Table of Contents

## 0.1 FunctionDef fibonacci(n)

# 1 Function: fibonacci(n: int)

## 1.1 Overview

The fibonacci function computes the nth Fibonacci number using an iterative approach.

## 1.2 parameters

* **n** (int): The 0-indexed position in the Fibonacci sequence for which to find the number. This value must be non-negative.

## 1.3 Description

This function calculates a number in the Fibonacci sequence based on its index n. The Fibonacci sequence starts with 0 and 1, and each subsequent number is the sum of the two preceding ones (0, 1, 1, 2, 3, 5, …).

The function begins by validating the input n. If n is a negative number, it raises a ValueError because 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 sequence, F(0) and F(1).

The core logic resides in 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 tuple assignment (a, b = b, a + b) efficiently calculates the next number in the sequence without needing a temporary variable.

After the loop completes, the variable a holds the nth Fibonacci number, which is then returned.

# Inside the loop for n=3:  
# Initial: a=0, b=1  
# Iteration 1: a=1, b=1 (b, a+b -> 1, 0+1)  
# Iteration 2: a=1, b=2 (b, a+b -> 1, 1+1)  
# Iteration 3: a=2, b=3 (b, a+b -> 2, 1+2)  
# Loop ends. Returns a, which 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 memory-efficient and avoids the recursion depth limits that can be an issue with naive recursive solutions, making it suitable for calculating larger Fibonacci numbers.

**Output Example**: The function returns a single integer value. For an input of 9, the output would look like: 34

## 1.5 Example

# Example 1: Calculate the 9th Fibonacci number (0-indexed)  
fib\_9 = fibonacci(9)  
print(f"The 9th Fibonacci number is: {fib\_9}")  
  
# Example 2: Calculate the 0th Fibonacci number  
fib\_0 = fibonacci(0)  
print(f"The 0th Fibonacci number is: {fib\_0}")  
  
# Example 3: Attempting to use a negative index will raise an error  
try:  
 fibonacci(-1)  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

The 9th Fibonacci number is: 34  
The 0th Fibonacci number is: 0  
Error: n must be non-negative

## 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, returning a new dictionary.

## 2.2 parameters

* **mapping** (Dict[str, int]): A dictionary where keys are strings and values are integers. It is crucial 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 mapping strings to integers.

The primary logic is executed in two steps: 1. **Validation**: Before attempting to invert the dictionary, the function first validates that all values in the input mapping are unique. It does this by comparing the number of values with the number of unique values: len(mapping.values()) != len(set(mapping.values())). If the counts do not match, it indicates duplicate values are present, which would cause data loss during inversion (as multiple original keys would map to the same new key). In this case, a ValueError is raised.

1. **Inversion**: If all values are unique, the function proceeds with the inversion. It uses a dictionary comprehension: {value: key for key, value in mapping.items()}. This iterates through each key-value pair of the original mapping dictionary and creates a new dictionary where the original value becomes the new key and the original key becomes the new value.

For example, a pair ('apple', 1) in the input dictionary becomes (1, 'apple') in the output dictionary.

## 2.4 Usage Notes

* The function requires that all values in the input mapping dictionary be unique. If duplicate values are found, it will raise a ValueError.
* This function is not an in-place operation. It returns a new dictionary and does not modify the original mapping dictionary.
* The type hints indicate an input of Dict[str, int] and an output of Dict[int, str], but the logic will work for any dictionary with hashable keys and unique, hashable values.

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

{  
 1: "one",  
 2: "two",  
 3: "three"  
}

## 2.5 Example

# Example 1: Successful inversion with unique values  
original\_dict = {'one': 1, 'two': 2, 'three': 3}  
inverted\_dict = invert\_dictionary(original\_dict)  
print(f"Original: {original\_dict}")  
print(f"Inverted: {inverted\_dict}")  
  
# Example 2: Attempted inversion with duplicate values  
try:  
 duplicate\_value\_dict = {'a': 1, 'b': 2, 'c': 1}  
 print(f"\nAttempting to invert: {duplicate\_value\_dict}")  
 invert\_dictionary(duplicate\_value\_dict)  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

Original: {'one': 1, 'two': 2, 'three': 3}  
Inverted: {1: 'one', 2: 'two', 3: 'three'}  
  
Attempting to invert: {'a': 1, 'b': 2, 'c': 1}  
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 determines if a given string is a palindrome, ignoring letter casing and whitespace characters.

## 3.2 parameters

* text: str - The input string to be checked.

## 3.3 Description

This function evaluates whether a string reads the same forwards and backwards. It performs this check in two main steps:

1. **Normalization**: The function first creates a “normalized” version of the input text. It iterates through each character of the string, converts it to lowercase using ch.lower(), and discards any whitespace characters (like spaces, tabs, or newlines) using ch.isspace(). The resulting characters are then joined to form a new, clean string. For example, "Taco Cat" becomes "tacocat".
2. **Comparison**: The normalized string is then compared to its own reverse. The reversal is efficiently achieved using Python’s slice notation [::-1]. If the normalized string is identical to its reversed version, the function returns True, indicating it is a palindrome. Otherwise, it returns False.

# Internal logic breakdown  
text = "No lemon, no melon"  
  
# 1. Normalization  
# Filters out spaces and converts to lowercase  
normalized = ''.join(ch.lower() for ch in text if not ch.isspace())  
# normalized becomes "nolemon,nomelon"  
  
# 2. Comparison  
# The function checks if "nolemon,nomelon" == "nolem,onomelon"  
# In this case, it's False because of the comma.  
return normalized == normalized[::-1]

## 3.4 Usage Notes

* The function is case-insensitive. For example, Racecar is treated as racecar.
* All whitespace characters (spaces, tabs, newlines, etc.) are completely ignored during the check.
* Punctuation, numbers, and other symbols are **not** ignored and are included in the palindrome check. For instance, racecar! will be evaluated as False because the normalized string racecar! is not equal to its reverse !racecar.

**Output Example**: The function returns a boolean value.

True

## 3.5 Example

# Example 1: A classic palindrome with mixed casing 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 palindrome check with punctuation (will fail)  
result3 = is\_palindrome("madam i'm adam")  
print(f"'madam i'm adam' is a palindrome: {result3}")

**Output:**

'A man a plan a canal Panama' is a palindrome: True  
'Hello World' is a palindrome: False  
'madam i'm adam' is a palindrome: False