Table of Contents

## 0.1 FunctionDef num(a, b)

# 1 Function: def num(a, b)

## 1.1 Overview

The num function calculates 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 provides a straightforward way to perform addition. It accepts two parameters, a and b. Internally, it uses the + operator to compute the sum of these two values. The resulting sum is then returned by the function. While designed for numeric types like integers and floats, it will also work with any data types that support the addition operator, such as strings for concatenation.

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

## 1.4 Usage Notes

* This function is compatible with both integers and floating-point numbers.
* If strings are passed as arguments, the function will perform string concatenation instead of mathematical addition.
* Ensure that both arguments are of compatible types for the + operator to avoid a TypeError.

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

## 1.5 Example

# Example usage with integers  
result = num(5, 10)  
print(result)  
  
# Example usage with floats  
result\_float = num(3.14, 2.71)  
print(result\_float)

**Output:**

15  
5.85

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

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

## 2.1 Overview

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

## 2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| count | int | The number of integers to generate. This value must be non-negative. |
| 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 robust way to generate a list of random integers. The logic proceeds as follows:

1. It first validates the count parameter. If count is a negative number, the function raises a ValueError because it’s not possible to generate a negative number of items.
2. Next, it ensures the range is valid. It checks if the start value is greater than the end value. If it is, the function automatically swaps them. This convenience feature ensures that random.randint receives a valid range (start <= end) without requiring the user to order the parameters correctly.
3. 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) to produce a single integer that is uniformly sampled from the inclusive range [start, end]. These integers are collected into a list.

The resulting list, containing exactly count random integers, is then returned.

# Internal logic for swapping start and end  
if start > end:  
 start, end = end, start  
# Internal logic for generation  
return [random.randint(start, end) for \_ in range(count)]

## 2.4 Usage Notes

* The function will raise a ValueError if the count argument is less than zero.
* If the provided start value is greater than the end value, the function will automatically swap them to form a valid range.
* The range defined by start and end is inclusive, meaning both start and end are possible values in the output.
* This function requires the random module to be imported in the file.

**Output Example**: The function returns a list of integers. The contents will be random and will differ on each execution.

[15, 8, 92, 43, 77]

## 2.5 Example

# Example: Generate 5 random integers between 10 and 20 (inclusive).  
# Note: The 'random' module must be imported for this function to work.  
import random  
  
result = generate\_random\_integers(count=5, start=10, end=20)  
print(result)

**Output:**

# The actual output will vary with each run.  
[12, 19, 10, 15, 17]

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

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

## 3.1 Overview

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

## 3.2 parameters

* items (List[str]): A list of strings from which to choose a random item. This list must not be empty.

## 3.3 Description

This function provides a safe way to select a random element from a list. It begins by validating the input items list.

The function first checks if the items list is empty using the condition if not items:. If the list is found to be empty, it raises a ValueError with the message “items must not be empty” to prevent runtime errors and enforce the requirement of a non-empty sequence.

If the list is not empty, the function proceeds to use random.choice(items). This standard library function takes a sequence as input and returns a single item chosen uniformly at random. The chosen string is then returned as the result of the choose\_random\_item function.

# Internal logic for choosing an item  
import random  
items = ["apple", "banana", "cherry"]  
# The function will execute this line if the list is not empty  
selected\_item = 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 depends on Python’s built-in random module. Ensure it is imported in the file where the function is defined.
* The selection is uniformly random, meaning every item in the list has an equal probability of being chosen on any given call.

**Output Example**: A single string from the input list. As the selection is random, the output will vary between calls. For an input of ['red', 'green', 'blue'], a possible output is 'green'.

## 3.5 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  
fruits = ["apple", "banana", "cherry", "date"]  
random\_fruit = choose\_random\_item(fruits)  
print(f"The randomly chosen fruit is: {random\_fruit}")  
  
# Example of what happens with an empty list  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

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

(Note: The actual fruit chosen in the first line of the output will vary with each execution.)

## 3.6 FunctionDef shuffle\_copy(items)

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

## 4.1 Overview

The shuffle\_copy function creates and returns a randomly shuffled copy of a list of integers, leaving the original list unchanged.

## 4.2 parameters

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

## 4.3 Description

This function provides a safe way to shuffle a list without altering the original data structure. The process is as follows:

1. A shallow copy of the input items list is created using the list() constructor. This new list is stored in a variable named copy. This step is crucial for ensuring that the original items list remains unmodified.
2. The random.shuffle() function is then called on the copy. This function shuffles the elements of the copy list in-place, rearranging them into a random order.
3. Finally, the function returns the modified copy, which now contains the same elements as the original list but in a new, random sequence.

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

## 4.4 Usage Notes

* This function is non-mutating, meaning it will not change the original list provided as the items parameter.
* The function relies on Python’s built-in random module. Ensure that import random is present at the top of the file where this function is used.
* While the type hint specifies List[int], the function’s logic will work correctly with lists containing other data types (e.g., strings, floats, or mixed types).

**Output Example**: The function returns a new list. For an input of [1, 2, 3, 4, 5], a possible return value could be:

[4, 1, 5, 3, 2]

## 4.5 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."""  
 copy = list(items)  
 random.shuffle(copy)  
 return copy  
  
# Example usage  
original\_list = [10, 20, 30, 40, 50]  
print(f"Original list (before): {original\_list}")  
  
shuffled\_list = shuffle\_copy(original\_list)  
  
print(f"Original list (after): {original\_list}")  
print(f"Returned shuffled list: {shuffled\_list}")

**Output:**

Original list (before): [10, 20, 30, 40, 50]  
Original list (after): [10, 20, 30, 40, 50]  
Returned shuffled list: [40, 10, 50, 20, 30]

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