Table of Contents

## 0.1 FunctionDef num(a, b)

# 1 Function: num

## 1.1 Overview

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

## 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 implementation of addition. It accepts two arguments, a and b, and computes their sum using the + operator. The resulting value is then returned by the function. The primary purpose of this function is to perform a basic arithmetic sum operation.

# The function adds the two parameters a and b  
return a + b

## 1.4 Usage Notes

* The function is designed to work with numeric types such as integers (int) and floating-point numbers (float).
* If non-numeric types that support the + operator (like strings) are passed, the function will perform the operation defined for that type (e.g., concatenation for strings) which may lead to unexpected results.

**Output Example**: A numeric value representing the sum. 15

## 1.5 Example

# Example usage with two integers  
result = num(5, 10)  
print(result)  
  
# Example usage with a float and an integer  
result\_float = num(7.5, 3)  
print(result\_float)

**Output:**

15  
10.5

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

# 2 Function: generate\_random\_integers

## 2.1 Overview

The generate\_random\_integers function creates and 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 total number of random integers to generate in the list. |
| start | int | The inclusive lower bound for the random number generation. Defaults to 0. |
| end | int | The inclusive upper bound for the random number generation. Defaults to 100. |

## 2.3 Description

This function provides a straightforward way to generate a list of pseudo-random integers. It operates based on three parameters: count, start, and end.

The function first performs two validation checks: 1. It ensures that the count parameter is not a negative number. If count is less than zero, it raises a ValueError to prevent invalid list creation. 2. It checks if the start value is greater than the end value. If this is the case, it automatically swaps the two values. This ensures that the range is always valid, regardless of the order in which the bounds are provided.

After validation, the function uses a list comprehension to generate the random numbers. It iterates count times, and in each iteration, it calls random.randint(start, end). The random.randint method returns a single integer that is uniformly sampled from the inclusive range [start, end]. The final result is a list containing count of these random integers.

## 2.4 Usage Notes

* The function requires the random module to be imported.
* A ValueError will be raised if the count argument is a negative integer.
* The function is robust to the order of start and end parameters; it will automatically correct the range if start > end.
* The range defined by start and end is inclusive, meaning both start and end values can appear in the output list.

**Output Example**: A call with count=5, start=1, and end=10 might produce a list similar to this:

[8, 2, 10, 5, 3]

## 2.5 Example

# Note: The 'random' module must be imported for this function to work.  
import random  
  
# Example 1: Generate 5 random integers between 1 and 50.  
random\_list = generate\_random\_integers(count=5, start=1, end=50)  
print(f"Generated list: {random\_list}")  
  
# Example 2: Using default start (0) and end (100) values.  
default\_list = generate\_random\_integers(count=3)  
print(f"Generated list with defaults: {default\_list}")  
  
# Example 3: Providing start and end in reverse order.  
# The function will automatically swap them to generate from 10 to 20.  
reversed\_order\_list = generate\_random\_integers(count=4, start=20, end=10)  
print(f"Generated list with reversed bounds: {reversed\_order\_list}")

**Output:**

# Note: The actual output will vary with each execution due to randomness.  
Generated list: [42, 15, 33, 9, 27]  
Generated list with defaults: [81, 12, 95]  
Generated list with reversed bounds: [11, 19, 14, 16]

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

# 3 Function: choose\_random\_item

## 3.1 Overview

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

## 3.2 parameters

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

## 3.3 Description

This function provides a simple and safe way to retrieve a random element from a list of strings. The function’s logic proceeds in two main steps:

1. **Validation**: It first checks if the provided items list is empty using the condition if not items:. This is a crucial safeguard to prevent runtime errors that would occur when trying to select an item from an empty collection. If the list is empty, the function raises a ValueError with the message “items must not be empty”.
2. **Selection**: If the list is not empty, the function proceeds to the selection step. It utilizes the random.choice(items) method from Python’s built-in random module. This method efficiently and uniformly selects a single item from the items list, ensuring that every item has an equal probability of being chosen. The selected string is then returned as the output.

# The function first checks if the list is empty.  
if not items:  
 # If so, it raises an error.  
 raise ValueError("items must not be empty")  
# Otherwise, it returns a random choice from the list.  
return random.choice(items)

## 3.4 Usage Notes

* The input list items must not be empty. Providing an empty list will result in a ValueError.
* This function relies on Python’s random module. Ensure it is imported and available in the environment where the function is called.
* The selection is uniformly random, meaning each string in the items list has an equal chance of being returned on any given call.

**Output Example**: A single string chosen from the input items list. For an input of ["apple", "banana", "cherry"], a possible output would be:

"banana"

## 3.5 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.  
  
 Parameters:  
 items: A list of strings to choose from.  
  
 Returns:  
 A single string chosen uniformly at random.  
 """  
 if not items:  
 raise ValueError("items must not be empty")  
 return random.choice(items)  
  
# Example usage:  
fruit\_options = ["apple", "banana", "cherry", "date"]  
random\_fruit = choose\_random\_item(fruit\_options)  
print(f"A random fruit was chosen: {random\_fruit}")  
  
# Example of error handling with an empty list:  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error when choosing from an empty list: {e}")

**Output:**

A random fruit was chosen: cherry  
Error when choosing from an empty list: items must not be empty

(Note: The actual fruit chosen will vary with each execution.)

## 3.6 FunctionDef shuffle\_copy(items)

# 4 Function: shuffle\_copy

## 4.1 Overview

The shuffle\_copy function returns a shuffled copy of a given list of integers without modifying the original list.

## 4.2 parameters

* items (List[int]): A list of integers to be shuffled. The original list provided as input will not be altered.

## 4.3 Description

The shuffle\_copy function provides a safe, non-mutating way to get a randomized version of a list while preserving the original data.

The function operates in three main steps: 1. It first creates a shallow copy of the input items list by calling copy = list(items). This is a critical step that ensures any subsequent modifications do not affect the original list that was passed to the function. 2. It then uses the random.shuffle() method on the newly created copy. The random.shuffle() function shuffles the elements of a sequence in-place, meaning it directly modifies the copy list to randomize the order of its elements. 3. Finally, the function returns this shuffled copy.

This process guarantees that the caller receives a new list with the same elements as the input but in a random order, while the original input list remains unchanged.

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

## 4.4 Usage Notes

* The primary feature of this function is that it is non-mutating. The original list passed as the items parameter will not be changed.
* This function depends on Python’s built-in random module. Ensure it is available in the execution environment for the function to work correctly.
* The returned list will contain the exact same elements as the input list, only in a different, random order. The length and contents of the list remain the same.

**Output Example**: A possible return value for an input of [1, 2, 3] could be:

[2, 3, 1]

## 4.5 Example

The following example demonstrates how to use shuffle\_copy and confirms that the original list is not modified.

# Example usage  
# Assume the function shuffle\_copy and the random module are imported  
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}")

**Output:**

Original list: [10, 20, 30, 40, 50]  
Shuffled copy: [30, 50, 10, 20, 40]

*(Note: The order of elements in the “Shuffled copy” will vary with each execution due to its random nature.)*