Table of Contents

## 0.1 FunctionDef fibonacci(n)

# 1 Function: fibonacci

## 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 number.

## 1.3 Description

This function calculates the Fibonacci number at a specific index n. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, starting from 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 the first two numbers in the sequence, 0 and 1, respectively. It then enters a loop that iterates n times. In each iteration, it updates the values of a and b to the next pair in the sequence using tuple unpacking: a takes the current value of b, and 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.

# 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.  
# F(3) = 2

## 1.4 Usage Notes

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

**Output Example**: The function returns a single integer representing the Fibonacci number at the specified index.

## 1.5 Example

# Example usage  
# Calculate the 9th Fibonacci number (0-indexed)  
result = fibonacci(9)  
print(result)

**Output:**

34

## 1.6 FunctionDef invert\_dictionary(mapping)

# 2 Function: invert\_dictionary

## 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 with string keys and unique integer values that will be inverted.

## 2.3 Description

This function takes a dictionary, mapping, as input and creates a new dictionary where the keys and values are swapped.

The function first performs a critical validation check to ensure that the inversion is possible. Since dictionary keys must be unique, the values of the input dictionary must also be unique to serve as keys in the new, inverted dictionary. This is checked by comparing the number of values with the number of unique values: len(mapping.values()) != len(set(mapping.values())). If any duplicate values are found, a ValueError is raised with a descriptive message.

If all values are unique, the function proceeds to create the inverted dictionary using a dictionary comprehension: {value: key for key, value in mapping.items()}. This expression iterates through each key-value pair of the input mapping and constructs a new dictionary where each original value becomes a new key, and its corresponding original key becomes the new value.

# The core logic for inversion  
# This runs only after values are confirmed to be unique  
inverted\_dict = {value: key for key, value in mapping.items()}

## 2.4 Usage Notes

* The primary constraint of this function is that all values in the input mapping dictionary must be unique.
* If the values are not unique, the function will raise a ValueError, as a valid inverted dictionary cannot be created with duplicate keys.

**Output Example**: A dictionary where the keys are integers and the values are strings.

{  
 1: "key\_one",  
 2: "key\_two"  
}

## 2.5 Example

# Example 1: Successful inversion with unique values  
from typing import Dict  
  
def invert\_dictionary(mapping: Dict[str, int]) -> Dict[int, str]:  
 """Invert a dictionary with unique values."""  
 if len(mapping.values()) != len(set(mapping.values())):  
 raise ValueError("Values must be unique to invert dictionary")  
 return {value: key for key, value in mapping.items()}  
  
original\_dict = {"alpha": 10, "beta": 20, "gamma": 30}  
inverted\_result = invert\_dictionary(original\_dict)  
print(f"Original: {original\_dict}")  
print(f"Inverted: {inverted\_result}")  
  
# Example 2: Attempted inversion with duplicate values  
try:  
 invalid\_dict = {"alpha": 10, "beta": 20, "delta": 10}  
 print(f"\nAttempting to invert: {invalid\_dict}")  
 invert\_dictionary(invalid\_dict)  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

Original: {'alpha': 10, 'beta': 20, 'gamma': 30}  
Inverted: {10: 'alpha', 20: 'beta', 30: 'gamma'}  
  
Attempting to invert: {'alpha': 10, 'beta': 20, 'delta': 10}  
Error: Values must be unique to invert dictionary

## 2.6 FunctionDef is\_palindrome(text)

# 3 Function: is\_palindrome

## 3.1 Overview

The is\_palindrome function determines if a given text string is a palindrome, meaning it reads the same forwards and backwards, while ignoring character casing and spaces.

## 3.2 parameters

* **text** (str): The string to be checked.

## 3.3 Description

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

First, it normalizes the input text to create a consistent format for comparison. This is done by iterating through each character (ch) in the text string. For each character, it is converted to lowercase using ch.lower(), and any whitespace characters are filtered out using if not ch.isspace(). The resulting characters are then joined together to form a new string, normalized.

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 efficiently achieved using Python’s slice notation [::-1]. If the normalized string is identical to normalized[::-1], the function returns True, indicating the original text is a palindrome. Otherwise, it returns False.

return normalized == normalized[::-1]

## 3.4 Usage Notes

* The function is case-insensitive. For example, “Racecar” and “racecar” will both be treated as palindromes.
* All whitespace characters (spaces, tabs, newlines, etc.) are ignored during the check.
* Punctuation and special characters are **not** ignored and will be included in the palindrome check. For example, is\_palindrome("A man, a plan, a canal: Panama") would return False because the commas and colon are not symmetrical.

**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 string  
result2 = is\_palindrome("hello world")  
print(f"'hello world' is a palindrome: {result2}")  
  
# Example 3: A palindrome with numbers  
result3 = is\_palindrome("123 321")  
print(f"'123 321' is a palindrome: {result3}")

**Output:**

'A man a plan a canal Panama' is a palindrome: True  
'hello world' is a palindrome: False  
'123 321' is a palindrome: True