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 approach.

## 1.2 parameters

* n (int): The 0-indexed position in the Fibonacci sequence for which to find the value.

## 1.3 Description

This function calculates the nth Fibonacci number, where the sequence starts with 0 and 1.

The function first validates the input n. If n is a negative number, it raises a ValueError because the Fibonacci sequence is not defined for negative indices.

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

The function then enters a for loop that iterates n times. In each iteration, the values of a and b are updated simultaneously using tuple assignment: a, b = b, a + b. This operation effectively shifts the sequence forward: the new a becomes the old b, and the new b becomes the sum of the old a and b.

After the loop completes, the variable a holds the nth Fibonacci number, which is then returned. For an input of n=0, the loop does not execute, and the initial value of a (0) is correctly returned.

# Initialization  
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 sequence is 0, 1, 1, 2. The 3rd element (0-indexed) is 2.

## 1.4 Usage Notes

* The input n must be a non-negative integer. Providing a negative integer will result in a ValueError.
* The function uses a 0-indexed sequence. For example, fibonacci(0) returns the first number (0), and fibonacci(1) returns the second number (1).
* This iterative implementation is efficient in terms of memory and performance compared to a naive recursive approach, as it avoids redundant calculations and deep recursion stacks.

**Output Example**: The function returns a single integer value.

55

## 1.5 Example

# Example usage  
# Calculate the 10th Fibonacci number (0-indexed)  
result = fibonacci(10)  
print(result)

**Output:**

55

## 1.6 FunctionDef invert\_dictionary(mapping)

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

## 2.1 Overview

The invert\_dictionary function inverts a given dictionary by swapping its keys and values, creating a new dictionary where the original values become the keys and the original keys become the values.

## 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 for the inversion to be possible.

## 2.3 Description

This function provides a safe way to invert a dictionary. The core logic involves two main steps:

1. **Uniqueness Validation**: Before performing the inversion, the function first validates that all values in the input mapping dictionary are unique. It does this by comparing the number of values (len(mapping.values())) with the number of unique values (len(set(mapping.values()))). If these counts are not equal, it indicates the presence of duplicate values, which would lead to data loss during inversion (as dictionary keys must be unique). In this case, a ValueError is raised with the message “Values must be unique to invert dictionary”.
2. **Inversion**: If the values are unique, the function proceeds to invert the 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 creates a new entry in the returned dictionary where the original value is the new key and the original key is the new value.

# Internal logic for inversion  
# This is executed only if values are unique  
inverted = {value: key for key, value in mapping.items()}

## 2.4 Usage Notes

* The most critical requirement for this function is that the values of the input dictionary must be unique. The function will raise a ValueError if any duplicate values are found.
* The input dictionary is expected to have string keys and integer values, and the output will be a dictionary with integer keys and string values, as per the type hints.

**Output Example**: A new dictionary with keys and values swapped.

## 2.5 Example

# Example usage with a valid dictionary  
valid\_mapping = {'user\_one': 101, 'user\_two': 102, 'user\_three': 103}  
inverted\_mapping = invert\_dictionary(valid\_mapping)  
print(inverted\_mapping)  
  
# Example of an invalid dictionary that will raise an error  
try:  
 invalid\_mapping = {'apple': 5, 'banana': 10, 'cherry': 5}  
 invert\_dictionary(invalid\_mapping)  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

{101: 'user\_one', 102: 'user\_two', 103: 'user\_three'}  
Error: 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 checks if a given string is a palindrome, meaning it reads the same forwards and backwards, after ignoring character casing and spaces.

## 3.2 parameters

* **text** (str): The string that will be checked to determine if it is a palindrome.

## 3.3 Description

This function determines if a string is a palindrome through a two-step process: normalization and comparison.

First, the function normalizes the input text. It iterates through each character of the string. For each character, it discards any whitespace (like spaces, tabs, or newlines) using ch.isspace() and converts the remaining characters to lowercase using ch.lower(). These processed characters are then joined together to form a new, clean string stored in the normalized variable.

For example, an input of "Taco Cat" would be transformed into "tacocat".

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

Second, the function compares the normalized string with its reverse. The reversal is achieved using Python’s slice notation [::-1]. If the normalized string is identical to normalized[::-1], the expression evaluates to True, indicating the original text is a palindrome. Otherwise, it evaluates to False. This boolean result is then returned.

return normalized == normalized[::-1]

## 3.4 Usage Notes

* **Case-Insensitive**: The comparison ignores the original casing of the letters. For example, is\_palindrome("Racecar") and is\_palindrome("racecar") both return True.
* **Whitespace Removal**: All whitespace characters are removed from the string before the check is performed.
* **Punctuation and Numbers**: The function does not remove punctuation or numbers. A string like "madam, I'm Adam" will be normalized to "madam,i'madam", which is not a palindrome, and the function will correctly return False.

**Output Example**: For an input of "No lemon no melon", the internal normalized string becomes "nolemonnomelon". The function then compares this to its reverse, which is also "nolemonnomelon", and returns True.

## 3.5 Example

# Example 1: A classic palindrome with mixed case and spaces  
result1 = is\_palindrome("A man a plan a canal Panama")  
print(f"'A man a plan a canal Panama' is a palindrome: {result1}")  
  
# Example 2: A simple non-palindrome  
result2 = is\_palindrome("Hello World")  
print(f"'Hello World' is a palindrome: {result2}")  
  
# Example 3: A string that is not a palindrome due to punctuation  
result3 = is\_palindrome("Eva, can I see bees in a cave?")  
print(f"'Eva, can I see bees in a cave?' is a palindrome: {result3}")

**Output:**

'A man a plan a canal Panama' is a palindrome: True  
'Hello World' is a palindrome: False  
'Eva, can I see bees in a cave?' is a palindrome: False