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 provides a straightforward way to perform addition. It accepts two parameters, a and b, which are expected to be numerical values (integers or floats). The core logic uses the standard Python addition operator (+) to compute the sum of a and b. 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 simply returns the result of a + b  
return a + b

## 1.4 Usage Notes

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

**Output Example**: A single numerical value representing the sum.

## 1.5 Example

# Example usage with two integers  
result = num(5, 10)  
print(result)  
  
# Example usage with two floats  
result\_float = num(3.14, 2.71)  
print(result\_float)

**Output:**

15  
5.85

## 1.6 FunctionDef generate\_random\_integers

# 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. 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. It operates in three main steps:

1. **Input Validation**: It first checks if the count parameter is a non-negative number. If count is less than zero, it raises a ValueError to prevent invalid list creation.
2. **Range Correction**: The function ensures that the range defined by start and end is valid. If the provided start value is greater than the end value, it automatically swaps them. This allows the user to specify the range boundaries in any order without causing an error.
3. **Random Number Generation**: Using a list comprehension, the function iterates count times. 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 integers are collected into a list which is then returned.

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

## 2.4 Usage Notes

* The function requires the random module to be imported to work correctly.
* A ValueError will be raised if the count argument is a negative integer.
* If start is greater than end, their values will be swapped internally, so generate\_random\_integers(5, 10, 1) is equivalent to generate\_random\_integers(5, 1, 10).
* The start and end parameters are optional and have default values of 0 and 100 respectively.

**Output Example**: The function returns a list of integers.

[42, 8, 99, 23, 67]

## 2.5 Example

# Example: Generate 5 random integers between 10 and 20 (inclusive).  
# Note: The 'random' module must be imported first.  
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)]  
  
result = generate\_random\_integers(5, 10, 20)  
print(result)

**Output:**

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

[15, 11, 20, 18, 10]

## 2.6 FunctionDef fibonacci

# 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 index of the Fibonacci number to be computed.

## 3.3 Description

This function calculates a number in the Fibonacci sequence, which starts with 0 and 1, and each subsequent number is the sum of the two preceding ones (e.g., 0, 1, 1, 2, 3, 5, …).

The function begins by validating 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, 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 using tuple assignment: a, b = b, a + b. This effectively shifts the sequence forward: 1. The current value of b becomes the new a. 2. The sum of the old a and b becomes the new b.

After the loop completes n iterations, the variable a will hold the nth Fibonacci number, which is then returned. For n=0, the loop does not run, and the initial value of a (0) is correctly returned.

# For n = 3:  
# Initial state: a = 0, b = 1  
# Iteration 1: a becomes 1, b becomes 0 + 1 = 1  
# Iteration 2: a becomes 1, b becomes 1 + 1 = 2  
# Iteration 3: a becomes 2, b becomes 1 + 2 = 3  
# Loop ends. The function returns a, which is 2.

## 3.4 Usage Notes

* The input n must be a non-negative integer. The function will raise a ValueError for negative inputs.
* The function uses a 0-indexed sequence, so fibonacci(0) returns 0, fibonacci(1) returns 1, and so on.
* This iterative implementation is memory-efficient and avoids the recursion depth limits and performance issues associated with naive recursive solutions for large n.

**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)  
result = fibonacci(9)  
print(result)

**Output:**

34

## 3.6 FunctionDef choose\_random\_item

# 4 Function: choose\_random\_item(items: List[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 non-empty list of strings from which to choose a random item.

## 4.3 Description

This function provides a safe way to select a random element from a list. It begins by performing a validation check on the input items. If the provided 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 proceeds to use the random.choice() method from Python’s standard random library. This method takes the items list as an argument and returns one of its elements, with each element having an equal probability of being selected.

# Internal logic for a non-empty list  
return random.choice(items)

The function ensures that a valid string item is always returned, provided the input list is not empty.

## 4.4 Usage Notes

* This function requires the input items to be a list of strings.
* A ValueError will be raised if an empty list is passed as an argument. Ensure you handle this exception or validate the list’s contents before calling the function.
* The selection is uniformly random, meaning every item in the list has an equal chance of being chosen on any given call.
* The function depends on Python’s built-in random module.

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

## 4.5 Example

# Example usage  
# Note: The 'random' module must be available in the environment.  
import random  
from typing import List  
  
# Definition of the function as provided  
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 possible choices  
color\_options = ["red", "green", "blue", "yellow", "purple"]  
  
# Call the function to get a random color  
selected\_color = choose\_random\_item(color\_options)  
  
print(f"The randomly selected color is: {selected\_color}")  
  
# Example of error handling for an empty list  
try:  
 empty\_list = []  
 choose\_random\_item(empty\_list)  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

The randomly selected color is: blue  
Error: items must not be empty

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

## 4.6 FunctionDef shuffle\_copy

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

## 5.1 Overview

The shuffle\_copy function returns a new, randomly shuffled copy of a given list of integers without modifying the original list.

## 5.2 parameters

* items: List[int] - The list of integers that will be copied and shuffled.

## 5.3 Description

This function provides a safe and non-destructive way to shuffle the elements of a list. The core logic operates in three steps:

1. A new list named copy is created as a shallow copy of the input items list. This is achieved with the expression list(items). This step is critical to ensure that the original list passed to the function remains unchanged.
2. The random.shuffle(copy) function is called. This function, part of Python’s standard random module, shuffles the elements of the copy list *in-place*. It rearranges the items into a random permutation.
3. The function returns the copy list, which now contains the same elements as the original items list but in a new, random order.

# The function creates a copy to avoid changing the original list.  
original = [1, 2, 3]  
shuffled = shuffle\_copy(original)   
# `original` is still [1, 2, 3]  
# `shuffled` might be [2, 3, 1]

## 5.4 Usage Notes

* This function is non-mutating. The original list passed as the items argument will not be altered.
* The function requires the random module to be imported to use random.shuffle.
* The order of the elements in the returned list is non-deterministic and will likely be different each time the function is called.

**Output Example**: A possible return value for an input of [1, 2, 3, 4, 5] could be:

[3, 1, 5, 2, 4]

## 5.5 Example

import random  
from typing import List  
  
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  
my\_numbers = [10, 20, 30, 40, 50, 60]  
shuffled\_numbers = shuffle\_copy(my\_numbers)  
  
print(f"Original List: {my\_numbers}")  
print(f"Shuffled Copy: {shuffled\_numbers}")  
  
# Verify the original list is unchanged  
print(f"Original list is still the same: {my\_numbers == [10, 20, 30, 40, 50, 60]}")

**Output:**

Original List: [10, 20, 30, 40, 50, 60]  
Shuffled Copy: [40, 10, 60, 20, 50, 30]  
Original list is still the same: True

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