Table of Contents

## 0.1 FunctionDef fibonacci(n)

# 1 Function: fibonacci(n: int)

## 1.1 Overview

The fibonacci function computes the nth number in the Fibonacci sequence using an iterative method.

## 1.2 parameters

* n (int): The 0-indexed index of the Fibonacci number to compute.

## 1.3 Description

This function calculates a Fibonacci number based on its index n. The logic proceeds as follows:

1. **Input Validation**: The function first checks if the input n is less than 0. Since the Fibonacci sequence is not defined for negative indices, it raises a ValueError if this condition is met.
2. **Initialization**: Two variables, a and b, are initialized to 0 and 1 respectively. These represent the first two numbers in the Fibonacci sequence, F(0) and F(1).
3. **Iteration**: The function then enters a for loop that iterates n times. Inside the loop, the core Fibonacci calculation happens:
   * The value of b is assigned to a.
   * The sum of the old values of a and b is assigned to b. This simultaneous update, a, b = b, a + b, efficiently advances the sequence one step at a time.
4. **Return Value**: After the loop completes, the variable a holds the nth Fibonacci number. For n=0, the loop does not run, and the initial value of a (0) is returned. For any n > 0, the loop updates a to the correct value in the sequence.

# Initialization for the sequence  
a, b = 0, 1  
  
# For n=3, the loop runs 3 times:  
# 1. a becomes 1, b becomes 0 + 1 = 1  
# 2. a becomes 1, b becomes 1 + 1 = 2  
# 3. a becomes 2, b becomes 1 + 2 = 3  
  
# The function returns a, which is 2 (the 3rd Fibonacci number)

## 1.4 Usage Notes

* The input n must be a non-negative integer. Providing a negative number will result in a ValueError.
* The function is 0-indexed, meaning fibonacci(0) returns the first number (0), fibonacci(1) returns the second (1), and so on.
* This iterative approach is memory-efficient compared to a naive recursive implementation, as it avoids a deep call stack.

**Output Example**: A single integer representing the calculated Fibonacci number. 13

## 1.5 Example

# Example usage: Compute the 7th Fibonacci number (0-indexed)  
n\_index = 7  
result = fibonacci(n\_index)  
print(f"The Fibonacci number at index {n\_index} is: {result}")

**Output:**

The Fibonacci number at index 7 is: 13

## 1.6 FunctionDef invert\_dictionary(mapping)

# 2 Function: invert\_dictionary(mapping: Dict[str, int]) -> Dict[int, str]

## 2.1 Overview

The invert\_dictionary function swaps the keys and values of a given dictionary, returning a new inverted dictionary.

## 2.2 parameters

* mapping: Dict[str, int]
  + A dictionary mapping string keys to integer values. It is essential that all values in this dictionary are unique, as they will become the keys in the new dictionary.

## 2.3 Description

This function provides a safe way to invert a dictionary by first validating the uniqueness of its values.

The function begins by checking if all values in the input mapping dictionary are unique. It accomplishes this by comparing the total number of values (len(mapping.values())) with the number of unique values, which is determined by converting the values to a set (len(set(mapping.values()))).

If duplicate values are detected (i.e., the lengths do not match), the function cannot perform a valid inversion, as this would lead to key collisions in the resulting dictionary. In this scenario, it raises a ValueError with the message “Values must be unique to invert dictionary”.

If all values are unique, the function proceeds to create and return a new dictionary. It uses a dictionary comprehension, {value: key for key, value in mapping.items()}, to iterate through each key-value pair of the original mapping. For each pair, it constructs a new entry where the original value becomes the new key and the original key becomes the new value.

## 2.4 Usage Notes

* The function will raise a ValueError if the input dictionary contains duplicate values.
* This function returns a new dictionary and does not modify the original mapping dictionary in place.
* The type hints indicate an inversion from Dict[str, int] to Dict[int, str], but the logic applies to any dictionary where keys and values have types suitable for dictionary keys.

**Output Example**: A successfully inverted dictionary.

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

## 2.5 Example

**Successful Inversion**

# Example usage with unique values  
original\_dict = {'apple': 1, 'banana': 2, 'cherry': 3}  
inverted\_dict = invert\_dictionary(original\_dict)  
print(inverted\_dict)

**Output:**

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

**Error Case (Duplicate Values)**

# Example usage with duplicate values  
import pytest  
  
non\_unique\_dict = {'apple': 1, 'banana': 2, 'orange': 1}  
  
try:  
 invert\_dictionary(non\_unique\_dict)  
except ValueError as e:  
 print(e)

**Output:**

Values must be unique to invert dictionary

## 2.6 FunctionDef is\_palindrome(text)

# 3 Function: is\_palindrome(text: str)

## 3.1 Overview

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

## 3.2 parameters

* text (str): The input string to be checked for palindrome properties.

## 3.3 Description

This function evaluates whether a string reads the same forwards and backwards. To achieve this, it first performs a normalization step on the input text.

The normalization process involves creating a new string, normalized, by iterating through each character of the input text. During this iteration, two transformations occur: 1. Any character that is a whitespace (e.g., space, tab, newline) is ignored and excluded from the new string. This is checked using ch.isspace(). 2. All remaining characters are converted to their lowercase equivalents using ch.lower().

For example, an input of "Taco Cat" would be normalized to "tacocat".

normalized = ''.join(ch.lower() for ch in text if not ch.isspace())

After normalization, the function compares the normalized string to its reverse. The reversal is achieved using Python’s slice notation [::-1]. If the normalized string is identical to its reversed version, the function returns True; otherwise, it returns False.

return normalized == normalized[::-1]

## 3.4 Usage Notes

* **Case-Insensitive**: The comparison ignores the original casing of the letters. For example, “Level” and “level” are both considered palindromes.
* **Whitespace Ignored**: All whitespace characters are removed before the check. This allows phrases like “taco cat” to be correctly identified as a palindrome.
* **Punctuation and Symbols**: The function does not explicitly remove punctuation or symbols. They are included in the normalized string and will affect the palindrome check. For instance, is\_palindrome("madam.") would return False because the period is not mirrored at the beginning of the string.

**Output Example**: The function returns a boolean value. - True if the text is a palindrome. - False if it is not.

## 3.5 Example

# Example 1: A classic palindrome with mixed casing and spaces  
test\_string\_1 = "A man a plan a canal Panama"  
result\_1 = is\_palindrome(test\_string\_1)  
print(f"Is '{test\_string\_1}' a palindrome? {result\_1}")  
  
# Example 2: A simple non-palindrome string  
test\_string\_2 = "hello world"  
result\_2 = is\_palindrome(test\_string\_2)  
print(f"Is '{test\_string\_2}' a palindrome? {result\_2}")  
  
# Example 3: A palindrome with punctuation (evaluates to False)  
test\_string\_3 = "Eva, can I see bees in a cave?"  
result\_3 = is\_palindrome(test\_string\_3)  
print(f"Is '{test\_string\_3}' a palindrome? {result\_3}")

**Output:**

Is 'A man a plan a canal Panama' a palindrome? True  
Is 'hello world' a palindrome? False  
Is 'Eva, can I see bees in a cave?' a palindrome? False