Table of Contents

## 1 FunctionDef fibonacci(n)

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

**parameters**: The parameters of this Function. · n: The 0-indexed integer representing the position in the Fibonacci sequence to compute.

**Code Description**: The function begins by checking if the input n is a negative number. If it is, a ValueError is raised with the message “n must be non-negative”. Next, two variables, a and b, are initialized to 0 and 1 respectively, representing the first two numbers in the Fibonacci sequence. The code then 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 old a and b is assigned to b. This process effectively calculates the next number in the sequence. After the loop has finished, the function returns the final value of a, which holds the nth Fibonacci number.

**Note**: The function expects a non-negative integer as input. Providing a negative integer will result in a ValueError. The sequence is 0-indexed, meaning fibonacci(0) returns the first number, 0.

**Output Example**: Calling fibonacci(9) returns 34. ## FunctionDef invert\_dictionary(mapping) **invert\_dictionary**: The function of invert\_dictionary is to create a new dictionary by swapping the keys and values of a given dictionary, ensuring the original values are unique.

**parameters**: The parameters of this Function. · mapping: A dictionary with string keys and integer values, where all values are expected to be unique.

**Code Description**: The function first validates the input dictionary mapping. It compares the number of values in the dictionary with the number of unique values by converting the values to a set. If the counts are not equal, it indicates that the original dictionary contains duplicate values. In this case, a ValueError is raised with the message “Values must be unique to invert dictionary”. If all values are unique, the function proceeds to use 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 created inverted dictionary is then returned.

**Note**: This function will raise a ValueError if the input dictionary contains duplicate values, as this would result in key collisions in the inverted dictionary. The keys of the input dictionary must be strings and the values must be integers as specified by the type hints.

**Output Example**: Calling invert\_dictionary({'apple': 1, 'banana': 2}) returns {1: 'apple', 2: 'banana'}. ## FunctionDef is\_palindrome(text) **is\_palindrome**: The function of is\_palindrome is to determine if a given string is a palindrome, disregarding character casing and any whitespace.

**parameters**: The parameters of this Function. · text: The string to be checked for the palindrome property.

**Code Description**: The function first normalizes the input text. It creates a new string called normalized by iterating through each character of the input text. During this process, it discards any character that is a whitespace character (using ch.isspace()) and converts all other characters to their lowercase equivalents. These processed characters are then joined together into a single string.

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

**Note**: The check is case-insensitive. All whitespace characters (spaces, tabs, newlines, etc.) are ignored, not just simple spaces.

**Output Example**: Calling is\_palindrome("No lemon no melon") returns True.