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: Index of the Fibonacci number (0-indexed).

**Code Description**: The function fibonacci takes an integer n as input, representing the desired Fibonacci number’s index. It first checks if n is negative, raising a ValueError if it is. If n is non-negative, it initializes two variables, a and b, to 0 and 1, respectively, representing the first two Fibonacci numbers. It then iterates n times, updating a and b in each iteration. In each iteration, a takes the value of b, and b becomes the sum of the previous a and b. After the loop completes, the function returns the value of a, which represents the nth Fibonacci number.

**Note**: The function uses an iterative approach to calculate the Fibonacci number, which is generally more efficient than a recursive approach for larger values of n. The function handles the edge case of negative input by raising a ValueError.

**Output Example**: Calling fibonacci(6) returns 8.

## 2 FunctionDef invert\_dictionary(mapping)

**invert\_dictionary**: The function of invert\_dictionary is to invert a dictionary, mapping its values to its keys, given that the values are unique.

**parameters**: The parameters of this Function. · mapping: Dictionary mapping strings to integers with unique values.

**Code Description**: The function invert\_dictionary takes a dictionary mapping as input, where keys are strings and values are integers. It first checks if the values in the dictionary are unique by comparing the length of the values list to the length of the set of values. If the values are not unique, it raises a ValueError. If the values are unique, it uses a dictionary comprehension to create a new dictionary where the keys and values are swapped. The new dictionary is then returned.

**Note**: The function raises a ValueError if the input dictionary’s values are not unique. This is to ensure that the inverted dictionary is well-defined (i.e., each key maps to a single value).

**Output Example**: If the input is {'a': 1, 'b': 2, 'c': 3}, the output will be {1: 'a', 2: 'b', 3: 'c'}.

## 3 FunctionDef is\_palindrome(text)

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

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

**Code Description**: The function first normalizes the input string text by converting all characters to lowercase and removing spaces. It achieves this using a generator expression within the ''.join() method. The generator (ch.lower() for ch in text if not ch.isspace()) iterates through each character ch in the input text. If the character is not a space (if not ch.isspace()), it’s converted to lowercase (ch.lower()) and included in the generated sequence. The ''.join() method then concatenates these characters to form the normalized string. Finally, the function compares the normalized string with its reverse (normalized[::-1]). If they are identical, the function returns True, indicating that the original string is a palindrome; otherwise, it returns False.

**Note**: The function handles strings with mixed casing and spaces.

**Output Example**: Calling is\_palindrome(“Racecar”) returns True. Calling is\_palindrome(“A man, a plan, a canal: Panama”) returns True. Calling is\_palindrome(“hello”) returns False.