Table of Contents

## 1 FunctionDef num(a, b)

Function - num:

Purpose: Calculates and returns the sum of two input values.

Parameters: • a: The first value to be used in the addition. This can be any data type that supports the addition operator (+), such as an integer, float, or string. • b: The second value to be used in the addition. It should be of a type compatible with parameter ‘a’ for the addition operation to succeed.

Detailed Description: This function takes two arguments, a and b, and performs an addition operation on them using Python’s + operator. The function directly returns the result of this operation. The specific behavior of the addition depends on the data types of the input arguments; for numeric types, it performs arithmetic addition, and for sequence types like strings or lists, it performs concatenation.

Important Notes: • The function does not perform any type checking or validation on the input parameters. • If the provided arguments are of incompatible types for the + operator (e.g., an integer and a string), a TypeError will be raised at runtime. • The caller is responsible for ensuring that the two arguments are compatible for addition.

**Output Example**:

5

Usage Example:

# Example 1: Adding two integers  
sum\_integers = num(10, 5)  
print(sum\_integers)  
# Output: 15  
  
# Example 2: Adding two floating-point numbers  
sum\_floats = num(7.5, 2.5)  
print(sum\_floats)  
# Output: 10.0  
  
# Example 3: Concatenating two strings  
concatenated\_string = num("hello", " world")  
print(concatenated\_string)  
# Output: "hello world"

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

Function - generate\_random\_integers:

Purpose: Generates and returns a list of a specified number of pseudo-random integers within an inclusive range.

Parameters: • count (int): The total number of integers to generate and include in the output list. • start (int): The inclusive lower bound for the random number generation. The default value is 0. • end (int): The inclusive upper bound for the random number generation. The default value is 100.

Detailed Description: This function produces a list of pseudo-random integers based on the provided arguments. The process begins by validating the count parameter. If count is a negative number, the function immediately raises a ValueError to prevent an invalid operation.

Next, the function checks if the start value is greater than the end value. If this condition is true, it swaps the two values to ensure that start is the lower bound and end is the upper bound, creating a valid range for the subsequent random number generation.

Finally, the function uses a list comprehension to construct the output list. It iterates count times, and in each iteration, it calls random.randint(start, end) to generate a single integer that is uniformly sampled from the inclusive range [start, end]. These generated integers are collected into a list, which is then returned.

Important Notes: • This function requires the random module to be imported for the random.randint call to work. • A ValueError will be raised if the count argument is less than zero. • The function gracefully handles cases where the start parameter is greater than the end parameter by automatically swapping them. • The range for random number generation is inclusive, meaning both the start and end values can appear in the returned list.

**Output Example**: A possible return value for generate\_random\_integers(5, 1, 10):

[7, 2, 10, 5, 3]

Usage 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 1: Generate 5 random integers with default range (0 to 100)  
random\_list\_1 = generate\_random\_integers(5)  
print(f"Generated list with default range: {random\_list\_1}")  
# A possible output: Generated list with default range: [87, 23, 9, 75, 42]  
  
# Example 2: Generate 3 random integers in a custom range (10 to 20)  
random\_list\_2 = generate\_random\_integers(3, start=10, end=20)  
print(f"Generated list with custom range: {random\_list\_2}")  
# A possible output: Generated list with custom range: [15, 19, 11]  
  
# Example 3: Using a reversed range (function will swap them)  
random\_list\_3 = generate\_random\_integers(4, start=50, end=40)  
print(f"Generated list with reversed range: {random\_list\_3}")  
# A possible output: Generated list with reversed range: [48, 41, 45, 50]

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

Function - choose\_random\_item:

Purpose: Selects and returns a single string chosen uniformly at random from a non-empty list of strings.

Parameters: • items (List[str]): A list of strings from which a single item will be randomly selected. This list must not be empty.

Detailed Description: The function first performs a validation check to determine if the provided items list is empty. If the list contains no elements, the function raises a ValueError with the message “items must not be empty” to prevent errors in the subsequent random selection process.

If the list is not empty, the function utilizes the random.choice() method to select one element from the items list. The selection is uniform, ensuring that every item in the list has an equal probability of being chosen. The function then returns the selected string.

Important Notes: • This function depends on the choice function from Python’s built-in random module, which must be available in the environment. • The function will raise a ValueError if an empty list is passed as an argument. Callers should either ensure the list is non-empty or handle this potential exception using a try-except block. • While the type hint specifies List[str], the underlying random.choice() method will work with any non-empty sequence (like a tuple or list of other types).

Output Example: A single string element randomly chosen from the provided items list. For an input of ['apple', 'banana', 'cherry'], a possible return value is 'banana'.

Usage Example:

import random  
from typing import List  
  
# Assume the choose\_random\_item function is defined as provided.  
  
# Example 1: Choosing from a list of options  
fruits = ["apple", "banana", "cherry", "date"]  
random\_fruit = choose\_random\_item(fruits)  
print(f"The chosen fruit is: {random\_fruit}")  
# Possible Output: The chosen fruit is: cherry (output will vary on each run)  
  
# Example 2: Demonstrating the error with an empty list  
empty\_list = []  
try:  
 choice = choose\_random\_item(empty\_list)  
except ValueError as e:  
 print(f"An error occurred: {e}")  
# Output: An error occurred: items must not be empty

## 4 FunctionDef shuffle\_copy(items)

Function - shuffle\_copy:

Purpose: Returns a shuffled copy of the given list without mutating the input.

Parameters: • items (List[int]): A list of integers that will be copied and then shuffled.

Detailed Description: This function provides a non-destructive way to shuffle the elements of a list. The logical flow is as follows: 1. A new list, copy, is created as a shallow copy of the input items list. This is done using the list() constructor, which ensures that the original list passed to the function is not modified. 2. The random.shuffle() function is then called on this copy. This function shuffles the elements of the copy list in-place, rearranging them into a pseudo-random order. 3. Finally, the function returns the copy list, which now contains the same elements as the original items list but in a randomized sequence.

Important Notes: • This function requires the random module to be imported to use the random.shuffle() method. • A primary feature of this function is that it is non-mutating. The original list provided as the items argument will remain unchanged after the function call. • Although the type hint specifies List[int], the function will work correctly with a list containing any type of element (e.g., strings, floats, or objects) because random.shuffle operates on any mutable sequence.

**Output Example**: If the input items list is [1, 8, 3, 5, 2], a possible return value would be:

[5, 1, 8, 3, 2]

*(Note: The actual order of elements in the returned list will be random upon each execution.)*

Usage Example:

import random  
from typing import List  
  
# Assume the shuffle\_copy function is defined and available.  
  
# --- Example Usage ---  
original\_numbers = [10, 20, 30, 40, 50]  
print(f"Original list before call: {original\_numbers}")  
  
# Create a shuffled copy of the list  
shuffled\_numbers = shuffle\_copy(original\_numbers)  
  
print(f"Returned shuffled list: {shuffled\_numbers}")  
print(f"Original list after call: {original\_numbers}")  
  
# --- Expected Output ---  
# Original list before call: [10, 20, 30, 40, 50]  
# Returned shuffled list: [40, 10, 50, 20, 30] (Note: The order will be random)  
# Original list after call: [10, 20, 30, 40, 50]