Table of Contents

## 1 FunctionDef fibonacci(n)

Function - fibonacci:

Purpose: Computes the nth Fibonacci number using an iterative approach.

Parameters: • n (int): The 0-indexed position in the Fibonacci sequence for which to compute the value.

Detailed Description: The function calculates the Fibonacci number at a specified index n. It begins by validating the input n. If n is a negative number, the function raises a ValueError because the Fibonacci sequence is not defined for negative indices.

If the input is valid (non-negative), the function initializes two variables, a and b, to 0 and 1 respectively. These represent the first two numbers in the Fibonacci sequence (F₀ and F₁).

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 previous a and b is assigned to b. This operation, a, b = b, a + b, effectively advances the sequence one step.

After the loop has completed n iterations, the variable a will hold the nth Fibonacci number. The function concludes by returning the final value of a.

Important Notes: • The function will raise a ValueError if the input n is less than 0. • The sequence is 0-indexed. For example, fibonacci(0) returns 0, fibonacci(1) returns 1, and fibonacci(2) returns 1. • This implementation is iterative, which is memory-efficient and avoids the risk of stack overflow errors that can occur with recursive solutions for large values of n. • No external libraries or modules are required to use this function.

**Output Example**:

13

Usage Example:

# Example 1: Calculate the 7th Fibonacci number (0-indexed)  
result = fibonacci(7)  
print(f"The 7th Fibonacci number is: {result}")  
# Output: The 7th Fibonacci number is: 13  
  
# Example 2: Calculate the 0th Fibonacci number  
result\_zero = fibonacci(0)  
print(f"The 0th Fibonacci number is: {result\_zero}")  
# Output: The 0th Fibonacci number is: 0  
  
# Example 3: Attempting to use a negative index  
try:  
 fibonacci(-5)  
except ValueError as e:  
 print(e)  
# Output: n must be non-negative

## 2 FunctionDef invert\_dictionary(mapping)

Function - invert\_dictionary:

Purpose: Creates a new dictionary by swapping the keys and values of an existing dictionary, ensuring all original values are unique.

Parameters: • mapping (Dict[str, int]): The dictionary to be inverted. It is expected to map string keys to integer values, and all integer values must be unique.

Detailed Description: The function first validates the input dictionary mapping to ensure it can be inverted without data loss. It performs this check by comparing the number of values in the dictionary with the number of unique values. This is achieved by comparing the length of mapping.values() with the length of a set created from mapping.values(). Since a set can only contain unique elements, if these two lengths are not equal, it indicates that the original dictionary contains duplicate values.

If duplicate values are detected, the function raises a ValueError with a descriptive message to inform the user that the inversion cannot be performed.

If all values are unique, the function proceeds to create the inverted dictionary using a dictionary comprehension. It iterates through each key-value pair of the input mapping and constructs a new dictionary where each original value becomes a new key, and each original key becomes its corresponding value. The newly created inverted dictionary is then returned.

Important Notes: • The function will raise a ValueError if the input dictionary contains duplicate values, as this would lead to key collisions in the resulting inverted dictionary. • The values from the input dictionary must be of a hashable type (e.g., int, str, tuple) to be used as keys in the returned dictionary. The type hint Dict[str, int] ensures this condition is met for the specified types. • The function returns a new dictionary and does not modify the original mapping dictionary.

**Output Example**:

{1: 'apple', 2: 'banana', 3: 'cherry'}

Usage Example:

# Example 1: Successful inversion with unique values  
original\_dict = {'a': 1, 'b': 2, 'c': 3}  
inverted\_dict = invert\_dictionary(original\_dict)  
print(f"Original: {original\_dict}")  
print(f"Inverted: {inverted\_dict}")  
# Expected Output:  
# Original: {'a': 1, 'b': 2, 'c': 3}  
# Inverted: {1: 'a', 2: 'b', 3: 'c'}  
  
# Example 2: Attempted inversion with duplicate values  
invalid\_dict = {'a': 1, 'b': 2, 'c': 1}  
try:  
 invert\_dictionary(invalid\_dict)  
except ValueError as e:  
 print(f"Error: {e}")  
# Expected Output:  
# Error: Values must be unique to invert dictionary

## 3 FunctionDef is\_palindrome(text)

Function - is\_palindrome:

Purpose: Determines if a given string is a palindrome by comparing it to its reverse after ignoring case and spaces.

Parameters: • text (str): The string that will be checked for palindrome properties.

Detailed Description: The function first normalizes the input text to prepare it for comparison. It creates a new string called normalized by iterating through each character of the input text. During this process, it converts every character to its lowercase equivalent and discards any character that is a whitespace (like spaces, tabs, or newlines). The resulting characters are joined together to form the normalized string.

Finally, the function checks if the normalized string is identical to its reverse. The reversal is achieved using Python’s slice notation [::-1]. The function returns True if the normalized string and its reversed version are the same, indicating a palindrome, and False otherwise.

Important Notes: • The check is case-insensitive because all characters are converted to lowercase before comparison. • All whitespace characters are ignored and removed from the string. • Punctuation, numbers, and other symbols are *not* ignored and will be included in the palindrome check. For example, “A man, a plan…” will likely not evaluate as a palindrome because the comma is not removed. • The function does not raise any specific errors and handles empty strings gracefully (an empty string is considered a palindrome).

**Output Example**:

True

Usage Example:

# Example 1: A simple palindrome with mixed casing  
result1 = is\_palindrome("Racecar")  
print(f"'Racecar' is a palindrome: {result1}")  
# Output: 'Racecar' is a palindrome: True  
  
# Example 2: A palindrome with spaces and mixed casing  
result2 = is\_palindrome("A man a plan a canal Panama")  
print(f"'A man a plan a canal Panama' is a palindrome: {result2}")  
# Output: 'A man a plan a canal Panama' is a palindrome: True  
  
# Example 3: A non-palindrome string  
result3 = is\_palindrome("hello world")  
print(f"'hello world' is a palindrome: {result3}")  
# Output: 'hello world' is a palindrome: False  
  
# Example 4: A string with punctuation (will return False)  
result4 = is\_palindrome("Madam, I'm Adam")  
print(f"'Madam, I'm Adam' is a palindrome: {result4}")  
# Output: 'Madam, I'm Adam' is a palindrome: False