Table of Contents

## 1 FunctionDef fibonacci(n)

## 2 Function fibonacci

**fibonacci**: The function of fibonacci is to compute the nth Fibonacci number using an iterative approach.

**parameters**: The parameters of this Function. · n: An integer representing the 0-indexed position of the desired Fibonacci number.

**Code Description**: The function begins by checking if the input n is a negative number. If it is, a ValueError is raised, as the Fibonacci sequence is not defined for negative indices.

It then initializes two variables, a and b, to 0 and 1 respectively. These represent the first two numbers in the Fibonacci sequence (F₀ and F₁).

The function enters a for loop that iterates n times. In each iteration, the values of a and b are updated simultaneously. The current value of b is assigned to a, and the sum of the previous a and b is assigned to b. This process effectively calculates the next number in the sequence.

After the loop has completed n iterations, the value of a will hold the nth Fibonacci number. The function then returns this final value of a.

**Note**: The function requires a non-negative integer as input. Providing a negative number will result in a ValueError. The calculation is performed iteratively, which is efficient in terms of memory usage compared to a naive recursive approach.

**Output Example**:

# Calling fibonacci(8) returns:  
21

## 3 FunctionDef invert\_dictionary(mapping)

## 4 Function invert\_dictionary

**invert\_dictionary**: The function of invert\_dictionary is to swap the keys and values of a given dictionary, raising an error if the original dictionary’s values are not unique.

**parameters**: The parameters of this Function. · mapping: A dictionary mapping string keys to integer values. The values in this dictionary must be unique for the inversion to succeed.

**Code Description**: The function first validates the input dictionary mapping to ensure all its values are unique. It does this by comparing the number of values in the dictionary (len(mapping.values())) with the number of unique values, which is determined by converting the values to a set and checking its length (len(set(mapping.values()))). If these two lengths are not equal, it signifies that there are duplicate values, and the function raises a ValueError with the message “Values must be unique to invert dictionary”. If all values are unique, the function proceeds to create a new 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 key and its corresponding original key becomes the value. This newly inverted dictionary is then returned.

**Note**: This function will fail and raise a ValueError if the input dictionary contains any duplicate values. The type hints specify a Dict[str, int] as input and Dict[int, str] as output, but the core logic will work with any dictionary where both keys and values are hashable types.

**Output Example**:

# Given the input:  
mapping = {'apple': 1, 'banana': 2, 'cherry': 3}  
  
# Calling invert\_dictionary(mapping) would return:  
{1: 'apple', 2: 'banana', 3: 'cherry'}

## 5 FunctionDef is\_palindrome(text)

## 6 Function is\_palindrome

**is\_palindrome**: The function of is\_palindrome is to determine if a given string is a palindrome, ignoring character casing and spaces.

**parameters**: The parameters of this Function. · text: The string to check for palindrome properties.

**Code Description**: The function first normalizes the input text. It does this by iterating through each character of the string, converting each character to lowercase, and discarding any whitespace characters. These processed characters are then joined together to form a new string called normalized.

Finally, the function compares the normalized string with its reversed version, which is created using the slice notation [::-1]. It returns True if the normalized string is identical to its reverse, indicating it is a palindrome, and False otherwise.

**Note**: The palindrome check is case-insensitive and ignores all whitespace characters (like spaces, tabs, or newlines) due to the normalization step.

**Output Example**: Calling is\_palindrome("Taco cat") returns True.