Table of Contents

## 1 FunctionDef fibonacci(n)

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

**parameters**: · n: An integer representing the 0-indexed position in the Fibonacci sequence for which the number is to be calculated.

**Code Description**: The function fibonacci calculates the Fibonacci number at a specified index n. It begins by validating the input n. If n is a negative number, a ValueError is raised, as the Fibonacci sequence is defined for non-negative integers. The function initializes two variables, a and b, to 0 and 1, respectively. These represent the first two numbers in the sequence, F(0) and F(1). A for loop then iterates n times. In each iteration, the values of a and b are updated simultaneously: a is assigned the current value of b, and b is assigned the sum of the previous a and b. This process effectively advances through the Fibonacci sequence. After the loop completes, the variable a holds the nth Fibonacci number, which is then returned as the result.

**Note**: The input n must be a non-negative integer. The function uses a 0-based index, so fibonacci(0) returns the first number in the sequence, which is 0. This iterative implementation is memory-efficient, especially for large values of n, compared to a simple recursive approach.

**Output Example**: 55 ## FunctionDef invert\_dictionary(mapping) **invert\_dictionary**: The function of invert\_dictionary is to swap the keys and values of a given dictionary, creating a new dictionary where the original values are the keys and the original keys are the values. **parameters**: · mapping: A dictionary of type Dict[str, int] where keys are strings and values are integers. The integer values within this dictionary must be unique. **Code Description**: The function first performs a validation check on the input mapping to ensure that all its values are unique. It achieves this by comparing the total number of values, obtained from len(mapping.values()), with the number of unique values, determined by converting the values to a set and getting its length, len(set(mapping.values())). Since a set can only contain unique elements, any difference in these lengths indicates the presence of duplicate values. If duplicates are found, the function raises a ValueError with the message “Values must be unique to invert dictionary”. This validation is critical because the values of the input dictionary become the keys of the output dictionary, and dictionary keys must be unique. If the validation passes, the function uses a dictionary comprehension, {value: key for key, value in mapping.items()}, to iterate through each key-value pair of the input mapping. It constructs and returns a new dictionary where each original value is now a key, and its corresponding original key is its value. **Note**: This function will fail and raise a ValueError if the input dictionary contains duplicate values. The uniqueness of values is a strict requirement for a successful inversion. **Output Example**: Given a valid input dictionary:

input\_mapping = {'apple': 10, 'banana': 20, 'cherry': 30}

The function will return the following inverted dictionary:

{10: 'apple', 20: 'banana', 30: 'cherry'}

If an invalid input with duplicate values is provided:

input\_mapping = {'apple': 10, 'banana': 20, 'date': 10}

The function will raise an error:

ValueError: Values must be unique to invert dictionary

## 2 FunctionDef is\_palindrome(text)

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

**parameters**: · text: The input string of type str that needs to be checked.

**Code Description**: The function is\_palindrome takes a single string argument text. It first normalizes the input string by creating a new string called normalized. This is achieved by iterating through each character (ch) of the input text. For each character, it checks if it is a whitespace character using the isspace() method. If the character is not a space, it is converted to its lowercase equivalent using lower() and then added to the normalized string. This process effectively removes all spaces and makes the comparison case-insensitive. Finally, the function compares the normalized string with its reverse, which is obtained using the slice notation [::-1]. If the normalized string is identical to its reversed version, the function returns True, indicating the original text is a palindrome. Otherwise, it returns False.

**Note**: This function is case-insensitive and ignores all whitespace characters within the input string when performing the palindrome check. For example, “Taco Cat” and “tacocat” would both be considered palindromes.

**Output Example**:

# Calling the function with a palindrome string  
is\_palindrome("A man a plan a canal Panama")  
# Expected output:  
True  
  
# Calling the function with a non-palindrome string  
is\_palindrome("hello world")  
# Expected output:  
False