard English vowels (a, e, i, o, u) are counted. Vowels with accents or from other alphabets (e.g., ‘á’, ‘ü’, ‘ø’) will not be counted.
* Any non-vowel characters, including consonants, numbers, whitespace, and symbols, are ignored.

**Output Example**: The function returns an integer representing the total count of vowels.

7

## 1.5 Example

# Example usage  
input\_string = "This is a Simple Test String 123!"  
vowel\_count = count\_vowels(input\_string)  
print(f"The text is: '{input\_string}'")  
print(f"The number of vowels is: {vowel\_count}")

**Output:**

The text is: 'This is a Simple Test String 123!'  
The number of vowels is: 7

## 1.6 FunctionDef pairwise\_sum(numbers)

# 2 Function: pairwise\_sum(numbers: Iterable[float])

## 2.1 Overview

The pairwise\_sum function computes the sum of an iterable of numbers using the Kahan summation algorithm, which provides a more numerically stable and precise result compared to a naive summation, especially for floating-point numbers.

## 2.2 parameters

* numbers (Iterable[float]): An iterable collection of numbers (floats or integers) that need to be summed. This can be a list, tuple, or any other iterable object.

## 2.3 Description

This function provides a robust method for summing floating-point numbers by minimizing rounding errors that can accumulate during computation. Standard summation can lose precision, particularly when adding a very small number to a very large one. This function implements the Kahan summation algorithm to counteract this effect.

The algorithm works by maintaining a running compensation variable that accumulates the error from each addition.

The logic proceeds as follows: 1. Initialize total and compensation to 0.0. 2. Iterate through each value in the input numbers. 3. For each value, it is first cast to a float. 4. The compensation from the previous step is subtracted from the current value to create a corrected value y. This reintroduces the “lost” part from the prior addition. y = float(value) - compensation 5. The corrected value y is added to the running total. Due to floating-point limitations, this addition may still be imprecise. t = total + y 6. The error from the addition in the previous step is calculated and stored in compensation. This is done by observing the difference between the result t and the original total, and then subtracting y. If the addition were perfect, this would be zero. compensation = (t - total) - y 7. The total is updated with the new sum t. 8. After iterating through all numbers, the final, more accurate total is returned.

# Inside the loop  
y = float(value) - compensation  
t = total + y  
compensation = (t - total) - y  
total = t

This process ensures that the low-order bits, which are often lost in standard floating-point addition, are carried over to the next iteration, leading to a final sum with higher precision.

## 2.4 Usage Notes

* This function is highly recommended when accuracy is critical and the dataset involves summing a large quantity of floating-point numbers or numbers with a wide range of magnitudes.
* While Python’s built-in sum() function is often sufficient, pairwise\_sum provides an explicit and reliable implementation of a high-precision summation algorithm.
* Any integers provided in the input numbers iterable will be automatically converted to floats during the calculation.

**Output Example**: The function returns a single floating-point number.

51.000000123

## 2.5 Example

# Example usage  
# A list containing numbers of vastly different magnitudes  
data = [150000.75, 0.000000123, 50.25, -150000.0]  
result = pairwise\_sum(data)  
print(result)  
  
# Another example with a generator  
data\_generator = (i \* 0.1 for i in range(10))  
result\_from\_generator = pairwise\_sum(data\_generator)  
print(result\_from\_generator)

**Output:**

51.000000123  
4.5

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

# 3 Function: split\_into\_chunks(text: str, size: int)

## 3.1 Overview

The split\_into\_chunks function divides a given string into a series of smaller, fixed-size substrings.

## 3.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| text | str | The input string that will be divided into chunks. |
| size | int | The desired length for each chunk. This value must be a positive integer. |

## 3.3 Description

This function provides a straightforward way to segment a string into multiple parts of a specified length.

The function first validates the size parameter. If size is less than or equal to zero, it is not a valid length for a chunk, so the function raises a ValueError.

If the size is valid, the function proceeds to slice the string. It uses a generator expression within a tuple() constructor for efficiency. The range(0, len(text), size) function generates the starting indices for each chunk. For each index i, a slice text[i : i + size] is created. This slice starts at index i and extends up to, but not including, index i + size.

The resulting substrings are collected into a tuple, which is then returned. If the length of the input text is not a multiple of size, the final chunk in the tuple will be shorter than the specified size.

# Internal logic for a text "abcdefg" and size 3  
# 1. Validate size: 3 > 0, so it's valid.  
# 2. Generate indices using range(0, 7, 3), which yields 0, 3, 6.  
# 3. For index 0: text[0:3] -> "abc"  
# 4. For index 3: text[3:6] -> "def"  
# 5. For index 6: text[6:9] -> "g" (slice automatically stops at the end of the string)  
# 6. Return the collected chunks as a tuple: ("abc", "def", "g")

## 3.4 Usage Notes

* The function will raise a ValueError if the size parameter is zero or a negative number.
* The last element in the returned tuple may be shorter than size if the total length of the text is not evenly divisible by size.
* If an empty string is passed as the text parameter, the function will return an empty tuple ().
* The return type is always a tuple of strings.

**Output Example**: A possible return value for a successful operation.

('This ', 'is a ', 'sampl', 'e.')

## 3.5 Example

# Example usage  
long\_string = "This is a sample string for demonstrating the chunking function."  
chunk\_length = 10  
  
# Split the string into chunks of 10 characters  
chunks = split\_into\_chunks(long\_string, chunk\_length)  
print(chunks)  
  
# Example with a string length not divisible by chunk size  
short\_string = "123456789"  
short\_chunks = split\_into\_chunks(short\_string, 4)  
print(short\_chunks)  
  
# Example of invalid size  
try:  
 split\_into\_chunks("some text", 0)  
except ValueError as e:  
 print(e)

**Output:**

('This is a ', 'sample str', 'ing for de', 'monstratin', 'g the chun', 'king funct', 'ion.')  
('1234', '5678', '9')  
size must be positive