Table of Contents

## 0.1 FunctionDef count\_vowels(text)

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

## 1.1 Overview

The count\_vowels function counts the total number of vowels within a given 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 simple and efficient way to determine the vowel count in any given text.

The core logic begins by defining a set named vowels which contains all English vowels in both lowercase (a, e, i, o, u) and uppercase (A, E, I, O, U). Using a set allows for highly efficient membership testing, which is faster than searching through a list or string repeatedly.

The function then iterates through each character (ch) in the input text. For each character, it checks if the character is present in the vowels set. A generator expression, (1 for ch in text if ch in vowels), yields a 1 for every character that is a vowel.

Finally, the built-in sum() function is used to tally all the 1s generated, producing the total count of vowels in the string.

# Internal logic breakdown  
vowels = set("aeiouAEIOU")  
# For a text "Hello", the generator would yield 1 for 'e' and 1 for 'o'.  
# sum() would then calculate 1 + 1 = 2.  
return sum(1 for ch in text if ch in vowels)

## 1.4 Usage Notes

* The function is case-insensitive. It will count both ‘a’ and ‘A’ as vowels.
* Only the characters ‘a’, ‘e’, ‘i’, ‘o’, ‘u’ (and their uppercase equivalents) are considered vowels. The character ‘y’ is not included.
* Non-alphabetic characters, such as numbers, punctuation, and whitespace, are ignored and do not contribute to the count.

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

7

## 1.5 Example

# Example usage  
input\_sentence = "Hello World! This is a Test."  
vowel\_count = count\_vowels(input\_sentence)  
print(f"The input string is: '{input\_sentence}'")  
print(f"The number of vowels is: {vowel\_count}")

**Output:**

The input string is: 'Hello World! This is a Test.'  
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 arithmetic sum of a sequence of numbers using a numerically stable algorithm to minimize floating-point errors.

## 2.2 parameters

* numbers: Iterable[float] - An iterable collection of numbers (e.g., a list, tuple, or generator) to be summed. The elements can be floats or integers, as they will be cast to floats internally.

## 2.3 Description

This function provides a high-precision method for summing floating-point numbers by implementing the Kahan summation algorithm. Standard summation can suffer from a loss of precision when adding a very small number to a very large number, as the lower-order bits of the small number are lost. The Kahan algorithm mitigates this issue by tracking a running compensation for the accumulated error.

The logic proceeds as follows: 1. Two floating-point variables, total and compensation, are initialized to 0.0. total holds the running sum, and compensation tracks the accumulated round-off error. 2. The function iterates through each value in the input numbers. 3. For each value, it is first corrected by subtracting the compensation from the previous iteration. This corrected value is stored in y. 4. The corrected value y is added to the running total, and the result is stored in a temporary variable t. 5. Due to the finite precision of floating-point arithmetic, the addition total + y might lose some low-order bits. The new compensation is calculated as (t - total) - y. This expression captures the round-off error from the summation. 6. The total is updated to t. 7. This process repeats for all numbers, with each step correcting for the error of the last. The final total is returned, providing a more accurate sum than a naive approach.

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

## 2.4 Usage Notes

* This function is more accurate than Python’s built-in sum() for floating-point numbers, especially when the input contains values with widely different magnitudes.
* The input iterable can contain both integers and floats; integers will be automatically cast to floats during the computation.
* It is particularly useful in scientific and financial calculations where precision is critical.

**Output Example**: A single floating-point number representing the sum.

10000000007.0

## 2.5 Example

# Example usage with numbers of vastly different magnitudes  
# A naive sum might result in 7.0 due to precision loss  
numbers\_list = [1e10, 1, 2, -1e10, 4]  
result = pairwise\_sum(numbers\_list)  
print(result)  
  
# Example with a simple list of floats  
more\_numbers = [0.1, 0.2, 0.3, 0.4, 0.5]  
result\_simple = pairwise\_sum(more\_numbers)  
print(result\_simple)

**Output:**

7.0  
1.5

## 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-length substrings.

## 3.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| text | str | The input string that needs to 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 raises a ValueError, as chunking into non-positive lengths is not a valid operation.

If the size is valid, the function proceeds to iterate through the input text using a generator expression. It generates a sequence of starting indices for the slices by using range(0, len(text), size). For each starting index i, it extracts a substring slice text[i : i + size]. This process continues until the entire string has been covered.

The final chunk may be shorter than the specified size if the total length of the text is not perfectly divisible by size. All the generated string chunks are then collected into a tuple and returned.

# The core logic uses a generator expression with range stepping  
tuple(text[i : i + size] for i in range(0, len(text), size))

## 3.4 Usage Notes

* The size parameter must be a positive integer. Providing 0 or a negative number will result in a ValueError.
* The returned value is a tuple of strings. Tuples are immutable, meaning their contents cannot be changed after creation.
* The last element in the returned tuple may have a length less than size if the input string’s length is not a multiple of size.

**Output Example**: A possible return value for splitting "abcdefg" with a size of 3.

('abc', 'def', 'g')

## 3.5 Example

# Example 1: Splitting a string into chunks of size 5  
text\_to\_split = "This is a sample text to be split."  
chunk\_size = 5  
  
result = split\_into\_chunks(text\_to\_split, chunk\_size)  
print(f"Original Text: '{text\_to\_split}'")  
print(f"Chunks of size {chunk\_size}: {result}")  
  
# Example 2: The last chunk is shorter  
text\_to\_split\_2 = "12345678"  
chunk\_size\_2 = 3  
  
result\_2 = split\_into\_chunks(text\_to\_split\_2, chunk\_size\_2)  
print(f"\nOriginal Text: '{text\_to\_split\_2}'")  
print(f"Chunks of size {chunk\_size\_2}: {result\_2}")  
  
# Example 3: Handling an invalid size  
try:  
 split\_into\_chunks("some text", 0)  
except ValueError as e:  
 print(f"\nError with invalid size: {e}")

**Output:**

Original Text: 'This is a sample text to be split.'  
Chunks of size 5: ('This ', 'is a ', 'sampl', 'e tex', 't to ', 'be sp', 'lit.')  
  
Original Text: '12345678'  
Chunks of size 3: ('123', '456', '78')  
  
Error with invalid size: size must be positive