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 input 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 basic arithmetic operation. It takes two parameters, a and b, and computes their sum using the standard addition operator (+). The resulting value is then returned by the function. The primary purpose is to encapsulate the addition of two numbers.

For example, if a is 10 and b is 5, the function will compute 10 + 5 and return the integer 15.

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

## 1.4 Usage Notes

* This function is intended for use with numeric types such as integers (int) and floating-point numbers (float).
* If string types are provided, the function will perform string concatenation rather than mathematical addition.
* Providing incompatible types (e.g., an integer and a None type) will result in a TypeError.

**Output Example**: A numeric value representing the sum, such as 15 or 12.5.

## 1.5 Example

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

**Output:**

15  
12.5

## 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 of a specified number of pseudo-random integers within a given 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 convenient way to generate a list of random integers. It operates in three main steps:

1. **Input Validation**: It first checks if the count parameter is a negative number. If it is, a ValueError is raised, as it’s impossible to generate a negative number of integers.
2. **Range Correction**: The function then compares the start and end parameters. If start is found to be greater than end, it automatically swaps their values. This ensures that random.randint receives a valid range (start <= end) and makes the function more robust to user input errors.
3. **Random Number Generation**: Finally, it uses a list comprehension to generate the list of integers. 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 which is then returned.

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

## 2.4 Usage Notes

* This function requires the random module to be imported.
* The count argument must be a non-negative integer. Providing a negative value will result in a ValueError.
* The range defined by start and end is inclusive, meaning both start and end can appear in the output list.
* If you provide a start value that is greater than the end value, the function will automatically swap them and proceed without error.

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

[3, 8, 1, 10, 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.  
random\_numbers = generate\_random\_integers(count=5, start=10, end=20)  
print(random\_numbers)

**Output:**

[15, 11, 20, 18, 10]

*(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 iterative approach.

## 3.2 parameters

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

## 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, starting from 0 and 1.

The function first validates the input n. If n is a negative number, it raises a ValueError because the Fibonacci sequence is not defined for negative indices.

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

The function then enters a for loop that iterates n times. In each iteration, it performs a simultaneous assignment: a, b = b, a + b. This updates a to the value of b (the next number in the sequence) and b to the sum of the previous a and b (the subsequent number).

After the loop completes, the variable a holds the nth Fibonacci number, which is then returned.

# For n = 4, the loop runs 4 times:  
# Initial: a = 0, b = 1  
# 1. a = 1, b = 0 + 1 = 1  
# 2. a = 1, b = 1 + 1 = 2  
# 3. a = 2, b = 1 + 2 = 3  
# 4. a = 3, b = 2 + 3 = 5  
# The loop finishes. The function returns a, which is 3.

## 3.4 Usage Notes

* The index n is 0-based. For example, fibonacci(0) returns 0, and fibonacci(1) returns 1.
* The function will raise a ValueError if a negative integer is provided for the n parameter.
* This iterative implementation is efficient in terms of memory and performance compared to a naive recursive solution, as it avoids redundant calculations and deep recursion stacks.

**Output Example**: The function returns a single integer value.

55

## 3.5 Example

# Example usage to find the 10th Fibonacci number  
try:  
 result = fibonacci(10)  
 print(f"The 10th Fibonacci number is: {result}")  
  
 # Example with index 0  
 result\_zero = fibonacci(0)  
 print(f"The 0th Fibonacci number is: {result\_zero}")  
  
 # Example that would raise an error  
 # fibonacci(-5)  
  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

The 10th Fibonacci number is: 55  
The 0th Fibonacci number is: 0

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

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

## 4.1 Overview

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

The function first performs a validation check to ensure the input list items is not empty. If the list is empty, it raises a ValueError to prevent the underlying random.choice() function from failing, as random.choice() cannot operate on an empty sequence.

If the list contains one or more items, the function utilizes random.choice(items). This standard library function takes the list as input and returns one of its elements. The selection is uniformly random, meaning every item in the list has an equal probability of being chosen. The selected string is then returned as the result.

# Internally, the function works like this:  
import random  
  
def choose\_random\_item(items: List[str]) -> str:  
 # 1. Check if the list is empty  
 if not items:  
 # 2. Raise an error if it is  
 raise ValueError("items must not be empty")  
 # 3. If not empty, use random.choice to pick and return an item  
 return random.choice(items)

## 4.4 Usage Notes

* The input list items must not be empty. Providing an empty list will result in a ValueError.
* The selection is uniformly random. Each string in the list has an equal chance of being returned.
* The function is type-hinted to accept a list of strings (List[str]) and return a single string (str).

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

## 4.5 Example

# Assumes the 'random' module is imported in the file where choose\_random\_item is defined.  
# Example usage:  
import random  
from typing import List  
  
# Definition of the function  
def choose\_random\_item(items: List[str]) -> str:  
 if not items:  
 raise ValueError("items must not be empty")  
 return random.choice(items)  
  
# A list of options  
options = ["apple", "banana", "cherry", "date"]  
  
# Call the function with the list  
selected\_fruit = choose\_random\_item(options)  
  
# Print the result  
print(f"The randomly selected fruit is: {selected\_fruit}")  
  
# Example of what happens with an empty list  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

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

*(Note: The selected fruit will vary with each execution as it is chosen randomly.)*

## 4.6 FunctionDef shuffle\_copy(items)

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

## 5.1 Overview

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

## 5.2 parameters

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

## 5.3 Description

This function provides a safe way to shuffle a list without altering the original data structure. The process is executed in three main steps:

1. A shallow copy of the input items list is created using the list() constructor. This step is crucial for preserving the original list, as any subsequent modifications will only affect the new copy.
2. The random.shuffle() method is then called on this 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 holds the same elements as the original list but in a new, randomized sequence.

# Internal logic of the function  
import random  
  
def shuffle\_copy(items: List[int]) -> List[int]:  
 # 1. Create a shallow copy  
 copy = list(items)  
 # 2. Shuffle the copy in-place  
 random.shuffle(copy)  
 # 3. Return the shuffled copy  
 return copy

## 5.4 Usage Notes

* This function is designed to be non-mutating. The input list items will not be altered by the function call.
* The function depends on Python’s built-in random module. Ensure it is imported in your project before calling this function.
* The function creates a shallow copy. For a list of integers, this is sufficient. If the list were to contain mutable objects (like other lists or dictionaries), the inner objects themselves would not be duplicated.

**Output Example**: The function returns a list of integers. The order of elements is random and will likely differ with each execution. For an input of [1, 2, 3, 4, 5], a possible output could be:

[4, 1, 5, 3, 2]

## 5.5 Example

import random  
from typing import List  
  
# Definition of the function  
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("Original List (unchanged):")  
print(original\_list)  
print("\nShuffled Copy:")  
print(shuffled\_list)

**Output:**

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