Table of Contents

## 0.1 FunctionDef num(a, b)

# 1 Function: def num(a, b)

## 1.1 Overview

The num function computes 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 simple mechanism for addition. It accepts two arguments, a and b, and utilizes the standard Python addition operator (+) to calculate their sum. The function then returns the resulting value. Although primarily intended for numeric types such as integers and floats, it will also function with any data types that overload the + operator, like strings for concatenation.

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

## 1.4 Usage Notes

* The function can handle both integers and floating-point numbers. The return type will be a float if at least one of the inputs is a float.
* If string values are passed for both a and b, the function will perform string concatenation.
* Attempting to add incompatible types (e.g., an int and a str) will raise a TypeError.

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

8

## 1.5 Example

# Example usage with two integers  
result = num(5, 3)  
print(result)  
  
# Example usage with a float and an integer  
float\_result = num(7.5, 2)  
print(float\_result)

**Output:**

8  
9.5

## 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 creates and 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. The default value is 0. |
| end | int | The inclusive upper bound for the random values. The default value is 100. |

## 2.3 Description

This function provides a convenient way to generate a list of pseudo-random integers. The core logic is built around Python’s random.randint function, which is called repeatedly within a list comprehension.

The function first performs two validation checks: 1. It ensures the count parameter is not negative. If count is less than zero, it raises a ValueError because it’s impossible to generate a negative number of items. 2. It checks if the start value is greater than the end value. If it is, the function automatically swaps them to ensure a valid range. This makes the function more robust and forgiving of user input errors.

After validation, it uses a list comprehension to construct the list. The loop runs count times, and in each iteration, random.randint(start, end) is called to produce an integer that is greater than or equal to start and less than or equal to end. The resulting 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 requires the random module to be imported.
* The count parameter 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 are possible values in the output list.
* If start is greater than end, the function will automatically swap the values to form a correct range, so generate\_random\_integers(5, 10, 1) is equivalent to generate\_random\_integers(5, 1, 10).

**Output Example**: A list of integers. The contents will be random and will vary with each call.

[42, 8, 99, 23, 76]

## 2.5 Example

import random  
from typing import List  
  
# Example usage: Generate 5 random integers between 1 and 10.  
result = generate\_random\_integers(count=5, start=1, end=10)  
print(result)  
  
# Example using default start and end values (0 to 100)  
result\_default = generate\_random\_integers(count=3)  
print(result\_default)

**Output:**

# Note: The actual output will vary due to the random nature of the function.  
[3, 9, 1, 7, 5]  
[87, 22, 45]

## 2.6 FunctionDef fibonacci(n)

# 3 Function: fibonacci(n: int)

## 3.1 Overview

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

## 3.2 parameters

* n (int): The 0-indexed index of the Fibonacci number to compute.

## 3.3 Description

This function calculates a number in the Fibonacci sequence based on its index n. The Fibonacci sequence starts with 0 and 1, and each subsequent number is the sum of the two preceding ones (0, 1, 1, 2, 3, 5, 8, …).

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. The function then enters a for loop that iterates n times. In each iteration, the values of a and b are updated to advance to the next number in the sequence: a takes the value of b, and b becomes the sum of the old a and b.

# Core iterative logic  
a, b = 0, 1  
for \_ in range(n):  
 a, b = b, a + b

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

## 3.4 Usage Notes

* The input n must be a non-negative integer. Providing a negative value will result in a ValueError.
* The function is 0-indexed. For example, fibonacci(0) returns the first number (0), and fibonacci(1) returns the second number (1).
* This iterative implementation is memory-efficient and avoids the recursion depth limits that can occur with a naive recursive approach, especially for large values of n.

**Output Example**: A single integer representing the calculated Fibonacci number.

13

## 3.5 Example

# Example usage  
# Calculate the 7th Fibonacci number (0-indexed)  
result = fibonacci(7)  
print(result)

**Output:**

13

## 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 random element from a given list of strings.

## 4.2 parameters

* items: List[str] - A list of strings from which one item will be chosen at random. This list must not be empty.

## 4.3 Description

This function provides a safe way to select a random item from a list. It begins by performing a validation check on the input items list. If the list is empty (if not items:), the function immediately raises a ValueError to prevent runtime errors that would occur from attempting to choose from an empty sequence.

If the list is not empty, the function utilizes the random.choice() method from Python’s standard random library. The random.choice(items) call takes the list as an argument and returns one of its elements, with each element having an equal probability of being selected. The selected string is then returned as the final output.

## 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 relies on Python’s random module, which must be imported in the file where this function is defined.
* The selection is uniformly random, meaning every item in the list has an equal chance of being chosen.

**Output Example**: If the input list is ['red', 'green', 'blue'], a possible return value would be the string 'green'.

## 4.5 Example

import random  
from typing import List  
  
# Definition of the function  
def choose\_random\_item(items: List[str]) -> str:  
 """Choose a single random item from a non-empty sequence.  
  
 Parameters:  
 items: A list of strings to choose from.  
  
 Returns:  
 A single string chosen uniformly at random.  
 """  
 if not items:  
 raise ValueError("items must not be empty")  
 return random.choice(items)  
  
# Example usage  
options = ["apple", "banana", "cherry", "date"]  
selected\_fruit = choose\_random\_item(options)  
print(f"The randomly selected fruit is: {selected\_fruit}")  
  
# Example of error handling  
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

## 4.6 FunctionDef shuffle\_copy(items)

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

## 5.1 Overview

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

## 5.2 parameters

* items (List[int]): A list of integers that you want to shuffle.

## 5.3 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 copy = list(items). This is a critical step that prevents mutation of the original list passed to the function.
2. The random.shuffle() method is then called on this new copy. The random.shuffle() function shuffles the elements of a sequence in-place. By applying it to the copy, we modify only the copied list.
3. Finally, the function returns the copy, which now contains all the original elements but in a random order.

# Internal logic breakdown  
import random  
  
original\_list = [1, 2, 3, 4]  
  
# 1. Create a shallow copy  
copied\_list = list(original\_list)   
  
# 2. Shuffle the copy in-place  
random.shuffle(copied\_list)  
  
# 3. The copied\_list is now shuffled, original\_list is not  
# copied\_list might be [3, 1, 4, 2]  
# original\_list is still [1, 2, 3, 4]  
  
# 4. Return the shuffled copy  
# return copied\_list

## 5.4 Usage Notes

* This function is non-mutating, meaning it will not change the original items list provided as an argument.
* The function depends on Python’s built-in random module. Ensure this module is imported in the file where shuffle\_copy is defined.
* While the type hint specifies List[int], the function will work correctly with lists containing other data types (e.g., strings, floats, or mixed types).
* The returned list will always have the same length and elements as the input list.

**Output Example**: The function returns a list of the same elements in a new, random order. For an input of [10, 20, 30, 40], a possible output could be:

[30, 10, 40, 20]

## 5.5 Example

The following example demonstrates how to use shuffle\_copy and confirms that the original list is not modified.

import random  
from typing import List  
  
# Definition of the function  
def shuffle\_copy(items: List[int]) -> List[int]:  
 copy = list(items)  
 random.shuffle(copy)  
 return copy  
  
# Example usage  
original\_numbers = [1, 2, 3, 4, 5]  
shuffled\_numbers = shuffle\_copy(original\_numbers)  
  
print("Original List:", original\_numbers)  
print("Shuffled Copy:", shuffled\_numbers)

**Output:**

Original List: [1, 2, 3, 4, 5]  
Shuffled Copy: [3, 5, 1, 2, 4]

*(Note: The order of elements in “Shuffled Copy” will vary with each execution due to its random nature.)*