Table of Contents

## 0.1 FunctionDef fibonacci(n)

# 1 fibonacci

### 1.0.1 Overview

The fibonacci function computes the nth Fibonacci number using an efficient iterative approach.

### 1.0.2 Parameters

| Parameter | Type | Description |
| --- | --- | --- |
| n | int | The 0-indexed position in the Fibonacci sequence for which to compute the value. |

### 1.0.3 Description

This function calculates a number in the Fibonacci sequence based on its index n. The Fibonacci sequence starts with 0 and 1, and each subsequent number is the sum of the two preceding ones (e.g., 0, 1, 1, 2, 3, 5, 8…).

The function first validates the input n. If n is a negative number, it raises a ValueError because the Fibonacci sequence is not defined for negative indices.

It initializes two variables, a and b, to the first two numbers of the sequence, 0 and 1, respectively. It then enters a for loop that iterates n times. In each iteration, the values of a and b are updated. The current value of b is assigned to a, and the sum of the old a and b is assigned to b. This process effectively moves one step forward in the sequence.

# Inside the loop  
a, b = b, a + b

After the loop completes n iterations, the variable a will hold the nth Fibonacci number, which is then returned.

### 1.0.4 Usage Notes

* The function uses a 0-indexed sequence. For example, fibonacci(0) returns 0, fibonacci(1) returns 1, and so on.
* Providing a negative integer for n will result in a ValueError.
* This iterative implementation is memory-efficient compared to a naive recursive approach, as it avoids deep recursion stacks and recalculating the same values.

**Output Example**: The function returns a single integer representing the Fibonacci number at the specified index.

34

### 1.0.5 Example

# Example usage: Compute the 9th Fibonacci number (0-indexed)  
try:  
 result = fibonacci(9)  
 print(f"The 9th Fibonacci number is: {result}")  
  
 # Example with the start of the sequence  
 result\_zero = fibonacci(0)  
 print(f"The 0th Fibonacci number is: {result\_zero}")  
  
 # Example that would raise an error  
 # fibonacci(-1)  
  
except ValueError as e:  
 print(f"Error: {e}")

#### 1.0.5.1 Output

The 9th Fibonacci number is: 34  
The 0th Fibonacci number is: 0

## 1.1 FunctionDef invert\_dictionary(mapping)

### 1.1.1 Overview

The invert\_dictionary function inverts a given dictionary by swapping its keys and values to create a new dictionary.

### 1.1.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| mapping | Dict[str, int] | The dictionary to be inverted. It must have string keys, integer values, and all values must be unique. |

### 1.1.3 Description

The invert\_dictionary function takes a dictionary with string keys and integer values and returns a new dictionary where the keys and values have been swapped.

The function first validates the input dictionary to ensure that all its values are unique. It performs this check by comparing the count of all values against the count of unique values (derived by converting the values to a set). If duplicate values are detected, a ValueError is raised, as dictionary keys must be unique, and the original values become the keys in the new dictionary.

If the validation passes, the function uses a dictionary comprehension, {value: key for key, value in mapping.items()}, to construct the new dictionary. This expression iterates over each key-value pair in the input mapping, creating a new pair where the original value becomes the key and the original key becomes the value.

### 1.1.4 Usage Notes

* This function strictly requires that all values in the input mapping dictionary be unique. An attempt to invert a dictionary with duplicate values will result in a ValueError.
* The function is non-destructive; it does not modify the original dictionary. It returns a completely new dictionary instance.

**Output Example**: A successful call returns a new dictionary with integer keys and string values.

{1: 'one', 10: 'ten', 100: 'one\_hundred'}

### 1.1.5 Example

# Example 1: Successful inversion with unique values  
from typing import Dict  
  
def invert\_dictionary(mapping: Dict[str, int]) -> Dict[int, str]:  
 """Invert a dictionary with unique values."""  
 if len(mapping.values()) != len(set(mapping.values())):  
 raise ValueError("Values must be unique to invert dictionary")  
 return {value: key for key, value in mapping.items()}  
  
# --- Usage ---  
original\_dict = {'alpha': 10, 'beta': 20, 'gamma': 30}  
inverted\_dict = invert\_dictionary(original\_dict)  
print(f"Original: {original\_dict}")  
print(f"Inverted: {inverted\_dict}")  
  
# Example 2: Attempted inversion with duplicate values  
try:  
 non\_unique\_dict = {'a': 1, 'b': 2, 'c': 1}  
 print(f"\nAttempting to invert: {non\_unique\_dict}")  
 invert\_dictionary(non\_unique\_dict)  
except ValueError as e:  
 print(f"Error: {e}")

#### 1.1.5.1 Output

Original: {'alpha': 10, 'beta': 20, 'gamma': 30}  
Inverted: {10: 'alpha', 20: 'beta', 30: 'gamma'}  
  
Attempting to invert: {'a': 1, 'b': 2, 'c': 1}  
Error: Values must be unique to invert dictionary

## 1.2 FunctionDef is\_palindrome(text)

### 1.2.1 Overview

The is\_palindrome function determines if a given string is a palindrome, ignoring letter casing and whitespace.

### 1.2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| text | str | The input string to be evaluated. |

### 1.2.3 Description

The is\_palindrome function works by first normalizing the input text and then comparing the normalized string to its reverse.

The normalization process is handled in a single line: normalized = ''.join(ch.lower() for ch in text if not ch.isspace()). This expression iterates through every character (ch) in the input text. For each character, it performs two actions: 1. It converts the character to lowercase using ch.lower(). 2. It checks if the character is a whitespace character using ch.isspace().

Only characters that are not whitespace are kept. These processed characters are then joined together to form a new string called normalized.

Finally, the function returns the result of the boolean comparison normalized == normalized[::-1]. The slice [::-1] is a standard Python idiom for reversing a sequence. If the normalized string is identical to its reversed version, the function returns True, confirming it is a palindrome. Otherwise, it returns False.

### 1.2.4 Usage Notes

* The function is case-insensitive. For example, is\_palindrome("Racecar") and is\_palindrome("racecar") will both return True.
* All whitespace characters (e.g., spaces, tabs, newlines) are removed from the string before the palindrome check is performed.
* Punctuation and other symbols are **not** removed and are considered part of the string during the check. For example, is\_palindrome("A man, a plan, a canal: Panama") will return False because the comma and colon are included in the comparison.

**Output Example**: The function returns a boolean value.

True

### 1.2.5 Example

# Example 1: A classic palindrome with mixed casing and spaces  
# Normalizes to "nolemonnomelon"  
result1 = is\_palindrome("No lemon no melon")  
print(f"'No lemon no melon' is a palindrome: {result1}")  
  
# Example 2: A simple non-palindrome string  
# Normalizes to "helloworld"  
result2 = is\_palindrome("hello world")  
print(f"'hello world' is a palindrome: {result2}")  
  
# Example 3: A string with punctuation that fails the check  
# Normalizes to "madam,i'madam"  
result3 = is\_palindrome("Madam, I'm Adam")  
print(f"'Madam, I'm Adam' is a palindrome: {result3}")

#### 1.2.5.1 Output

'No lemon no melon' is a palindrome: True  
'hello world' is a palindrome: False  
'Madam, I'm Adam' is a palindrome: False