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 values.

## 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, and computes their sum using the + operator. The result of this operation is then returned by the function. The primary use case is for numeric addition, but it will also work with any data types that support the addition operator, such as string concatenation.

# The function adds 'a' and 'b' and returns the result.  
return a + b

## 1.4 Usage Notes

* This function is designed for numeric types like integers (int) and floating-point numbers (float).
* If strings are passed as arguments, the function will perform string concatenation instead of mathematical addition. For example, num("hello", " world") would return "hello world".
* Passing incompatible types (e.g., an integer and a string) will result in a TypeError.

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

## 1.5 Example

# Example usage with integers  
result = num(5, 10)  
print(result)  
  
# Example usage with floating-point numbers  
float\_result = num(7.5, 2.2)  
print(float\_result)

**Output:**

15  
9.7

## 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 defined inclusive range.

## 2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| count | int | The number of random integers to generate. This value must be non-negative. |
| 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 convenient way to generate a list of pseudo-random integers. It takes three parameters: count, start, and end.

The function first validates the count parameter. If count is a negative number, it raises a ValueError because it’s not possible to generate a negative quantity of integers.

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

Finally, it uses a list comprehension to call the random.randint(start, end) function count times. Each call generates a single integer 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

* The function will raise a ValueError if the count argument is less than 0.
* If the start value is greater than the end value, the function will automatically swap them to form a valid range.
* The generated integers are within an inclusive range, meaning both the start and end values can be present in the output list.
* This function relies on Python’s built-in random module.

**Output Example**: A list of integers.

[45, 8, 92, 33, 67]

## 2.5 Example

# Example: Generate 5 random integers between 10 and 20 (inclusive).  
random\_numbers = generate\_random\_integers(5, 10, 20)  
print(random\_numbers)  
  
# Example: Generate 3 random integers with default range (0 to 100).  
random\_numbers\_default = generate\_random\_integers(3)  
print(random\_numbers\_default)  
  
# Example: The function automatically corrects the range if start > end.  
# This is equivalent to generate\_random\_integers(4, 50, 60).  
random\_numbers\_swapped = generate\_random\_integers(4, 60, 50)  
print(random\_numbers\_swapped)

**Output:**

# The output will be random, but will look similar to this:  
[18, 11, 20, 15, 10]  
[76, 2, 81]  
[55, 59, 51, 58]

## 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 random item from a given list of strings.

## 3.2 parameters

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

## 3.3 Description

This function provides a simple way to get a random element from a list of strings. The core logic relies on Python’s built-in random module.

First, the function performs a crucial validation check to ensure the input list items is not empty. If an empty list is passed (if not items), the function immediately raises a ValueError to prevent runtime errors and inform the developer of the invalid input.

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

# Internal logic for selecting an item  
import random  
items\_list = ["apple", "banana", "cherry"]  
# The function effectively does this:  
selected\_item = random.choice(items\_list)

## 3.4 Usage Notes

* The input list items must not be empty. Providing an empty list will result in a ValueError.
* The function depends on Python’s random module. Ensure it is available in the execution environment.
* The selection is uniformly random. Each item in the list has an equal probability of being returned.

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

'banana'

## 3.5 Example

# Example usage  
import random # Required for the function to work  
  
# Define a list of items  
fruit\_options = ["apple", "banana", "cherry", "date"]  
  
# Call the function to get a random fruit  
random\_fruit = choose\_random\_item(fruit\_options)  
  
print(f"The randomly chosen fruit is: {random\_fruit}")

**Output:**

# The output will vary with each execution. A possible output is:  
The randomly chosen fruit is: cherry

## 3.6 FunctionDef shuffle\_copy(items)

# 4 Function: shuffle\_copy

## 4.1 Overview

The shuffle\_copy Function returns a new, randomly shuffled copy of a given list, ensuring the original list remains unchanged.

## 4.2 parameters

* **items** (List[int]): A list of integers that will 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 involves three main steps:

1. A shallow copy of the input items list is created using the list() constructor. This new list is assigned to the copy variable. This step is critical to ensure that any subsequent modifications do not affect the original list passed to the function.
2. The random.shuffle() function 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 function returns the modified copy, which now contains the same elements as the original list but in a new, randomized sequence.

# Internal logic demonstration  
import random  
  
original\_items = [10, 20, 30, 40]  
  
# 1. Create a copy  
copy = list(original\_items)  
# copy is now [10, 20, 30, 40], but is a different object in memory  
  
# 2. Shuffle the copy in-place  
random.shuffle(copy)  
# copy might now be [30, 10, 40, 20]  
  
# 3. Return the shuffled copy  
# The function returns [30, 10, 40, 20]  
# The original\_items list remains [10, 20, 30, 40]

## 4.4 Usage Notes

* The primary benefit of this function is that it is non-mutating. The original list you pass as an argument will not be changed.
* This function depends on Python’s built-in random module. Ensure it is available in the environment.
* The returned list is a new list instance, separate from the input list.
* The shuffling is pseudo-random, meaning the sequence is generated by a deterministic algorithm but is practically unpredictable.

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

## 4.5 Example

import random  
  
# Assume shuffle\_copy is defined as in the provided code  
def shuffle\_copy(items: list) -> list:  
 copy = list(items)  
 random.shuffle(copy)  
 return copy  
  
# Example usage  
my\_numbers = [1, 2, 3, 4, 5, 6]  
shuffled\_numbers = shuffle\_copy(my\_numbers)  
  
print(f"Original List: {my\_numbers}")  
print(f"Shuffled Copy: {shuffled\_numbers}")  
  
# Another example  
source\_data = [100, 200, 300]  
randomized\_data = shuffle\_copy(source\_data)  
  
print(f"\nOriginal Data: {source\_data}")  
print(f"Randomized Data: {randomized\_data}")

**Output:**

Original List: [1, 2, 3, 4, 5, 6]  
Shuffled Copy: [4, 1, 6, 3, 5, 2]  
  
Original Data: [100, 200, 300]  
Randomized Data: [200, 300, 100]

*(Note: The actual output of the shuffled lists will vary with each execution due to the nature of random shuffling.)*