Table of Contents

## 0.1 FunctionDef num(a, b)

### 0.1.1 Overview

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

### 0.1.2 Parameters

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

### 0.1.3 Description

The num function provides a straightforward way to perform addition. It accepts two arguments, a and b, which are expected to be numerical types such as integers or floating-point numbers. The core logic of the function is to use the addition operator (+) to compute the sum of these two arguments. The resulting sum is then returned by the function.

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

def num(a,b):  
 return a+b

### 0.1.4 Usage Notes

* This function is designed for numeric types (int, float). While it may work with other types that support the + operator (like string concatenation), its primary purpose is arithmetic addition.
* The data type of the returned value depends on the input types. For instance, adding an int and a float will result in a float.

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

15

### 0.1.5 Example

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

#### 0.1.5.1 Output

15  
10.5

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

# 1 generate\_random\_integers

### 1.0.1 Overview

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

### 1.0.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| count | int | The total 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. |

### 1.0.3 Description

This function provides a straightforward way to generate a list of random integers. It relies on Python’s built-in random module.

The function first performs input validation. It 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.

Next, it ensures the range defined by start and end is valid. If the provided start value is greater than the end value, the function automatically swaps them. This allows users to specify the range boundaries without concern for their order.

Finally, it uses a list comprehension to generate the list of random integers. For each number from 0 to count - 1, it calls random.randint(start, end) to produce an integer that is uniformly sampled from the inclusive range [start, end]. The resulting integers are collected into a list and returned.

# Internal logic for swapping bounds  
if start > end:  
 start, end = end, start  
  
# List comprehension to generate numbers  
return [random.randint(start, end) for \_ in range(count)]

### 1.0.4 Usage Notes

* The function requires the random module to be imported to work correctly.
* A ValueError will be raised if the count argument is a negative integer.
* The range specified by start and end is inclusive, meaning both start and end can appear in the output list.
* If start is greater than end, their values are swapped internally, so generate\_random\_integers(5, 10, 1) is equivalent to generate\_random\_integers(5, 1, 10).

**Output Example**: The function returns a list of integers.

[12, 5, 88, 42, 99]

### 1.0.5 Example

import random  
from typing import List  
  
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 1 and 50.  
random\_numbers = generate\_random\_integers(5, 1, 50)  
print(random\_numbers)  
  
# Example with default parameters: Generate 3 random integers between 0 and 100.  
default\_random\_numbers = generate\_random\_integers(3)  
print(default\_random\_numbers)

#### 1.0.5.1 Output

# The actual output will vary with each execution due to randomness.  
[23, 4, 48, 11, 35]  
[76, 19, 82]

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

# 2 choose\_random\_item

### 2.0.1 Overview

The choose\_random\_item function chooses a single random item from a non-empty sequence of strings.

### 2.0.2 Parameters

| Parameter | Type | Description |
| --- | --- | --- |
| items | List[str] | A list of strings to choose from. This list must not be empty. |

### 2.0.3 Description

This function provides a simple and safe way to select one item at random from a given list of strings.

The function’s logic begins with a validation check. It first evaluates if the input items list is empty. If the list has no elements, the function raises a ValueError with the message “items must not be empty”. This preventative measure ensures that the underlying random.choice function, which cannot operate on an empty sequence, is not called with invalid input, thus avoiding a potential IndexError.

If the list is not empty, the function proceeds to call random.choice(items). This standard library function selects a single item from the sequence with a uniform probability distribution, meaning every item in the list has an equal chance of being chosen. The selected string is then returned as the result.

### 2.0.4 Usage Notes

* The input list items **must not be empty**. Providing an empty list will result in a ValueError.
* The function relies on Python’s built-in random module. Ensure this module is available in the execution environment.
* The selection is uniformly random, meaning each item in the list has an equal probability of being chosen on any given call.

**Output Example**: The function returns a single string that was present in the input items list. For an input of ['red', 'green', 'blue'], a possible return value is 'green'.

### 2.0.5 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.  
  
 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  
options = ["lion", "tiger", "bear", "wolf"]  
selected\_animal = choose\_random\_item(options)  
print(f"Selected animal: {selected\_animal}")  
  
# Example of error handling  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error: {e}")

#### 2.0.5.1 Output

Selected animal: tiger  
Error: items must not be empty

*(Note: The selected animal will vary with each execution as it is chosen randomly.)* ## FunctionDef shuffle\_copy(items) # shuffle\_copy:

### 2.0.6 Overview

The shuffle\_copy Function returns a shuffled copy of the given list without mutating the input.

### 2.0.7 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| items | List[int] | A list of integers to be shuffled. |

### 2.0.8 Description

The shuffle\_copy function is designed to create a randomized version of a list while ensuring the original list remains unchanged. This is achieved through a non-destructive shuffling process.

The function begins by creating a shallow copy of the input items list using the list() constructor. This step is crucial as it creates a new list in memory, independent of the original.

copy = list(items)

Next, it utilizes the random.shuffle() method to shuffle the elements of the newly created copy list. The random.shuffle() function performs the shuffle operation in-place, meaning it directly modifies the sequence it is given. Since it operates on the copy, the original items list is unaffected.

Finally, the function returns the shuffled copy.

### 2.0.9 Usage Notes

* This function guarantees that the original list passed as the items parameter will not be altered.
* The function depends on Python’s built-in random module. Ensure this module is imported in your project before calling shuffle\_copy.
* While the type hint specifies List[int], the function’s logic will work correctly with lists containing elements of any type (e.g., strings, floats, or mixed types).

**Output Example**: The function returns a new list with the same elements as the input list but in a random order. The exact order will differ on each execution.

[4, 1, 5, 2, 3]

### 2.0.10 Example

import random  
from typing import List  
  
# Definition of the function  
def shuffle\_copy(items: List[int]) -> List[int]:  
 """Return a shuffled copy of the given list without mutating the input."""  
 copy = list(items)  
 random.shuffle(copy)  
 return copy  
  
# Example usage  
original\_list = [1, 2, 3, 4, 5]  
shuffled\_list = shuffle\_copy(original\_list)  
  
print("Original List (unchanged):")  
print(original\_list)  
print("\nShuffled Copy:")  
print(shuffled\_list)

#### 2.0.10.1 Output

Original List (unchanged):  
[1, 2, 3, 4, 5]  
  
Shuffled Copy:  
[3, 5, 1, 4, 2]