Table of Contents

## 0.1 FunctionDef choose\_random\_item

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

## 1.1 Overview

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

## 1.2 parameters

* **items** List[str]: A non-empty list of strings from which a single item will be randomly selected.

## 1.3 Description

This function provides a simple way to choose one element uniformly at random from a sequence.

The function first performs a validation check to ensure the input list items is not empty. If the list is empty, it immediately raises a ValueError with the message “items must not be empty” to prevent runtime errors and enforce the requirement of a non-empty sequence.

If the list contains one or more items, the function then utilizes the random.choice() method. This method takes the items list as an argument and returns a single element chosen from it. Each item in the list has an equal probability of being selected. The selected string is then returned as the output of the function.

# Internal logic for a non-empty list  
import random  
my\_list = ["apple", "banana", "cherry"]  
# The following line is equivalent to the function's core operation  
selected = random.choice(my\_list)

## 1.4 Usage Notes

* This function requires the random module to be imported in the script.
* The input list items must not be empty. Providing an empty list will result in a ValueError.
* The selection is uniformly random, meaning every item in the list has an equal chance of being chosen on any given call.

**Output Example**: The function returns a single string. For an input of ['red', 'green', 'blue'], a possible return value would be:

"green"

## 1.5 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."""  
 if not items:  
 raise ValueError("items must not be empty")  
 return random.choice(items)  
  
# Example usage  
fruit\_options = ["apple", "banana", "cherry", "date"]  
chosen\_fruit = choose\_random\_item(fruit\_options)  
print(f"The randomly chosen fruit is: {chosen\_fruit}")  
  
# Example of error handling  
try:  
 choose\_random\_item([])  
except ValueError as e:  
 print(f"Error: {e}")

**Output:**

The randomly chosen fruit is: cherry  
Error: items must not be empty

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

### 1.5.1 FunctionDef choose\_random\_item

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

## 2.1 Overview

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

## 2.2 parameters

* **items** (List[str]): A list of strings from which one item will be randomly chosen.

## 2.3 Description

This function provides a straightforward way to get a random element from a list. It takes a single argument, items, which is expected to be a list of strings.

Internally, the function utilizes Python’s built-in random module. It calls the random.choice() method, passing the input items list to it. The random.choice() method is specifically designed to return a randomly selected element from a non-empty sequence. The element chosen by random.choice() is then returned as the output of the choose\_random\_item function.

import random  
from typing import List  
  
def choose\_random\_item(items: List[str]) -> str:  
 """  
 Selects a random item from a list of strings.  
 """  
 return random.choice(items)

## 2.4 Usage Notes

* The input list items must not be empty. Providing an empty list will cause a IndexError to be raised by the underlying random.choice() method.
* While the type hint specifies List[str], the core random.choice() function can operate on any non-empty sequence (like a tuple or a list of other types). However, for intended use according to the function’s signature, a list of strings should be provided.

## 2.5 Example

import random  
from typing import List  
  
def choose\_random\_item(items: List[str]) -> str:  
 """  
 Selects a random item from a list of strings.  
 An IndexError will be raised if the list is 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 randomly selected option is: {selected\_option}")

**Output:**

The randomly selected option is: Option C

*(Note: The actual output is non-deterministic and will be one of the elements from the options list each time the code is run.)*

## 2.6 FunctionDef shuffle\_copy

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

## 3.1 Overview

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

## 3.2 parameters

* **items** List[int]: A list of integers to be shuffled. The original list will not be modified.

## 3.3 Description

This function provides a safe way to shuffle a list without altering the original data structure. The process is straightforward and consists of two main steps.

First, the function creates a shallow copy of the input items list by calling list(items). This is a critical step that prevents mutation of the original list passed to the function. The new list is stored in a local variable named copy.

Next, it utilizes the random.shuffle() method, which shuffles the elements of the copy list in-place. Because this operation is performed on the copy and not the original items list, the integrity of the input data is preserved.

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

# Step 1: Create a copy  
original = [1, 2, 3]  
copy = list(original) # copy is now [1, 2, 3], but a separate object  
  
# Step 2: Shuffle the copy in-place  
# random.shuffle(copy) might change copy to [3, 1, 2]  
  
# Step 3: Return the shuffled copy  
# The function returns [3, 1, 2] while original remains [1, 2, 3]

## 3.4 Usage Notes

* The primary advantage of this function is that it is non-mutating. The input list items is guaranteed to remain in its original order after the function call.
* This function depends on Python’s built-in random module. Ensure this module is imported (e.g., import random) in the scope where shuffle\_copy is defined and used.
* While the type hint specifies List[int], the function’s logic will work correctly with lists containing any type of element (e.g., strings, floats, or mixed types).

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

[4, 1, 5, 3, 2]

## 3.5 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\_numbers = [10, 20, 30, 40, 50]  
shuffled\_numbers = shuffle\_copy(original\_numbers)  
  
print(f"Original List: {original\_numbers}")  
print(f"Shuffled Copy: {shuffled\_numbers}")

**Output:**

Original List: [10, 20, 30, 40, 50]  
Shuffled Copy: [30, 50, 10, 40, 20] # Note: The order will be random on each execution.

### 3.5.1 FunctionDef shuffle\_copy

# 4 Function: shuffle\_copy(items: List[int]) -> List[int]

## 4.1 Overview

The shuffle\_copy function creates and returns a new list containing the same elements as the input list but in a random order.

## 4.2 parameters

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

## 4.3 Description

This function provides a non-destructive way to shuffle a list. The core logic involves two main steps:

1. **Create a Copy**: The function first creates a shallow copy of the input items list. This is a critical step to ensure that the original list passed to the function remains unchanged. This is typically achieved using a method like items.copy().
2. **Shuffle the Copy**: The newly created list is then shuffled in-place using a randomization algorithm, such as the Fisher-Yates shuffle. This reorders the elements within the copy into a random permutation.

Finally, the function returns the shuffled copy. The original items list is not modified by this operation.

## 4.4 Usage Notes

* This function is non-destructive; it does not alter the original input list items.
* A new list object is created and returned with each call.
* The order of elements in the returned list is random. Calling the function multiple times with the same input will likely produce different results.

## 4.5 Example

import random  
from typing import List  
  
# A possible implementation for demonstration  
def shuffle\_copy(items: List[int]) -> List[int]:  
 items\_copy = items.copy()  
 random.shuffle(items\_copy)  
 return items\_copy  
  
# Example usage  
original\_numbers = [1, 2, 3, 4, 5, 6]  
shuffled\_numbers = shuffle\_copy(original\_numbers)  
  
print(f"Original List: {original\_numbers}")  
print(f"Shuffled Copy: {shuffled\_numbers}")

**Output:**

Original List: [1, 2, 3, 4, 5, 6]  
Shuffled Copy: [4, 1, 6, 3, 5, 2] # Note: The actual order will be random and may vary.