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 provided 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 performs a basic addition operation. It takes two parameters, a and b, which are expected to be numerical values (integers or floats). The core logic of the function is a single return statement that executes the expression a + b. The result of this addition is then immediately returned as the output of the function.

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

# The core logic of the function  
return a + b

## 1.4 Usage Notes

* The function relies on Python’s + operator. Ensure that both a and b are of compatible types for addition.
* If string values are passed as arguments, the function will perform string concatenation instead of mathematical addition. For example, num("hello", "world") would return "helloworld".
* Providing incompatible types (e.g., an integer and a string) will result in a TypeError.

**Output Example**: If the inputs are 10 and 20, the returned value will be: 30

## 1.5 Example

# Example usage with two integers  
result = num(15, 30)  
print(result)  
  
# Example usage with a float and an integer  
float\_result = num(99.5, 10)  
print(float\_result)

**Output:**

45  
109.5

## 1.6 FunctionDef generate\_random\_integers

# 2 Function: generate\_random\_integers(count: int, start: int = 0, end: int = 100)

## 2.1 Overview

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

## 2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| count | int | The total number of integers to generate. |
| 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 straightforward way to create a list of random integers. The process is as follows:

1. **Input Validation**: The function first checks if the count parameter is a non-negative number. If count is less than zero, it raises a ValueError to prevent invalid list creation.
2. **Range Correction**: It then compares the start and end parameters. If start is found to be greater than end, the function automatically swaps their values. This ensures that random.randint receives a valid, ordered range, making the function more robust to user input.
3. **Random Number Generation**: Finally, it uses a list comprehension to generate the random numbers. It iterates count times, and in each iteration, it calls random.randint(start, end). This call produces a single integer uniformly sampled from the inclusive range [start, end]. The generated integers are collected into a list which is then returned.

# Internal logic for swapping an inverted range  
if start > end:  
 start, end = end, start

## 2.4 Usage Notes

* The count parameter must be a non-negative integer. Providing a negative value will raise a ValueError.
* If the provided start value is greater than the end value, the function will automatically swap them to form a valid range. For instance, calling generate\_random\_integers(5, 50, 10) will be executed as if it were generate\_random\_integers(5, 10, 50).
* The generated integers are inclusive of both the start and end bounds.
* This function depends on Python’s built-in random module.

**Output Example**: A list containing the specified count of integers.

[42, 8, 99, 23, 75]

## 2.5 Example

# This function requires the 'random' module to be imported first.  
import random  
  
# Example: Generate 5 random integers between 10 and 20 (inclusive).  
random\_list = generate\_random\_integers(count=5, start=10, end=20)  
print(random\_list)

**Output:**

[15, 11, 20, 18, 10]  
# Note: The actual output will vary with each execution due to its random nature.

## 2.6 FunctionDef fibonacci

# 3 Function: fibonacci(n: int)

## 3.1 Overview

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

## 3.2 parameters

* n: int - The 0-based index of the Fibonacci number to compute.

## 3.3 Description

This function provides a memory-efficient way to calculate a number in the Fibonacci sequence. The logic proceeds as follows:

1. It first validates the input n. If n is a negative number, a ValueError is raised, as the Fibonacci sequence is not defined for negative indices.
2. Two variables, a and b, are initialized to 0 and 1 respectively. These represent the first two numbers in the sequence, F(0) and F(1).
3. A for loop iterates n times. In each iteration, the values of a and b are updated. The current value of b is assigned to a, and the sum of the old a and b is assigned to b. This is achieved in a single step using tuple assignment: a, b = b, a + b.
4. After the loop completes, a holds the value of 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.

# Inside the loop for n=5:  
# Initial: a=0, b=1  
# 1. a=1, b=1  
# 2. a=1, b=2  
# 3. a=2, b=3  
# 4. a=3, b=5  
# 5. a=5, b=8  
# Loop ends, returns a, which is 5.

## 3.4 Usage Notes

* The input n must be a non-negative integer. Providing a negative value will result in a ValueError.
* The function uses 0-indexing, meaning fibonacci(0) returns the first element of the sequence, which is 0.
* This iterative implementation is efficient in terms of memory and performance for large values of n 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

## 3.5 Example

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

**Output:**

55

## 3.6 FunctionDef choose\_random\_item

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

## 4.1 Overview

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

## 4.2 parameters

* **items** (List[str]): A non-empty list of strings from which to choose a random item.

## 4.3 Description

This function provides a simple way to get a random element from a list. The core logic is built around Python’s random.choice() method.

First, the function performs a validation check to ensure the input list items is not empty. If items is an empty list, the function will raise a ValueError with the message “items must not be empty”. This prevents runtime errors from the underlying random.choice() function, which cannot operate on an empty sequence.

If the list is valid (i.e., contains at least one element), the function then calls random.choice(items). This method selects a single item from the list with a uniform probability, meaning every item has an equal chance of being chosen. The selected string is then returned as the result.

# Internal logic for a non-empty list  
import random  
items = ["apple", "banana", "cherry"]  
# The following line is the core operation  
selected\_item = random.choice(items)  
# selected\_item could be "apple", "banana", or "cherry"

## 4.4 Usage Notes

* The input list items must not be empty. Providing an empty list will result in a ValueError.
* This function depends on Python’s random module. Ensure it is imported in the execution environment.
* The selection is uniformly random, giving each item in the list an equal probability of being chosen.

**Output Example**: The function returns a single string from the provided list.

"banana"

## 4.5 Example

import random  
from typing import List  
  
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:  
fruits = ["apple", "banana", "cherry", "date", "elderberry"]  
random\_fruit = choose\_random\_item(fruits)  
print(f"The randomly chosen fruit is: {random\_fruit}")  
  
# Example of error handling  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

The output for the successful call is random. One possible result is shown below.

The randomly chosen fruit is: cherry  
Error: items must not be empty

## 4.6 FunctionDef shuffle\_copy

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

## 5.1 Overview

The shuffle\_copy function creates and returns a shuffled copy of a given list, ensuring the original list remains unchanged.

## 5.2 parameters

* items (List[int]): A list of integers that you want to shuffle.

## 5.3 Description

This function provides a safe way to shuffle a list without altering the original data structure. The process is straightforward and involves three main steps:

1. A shallow copy of the input items list is created using list(items). This is a critical step to ensure that the original list passed to the function is not modified. The new copy is stored in a local variable named copy.
2. The random.shuffle() method is then called on the copy. This function shuffles the elements of the list in-place, rearranging them into a random order.
3. Finally, the modified copy list, now containing the elements in a random sequence, is returned to the caller.

# Core logic of the function  
copy = list(items)  
random.shuffle(copy)  
return copy

## 5.4 Usage Notes

* The primary benefit of this function is that it is non-mutating. The original list you provide as an argument will remain in its original order after the function completes.
* This function relies on Python’s random module. You must ensure that the random module is imported in the environment where this function is used.
* While the type hint specifies List[int], the function’s logic will correctly handle lists containing elements of any data type (e.g., strings, floats, or mixed types).

**Output Example**: A new list containing the same elements as the input items list, but arranged in a random order. For an input of [1, 2, 3], a possible output could be [2, 3, 1].

## 5.5 Example

import random # This import is necessary for random.shuffle() to work  
  
# Example usage of the shuffle\_copy function  
original\_numbers = [10, 20, 30, 40, 50]  
shuffled\_numbers = shuffle\_copy(original\_numbers)  
  
print(f"Original List: {original\_numbers}")  
print(f"Shuffled Copy: {shuffled\_numbers}")  
  
# Another example with different data  
original\_items = ['apple', 'banana', 'cherry']  
shuffled\_items = shuffle\_copy(original\_items)  
print(f"\nOriginal Items: {original\_items}")  
print(f"Shuffled Items: {shuffled\_items}")

**Output:**

Original List: [10, 20, 30, 40, 50]  
Shuffled Copy: [30, 50, 10, 20, 40]  
  
Original Items: ['apple', 'banana', 'cherry']  
Shuffled Items: ['cherry', 'apple', 'banana']

*(Note: The actual order of elements in the shuffled output will be random and may differ with each execution.)*