Table of Contents

## 1 FunctionDef num(a, b)

**num**: This function calculates and returns the sum of two input values. **parameters**: The parameters of this Function. · a: The first operand for the addition operation. It can be a number (integer, float) or another data type that supports the + operator, such as a string or a list. · b: The second operand for the addition operation. It must be of a type that is compatible with the type of parameter a for the + operator. **Code Description**: The function num takes two arguments, a and b. It computes the result of a + b and returns this result. The behavior of the + operator is dependent on the data types of the input arguments. If both arguments are numbers, it performs arithmetic addition. If they are strings, it performs string concatenation. **Note**: Ensure that both arguments passed to the function are of types that can be used with the addition (+) operator. Attempting to add incompatible types, such as an integer and a string, will raise a TypeError. **Output Example**:

15

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

**generate\_random\_integers**: The function of generate\_random\_integers is to return a list of a specified number of pseudo-random integers within a given range.

**parameters**: · count: An integer specifying the number of random integers to generate. · start: An optional integer representing the inclusive lower bound of the random number range. The default value is 0. · end: An optional integer representing the inclusive upper bound of the random number range. The default value is 100.

**Code Description**: The function generate\_random\_integers is defined to accept three parameters: count, start, and end, and it is type-hinted to return a list of integers. Initially, the function validates the count parameter. If count is a negative number, it raises a ValueError with the message “count must be non-negative”, as it is impossible to generate a negative quantity of numbers. Next, it 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 always the lower bound and end is the upper bound of the range. This makes the function more flexible, as the user does not need to ensure the order of start and end. The core of the function is a list comprehension that iterates count times. In each iteration, it calls random.randint(start, end) to generate a single pseudo-random integer that is greater than or equal to start and less than or equal to end. These generated integers are collected into a list, which is the final return value of the function.

**Note**: This function requires the random module to be imported. The function handles cases where the start parameter is greater than the end parameter by automatically swapping them. Providing a negative value for the count parameter will result in a ValueError. The range defined by start and end is inclusive, meaning both start and end values can be present in the output list.

**Output Example**: A call to generate\_random\_integers(5, 10, 20) could produce the following list:

[12, 18, 10, 20, 15]

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

**choose\_random\_item**: The function of choose\_random\_item is to select and return a single random item from a given list of strings. **parameters**: · items: A non-empty list of strings from which to choose an item. **Code Description**: This function is designed to randomly pick one string from a list of strings. It accepts a single parameter, items, which must be a list of strings. The function first performs a validation check to ensure the provided list is not empty. If the items list is empty, it raises a ValueError with the message “items must not be empty” to prevent errors in the subsequent logic. If the list is valid and contains one or more elements, the function then utilizes the random.choice() method to select a single item from the list. The selection is made with uniform probability, meaning every item in the list has an equal chance of being chosen. The chosen string is then returned as the output of the function. **Note**: This function requires the standard Python random module to be imported. The input list items must not be empty; providing an empty list will result in a ValueError. **Output Example**: ‘banana’ ## FunctionDef shuffle\_copy(items) **shuffle\_copy**: The function of shuffle\_copy is to return a shuffled copy of a given list of integers without modifying the original list. **parameters**: · items: A list of integers (List[int]) that will be copied and shuffled. **Code Description**: The function begins by creating a shallow copy of the input list items using the list() constructor and assigning it to a new variable named copy. This operation ensures that the original list passed to the function is not altered. Next, it utilizes the random.shuffle() method to rearrange the elements of the copy list into a random order. The random.shuffle() function modifies the list in-place. Finally, the function returns the copy list, which now contains the same elements as the original items list but in a new, randomized sequence. **Note**: This function is non-mutating, meaning it does not alter the original list provided as an argument. It is a safe way to get a shuffled version of a list while preserving the original data. This behavior is distinct from random.shuffle(), which shuffles a list in-place and returns None. **Output Example**: Given the input items = [1, 2, 3, 4, 5], a possible return value would be:

[4, 1, 5, 3, 2]

The exact order of elements in the returned list will vary with each call to the function due to the random nature of the shuffling process.