Table of Contents

## 0.1 FunctionDef num(a, b)

# 1 Function: def num(a, b)

## 1.1 Overview

The num function calculates and returns the sum of two input arguments.

## 1.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| a | int / float | The first number to be added. |
| b | int / float | The second number to be added. |

## 1.3 Description

This function provides a straightforward way to perform addition. It accepts two parameters, a and b. Internally, it uses the standard Python addition operator (+) to compute the sum of these two values. The resulting sum is then returned as the output of the function.

For example, if a is 5 and b is 10, the function will compute 5 + 10 and return the value 15.

# The function simply returns the result of a + b  
return a + b

## 1.4 Usage Notes

* The function is designed for numeric types like integers and floats. However, it will work with any two types that support the addition (+) operator, such as strings (for concatenation).
* The function does not include any error handling. Passing incompatible types (e.g., an integer and a string) will result in a TypeError.

**Output Example**: A numeric value representing the sum of the inputs.

## 1.5 Example

# Example usage with two integers  
result = num(15, 25)  
print(result)  
  
# Example usage with two floating-point numbers  
float\_result = num(10.5, 2.2)  
print(float\_result)

**Output:**

40  
12.7

## 1.6 FunctionDef generate\_random\_integers(count, start, end)

# 2 Function: generate\_random\_integers(count: int, start: int = 0, end: int = 100) -> List[int]

## 2.1 Overview

The generate\_random\_integers function returns a list of a specified number of pseudo-random integers within a given inclusive range.

## 2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| count | int | The number of integers to generate in the list. |
| start | int | The inclusive lower bound for the random values. Defaults to 0. |
| end | int | The inclusive upper bound for the random values. Defaults to 100. |

## 2.3 Description

This function provides a convenient way to generate a list of random integers. It operates in three main steps:

1. **Input Validation**: It first checks if the count parameter is a non-negative number. If count is less than 0, it raises a ValueError to prevent invalid list creation.
2. **Range Correction**: The function ensures the range is valid by comparing start and end. If start is found to be greater than end, it automatically swaps their values. This allows the user to provide the bounds in any order without causing an error.
3. **List Generation**: Using a list comprehension, the function iterates count times. In each iteration, it calls random.randint(start, end) to produce a single integer that is uniformly sampled from the inclusive range [start, end]. These integers are collected into a list which is then returned.

# Internal logic for generating the list  
return [random.randint(start, end) for \_ in range(count)]

## 2.4 Usage Notes

* This function requires the random module to be imported in the script.
* A ValueError will be raised if the count argument is a negative integer.
* The function is flexible with range ordering; if start > end, the values will be swapped internally to form a valid range.
* The returned integers are inclusive of both the start and end values.

**Output Example**: A list of integers. The actual values will be random.

[42, 17, 89, 3, 55]

## 2.5 Example

import random  
from typing import List  
  
# The function definition from the documentation  
def generate\_random\_integers(count: int, start: int = 0, end: int = 100) -> List[int]:  
 """Return a list of pseudo-random integers.  
  
 Parameters:  
 count: Number of integers to generate.  
 start: Inclusive lower bound for values.  
 end: Inclusive upper bound for values.  
  
 Returns:  
 A list containing `count` integers sampled uniformly in [start, end].  
 """  
 if count < 0:  
 raise ValueError("count must be non-negative")  
 if start > end:  
 start, end = end, start  
 return [random.randint(start, end) for \_ in range(count)]  
  
# Example usage: Generate 5 random integers between 10 and 50.  
random\_numbers = generate\_random\_integers(count=5, start=10, end=50)  
print(random\_numbers)

**Output:**

(Note: The output is random and will vary with each execution)

[23, 48, 11, 35, 19]

## 2.6 FunctionDef fibonacci(n)

# 3 Function: fibonacci(n: int)

## 3.1 Overview

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

## 3.2 parameters

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

## 3.3 Description

This function calculates a Fibonacci number based on its index n. It begins by validating the input, raising a ValueError if n is a negative number, as the Fibonacci sequence is not defined for negative indices.

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).

It then enters 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 process effectively shifts the sequence forward one step:

# Inside the loop  
a, b = b, a + b

After the loop has completed n iterations, the variable a will hold the value of the nth Fibonacci number. The function then returns this value. For an input of n=0, the loop does not execute, and the initial value of a (0) is correctly returned.

## 3.4 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, where fibonacci(0) returns 0, fibonacci(1) returns 1, and so on.
* This iterative implementation is memory-efficient compared to a naive recursive approach, as it avoids deep recursion stacks.

**Output Example**: A single integer representing the Fibonacci number at the specified index.

13

## 3.5 Example

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

**Output:**

34

## 3.6 FunctionDef choose\_random\_item(items)

# 4 Function: choose\_random\_item(items: List[str])

## 4.1 Overview

The choose\_random\_item function selects and returns a single, random element from a given list of strings.

## 4.2 parameters

* items (List[str]): A non-empty list of strings from which one item will be randomly selected.

## 4.3 Description

This function provides a simple way to pick one item uniformly at random from a sequence.

The function first validates the input list items. It checks if the list is empty using if not items:. If the list is found to be empty, it raises a ValueError with the message “items must not be empty”. This prevents errors from occurring in the subsequent random.choice call, which cannot operate on an empty sequence.

If the list is not empty, the function proceeds to call random.choice(items). This standard library function takes a non-empty sequence as an argument and returns a single element chosen from it at random, where each element has an equal chance of being selected. The chosen string is then returned as the output of the choose\_random\_item function.

# Internal logic  
if not items:  
 raise ValueError("items must not be empty")  
return random.choice(items)

## 4.4 Usage Notes

* The input list items must not be empty. Providing an empty list will raise a ValueError.
* This function depends on Python’s random module. Ensure it is imported in the script where this function is used (e.g., import random).
* The selection is uniformly random, meaning every item in the list has an equal probability of being chosen.

**Output Example**: A single string from the input list.

## 4.5 Example

import random  
from typing import List  
  
# Definition of the function  
def choose\_random\_item(items: List[str]) -> str:  
 """Choose a single random item from a non-empty sequence."""  
 if not items:  
 raise ValueError("items must not be empty")  
 return random.choice(items)  
  
# Example usage  
options = ["red", "green", "blue", "yellow"]  
selected\_color = choose\_random\_item(options)  
print(f"The randomly selected color is: {selected\_color}")

**Output:**

The randomly selected color is: blue

*(Note: The actual output will vary with each execution as it is chosen randomly.)*

## 4.6 FunctionDef shuffle\_copy(items)

# 5 Function: shuffle\_copy(items: List[int])

## 5.1 Overview

The shuffle\_copy function returns a new, randomly shuffled copy of a given list of integers without altering the original list.

## 5.2 parameters

* items (List[int]): A list of integers to be copied and shuffled.

## 5.3 Description

This function provides a safe way to shuffle a list by ensuring the original data structure is not mutated. The process is straightforward:

1. A shallow copy of the input items list is created using the list() constructor. This new list is assigned to a local variable copy. This step is crucial for preserving the original list.
2. The random.shuffle() function is then called on the copy. The random.shuffle() method shuffles the elements of a sequence in-place.
3. Finally, the function returns the modified copy, which now contains the same elements as the original items list but in a random order.

# Internal logic  
copy = list(items)  
random.shuffle(copy)  
return copy

## 5.4 Usage Notes

* This function is non-destructive. The original list passed as the items parameter will remain unchanged after the function call.
* The function depends on Python’s random module. Ensure this module is imported (e.g., import random) before calling this function.
* The output order is random and will be different on each execution.

**Output Example**: A possible return value for an input of [1, 2, 3, 4, 5].

[4, 1, 5, 3, 2]

## 5.5 Example

import random  
from typing import List  
  
def shuffle\_copy(items: List[int]) -> List[int]:  
 """Return a shuffled copy of the given list without mutating the input.  
  
 Parameters:  
 items: A list of integers.  
  
 Returns:  
 A new list containing the same integers in random order.  
 """  
 copy = list(items)  
 random.shuffle(copy)  
 return copy  
  
# Example usage  
original\_list = [10, 20, 30, 40, 50]  
shuffled\_list = shuffle\_copy(original\_list)  
  
print(f"Original List: {original\_list}")  
print(f"Shuffled Copy: {shuffled\_list}")

**Output:**

Original List: [10, 20, 30, 40, 50]  
Shuffled Copy: [30, 50, 10, 20, 40]