Table of Contents

## 0.1 FunctionDef num(a, b)

# 1 Function: def num(a, b)

## 1.1 Overview

The num function calculates and returns 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 performs a basic addition operation. It takes two parameters, a and b, which are expected to be numeric types such as integers or floats. The function uses the standard addition operator (+) to compute their sum. The resulting value 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.

# The function adds the two input values and returns the result.  
return a + b

## 1.4 Usage Notes

* This function is designed for numeric types (int, float).
* If string values are passed as arguments, the function will perform string concatenation instead of mathematical addition (e.g., num("hello", " world") returns "hello world").
* Passing incompatible types (e.g., an integer and a string) will result in a TypeError.

**Output Example**: The function returns a single value of the same or a promoted numeric type (e.g., adding an int and a float results in a float).

## 1.5 Example

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

**Output:**

15  
10.5

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

# 2 Function: generate\_random\_integers(count: int, start: int = 0, end: int = 100) -> List[int]

## 2.1 Overview

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

## 2.2 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. |

## 2.3 Description

This function provides a straightforward way to generate a list of random integers. The process is as follows:

1. **Input Validation**: The function first validates the count parameter. If count is a negative number, it raises a ValueError because it’s impossible to generate a negative number of items.
2. **Range Correction**: It checks if the start value is greater than the end value. If it is, the function automatically swaps them. This ensures that random.randint receives a valid range (start <= end) and makes the function more robust to user input errors.
3. **Generation**: Using a list comprehension, the function iterates count times. In each iteration, it calls random.randint(start, end) to produce a single pseudo-random integer that is uniformly distributed within the inclusive range [start, end].
4. **Return Value**: All generated 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

* This function depends on Python’s built-in random module. Ensure it is imported before use.
* The count parameter must be a non-negative integer.
* Both the start and end boundaries are inclusive, meaning they can appear in the output list.
* If start is provided as a larger number than end, the function will automatically swap them and proceed without error.

**Output Example**: A call to generate\_random\_integers(5, 1, 10) might produce a list like this:

[3, 9, 1, 7, 5]

## 2.5 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 10 and 20 (inclusive).  
random\_numbers = generate\_random\_integers(5, 10, 20)  
print(random\_numbers)  
  
# Example with swapped start and end values  
swapped\_random\_numbers = generate\_random\_integers(3, 50, 40)  
print(swapped\_random\_numbers)

**Output:**

[15, 11, 20, 18, 12]  
[43, 48, 41]

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

## 2.6 FunctionDef fibonacci(n)

# 3 Function: fibonacci(n: int)

## 3.1 Overview

The fibonacci function computes the nth number in the Fibonacci sequence using an efficient iterative approach.

## 3.2 parameters

* n: int - The 0-indexed position in the Fibonacci sequence for which to find the corresponding number.

## 3.3 Description

This function calculates a Fibonacci number based on its index n. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1.

The function begins by validating the input n. If n is a negative number, it raises a ValueError, as the Fibonacci sequence is defined for non-negative integers.

It initializes two variables, a and b, to 0 and 1 respectively. These represent the first two numbers in the sequence (F₀ and F₁).

The core logic resides in a for loop that iterates n times. In each iteration, the values of a and b are updated simultaneously: a takes the current value of b, and b takes the sum of the old a and b. This process effectively steps through the sequence.

For example: - Start: a=0, b=1 - After 1st iteration: a=1, b=1 (0+1) - After 2nd iteration: a=1, b=2 (1+1) - After 3rd iteration: a=2, b=3 (1+2)

After the loop completes, the variable a holds the nth Fibonacci number, which is then returned. If n is 0, the loop does not run, and the initial value of a (0) is correctly returned.

# Inside the function  
if n < 0:  
 raise ValueError("n must be non-negative")  
a, b = 0, 1  
for \_ in range(n):  
 a, b = b, a + b  
return a

## 3.4 Usage Notes

* The input n must be a non-negative integer. Providing a negative integer will result in a ValueError.
* The function uses a 0-indexed sequence. For example, fibonacci(0) returns 0, fibonacci(1) returns 1, and so on.
* This iterative implementation is highly efficient in terms of memory and performance for large values of n compared to a naive recursive approach, as it avoids redundant calculations and deep recursion stacks.

**Output Example**: The function returns a single integer representing the Fibonacci number at the specified index.

## 3.5 Example

# Example usage  
# Find the 9th Fibonacci number (0-indexed)  
n\_index = 9  
result = fibonacci(n\_index)  
print(f"The Fibonacci number at index {n\_index} is: {result}")  
  
# Example with index 0  
result\_zero = fibonacci(0)  
print(f"The Fibonacci number at index 0 is: {result\_zero}")

**Output:**

The Fibonacci number at index 9 is: 34  
The Fibonacci number at index 0 is: 0

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

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

## 4.1 Overview

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

## 4.2 parameters

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

## 4.3 Description

This function provides a simple way to get a random item from a sequence. The core logic is implemented in two steps:

1. **Input Validation**: The function first checks if the provided items list is empty using the condition if not items:. If the list is empty, it raises a ValueError with the message “items must not be empty”. This is a crucial safeguard to prevent the underlying random.choice function from failing, as it requires a non-empty sequence.
2. **Random Selection**: If the list is not empty, the function uses random.choice(items) to perform the selection. random.choice() is a standard Python library function that takes a sequence as input and returns one of its elements chosen uniformly at random.

The selected string is then returned as the output of the function.

# Internal logic  
if not items:  
 raise ValueError("items must not be empty")  
return random.choice(items)

## 4.4 Usage Notes

* The input list items must contain at least one element. Providing an empty list will result in a ValueError.
* This function depends on Python’s built-in random module. Ensure that import random is present at the top of your script.
* Each item in the list has an equal probability of being selected.

**Output Example**: A single string from the input list. For an input of ["red", "green", "blue"], a possible return value is "green".

## 4.5 Example

import random  
from typing import List  
  
# The function being documented  
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  
options = ["option A", "option B", "option C", "option D"]  
selected\_option = choose\_random\_item(options)  
print(f"The selected option is: {selected\_option}")  
  
# Example of what happens with an empty list  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

The selected option is: option C  
Error: items must not be empty

*(Note: The first line of the output is random and will be one of the items from the options list each time the code is run.)*

## 4.6 FunctionDef shuffle\_copy(items)

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

## 5.1 Overview

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

## 5.2 parameters

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

## 5.3 Description

This function provides a non-destructive way to shuffle a list. The logic proceeds in three steps:

First, it creates a shallow copy of the input items list by calling list(items). This ensures that any subsequent modifications do not affect the original list that was passed to the function.

copy = list(items)

Next, it uses the random.shuffle() method to shuffle the elements of the newly created copy list in-place. This function rearranges the items in the list into a random order.

random.shuffle(copy)

Finally, the function returns the modified copy, which now contains the same elements as the original items list but in a new, randomized sequence.

## 5.4 Usage Notes

* The primary feature of this function is that it is non-mutating. The original list passed as the items parameter will remain in its original order after the function completes.
* This function depends on Python’s random module. Ensure that import random is present at the top of the script where this function is defined or called.
* 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) as list() and random.shuffle() are generic.

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

[4, 1, 5, 2, 3]

## 5.5 Example

import random  
from typing import List  
  
# Assuming the function is defined in the current scope  
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]  
shuffled\_list = shuffle\_copy(original\_list)  
  
print(f"Original List (unchanged): {original\_list}")  
print(f"Shuffled Copy: {shuffled\_list}")

**Output:**

Original List (unchanged): [10, 20, 30, 40, 50]  
Shuffled Copy: [30, 50, 10, 20, 40]

*(Note: The actual order of the shuffled list will vary with each execution due to its random nature.)*