

# 1. WHAT IS LIST? HOW WILL YOU REVERSE A LIST?

- ✚ A list in Python is a versatile data structure that allows you to store multiple items in a single variable.
- ✚ Lists are ordered, changeable (mutable), and allow duplicate values.
- ✚ They are defined by placing elements inside square brackets [], separated by commas.
- ✚ Example of a list:

```
my_list = [1, 2, 3, 'apple', 'banana', True]
```

## ✚ Key Features of Lists:

- **Ordered:** The items have a defined order, and that order will not change unless explicitly modified.
- **Changeable:** You can add, remove, or change items after the list has been created.
- **Allow Duplicates:** Lists can contain multiple items with the same value.
- **Indexed:** Each item in a list has an index, starting from 0 for the first item.

## ❖ Reverse a List :

- ✚ There are several ways to reverse a list in Python.

## ❖ Using the reverse() Method:

- ✚ This method reverses the list in place, meaning it modifies the original list.

```
my_list = [1, 2, 3, 4, 5]  
my_list.reverse()  
print(my_list)
```

***Output: [5, 4, 3, 2, 1]***

#### ❖ Using List Slicing:

- ✚ This method creates a new list that is the reverse of the original list.

```
my_list = [1, 2, 3, 4, 5]  
reversed_list = my_list[::-1]  
print(reversed_list)
```

***# Output: [5, 4, 3, 2, 1]***

**2.HOW WILL YOU REMOVE LAST OBJECT FROM A LIST? SUPPOSE LIST1 IS [2, 33, 222, 14, AND 25], WHAT IS LIST1 [-1]?**

### ❖ Removing the Last Object from a List :

✚ To remove the last object from a list in Python, you can use the pop() method without any arguments. This method removes and returns the last item in the list.

Example:

```
list1 = [2, 33, 222, 14, 25]  
last_item = list1.pop()  
print(last_item) # Output: 25  
print(list1)
```

*# Output: [2, 33, 222, 14]*

### ❖ Accessing the Last Object in a List :

✚ you can access the last object in a list using negative indexing. The index -1 refers to the last item in the list.

Example :

```
list1 = [2, 33, 222, 14, 25]  
print(list1[-1])
```

*# Output: 25*

**3.DIFFERENTIATE BETWEEN APPEND ( ) AND  
EXTEND ( ) METHODS?**

Basis of Comparison	append()	extend()
<b>Syntax</b>	list_name.append(element)	list_name.extend(iterable)
<b>Input</b>	Accepts only one input element	Accepts input as an iterable (e.g., list, tuple)
<b>Functionality</b>	Adds a single element to the end of the existing list	Appends several items to a list as individual items at the end of the list
<b>Operation</b>	Adds the full input to the list as a single item	Adds each item to the list separately after iterating through each one in the input
<b>Efficiency</b>	Typically quicker and more effective since it only executes one operation at a time	Can take longer when adding elements from different iterables or with large inputs
<b>Time Complexity</b>	Constant time complexity, (O(1))	Time complexity of (O(K)), where (K) is the length of the list to be added

## 4. HOW WILL YOU COMPARE TWO LISTS?

✚ There are several ways to compare two lists in Python.

✚ Here are a few common methods:

### 1. Check if two lists are equal:

```
list1 = [1, 2, 3]
```

```
list2 = [1, 2, 3]
```

```
if list1 == list2:
```

```
    print("The lists are equal.")
```

```
else:
```

```
    print("The lists are not equal.")
```

### 2. Check if two lists have the same elements, regardless of order:

```
list1 = [1, 2, 3]
```

```
list2 = [3, 2, 1]
```

```
if sorted(list1) == sorted(list2):
```

```
    print("The lists have the same elements.")
```

```
else:
```

```
    print("The lists do not have the same elements.")
```

### 3. Check if one list is a subset of another:

```
list1 = [1, 2, 3]
```

```
list2 = [1, 2, 3, 4, 5]
```

```
if all(elem in list2 for elem in list1):
```

```
    print("list1 is a subset of list2.")
```

```
else:
```

```
    print("list1 is not a subset of list2.")
```

### 4. Find common elements between two lists:

```
list1 = [1, 2, 3]
```

```
list2 = [3, 4, 5]
```

```
common_elements = list(set(list1) & set(list2))
```

```
print(f"Common elements: {common_elements}")
```

### 5. Find elements that are in one list but not the other:

```
list1 = [1, 2, 3]
```

```
list2 = [3, 4, 5]
```

```
difference = list(set(list1) - set(list2))  
print(f"Elements in list1 but not in list2: {difference}")
```

## 5.WHAT IS TUPLE? DIFFERENCE BETWEEN LIST AND TUPLE.

### **TUPLE :**

- + A **tuple** is a collection data type in Python that is used to store multiple items in a single variable.
- + Tuples are ordered, immutable (unchangeable), and allow duplicate values.
- + They are defined by enclosing the elements in parentheses ().

Example :

```
my_tuple = (1, 2, 3, 4, 5)  
print(my_tuple)  
# Output: (1, 2, 3, 4, 5)
```

## Difference Between List and Tuple :

Feature	List	Tuple
Syntax	Defined using square brackets []	Defined using parentheses ()
Mutability	Mutable (can be changed)	Immutable (cannot be changed)
Methods	More built-in methods (e.g., append(), remove())	Fewer built-in methods (e.g., count(), index())
Performance	Slower due to mutability	Faster due to immutability
Memory Usage	Uses more memory	Uses less memory
Use Case	Suitable for collections of items that may change	Suitable for collections of items that should not change

6.HOW WILL YOU CREATE A DICTIONARY USING TUPLES IN PYTHON?



## Method 1: Using the dict() Constructor

This is the simplest and most common method.

```
# List of tuples
```

```
tuples = [('apple', 5), ('banana', 3), ('cherry', 8)]
```

```
# Convert list of tuples to dictionary
```

```
dictionary = dict(tuples)
```

```
print(dictionary)
```

```
# Output: {'apple': 5, 'banana': 3, 'cherry': 8}
```

## Method 2: Using Dictionary Comprehension

This method is concise and readable, especially for those familiar with Python's comprehension syntax.

```
# List of tuples
```

```
tuples = [('apple', 5), ('banana', 3), ('cherry', 8)]
```

```
# Convert list of tuples to dictionary using comprehension
```

```
dictionary = {key: value for key, value in tuples}
```

```
print(dictionary)
```

*# Output: {'apple': 5, 'banana': 3, 'cherry': 8}*

### **Method 3: Using a Loop**

This method is more explicit and can be useful for more complex transformations.

*# List of tuples*

*tuples = [('apple', 5), ('banana', 3), ('cherry', 8)]*

*# Initialize an empty dictionary*

*dictionary = {}*

*# Populate the dictionary using a loop*

*for key, value in tuples:*

*dictionary[key] = value*

*print(dictionary)*

*# Output: {'apple': 5, 'banana': 3, 'cherry': 8}*

**7.HOW DO YOU TRAVERSE THROUGH A  
DICTIONARY OBJECT IN PYTHON?**

✚ Traversing through a dictionary in Python can be done in several ways, depending on what you need to access: keys, values, or key-value pairs.

✚ Here are some common methods:

## 1. Iterating Over Keys

You can directly iterate over the dictionary to get the keys:

**Example :**

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

```
for key in my_dict:
```

```
    print(key)
```

## 2. Iterating Over Values

To iterate over the values, use the `.values()` method:

**Example:**

```
for value in my_dict.values():
```

```
    print(value)
```

## 3. Iterating Over Key-Value Pairs

To get both keys and values, use the `.items()` method:

**Example:**

```
for key, value in my_dict.items():
```

```
    print(f"Key: {key}, Value: {value}")
```

#### 4. Iterating Over Keys Using .keys()

You can also explicitly use the .keys() method to iterate over the keys.

**Example:**

```
for key in my_dict.keys():  
    print(key)
```

#### 5. Using Dictionary Comprehensions

For more complex operations, you can use dictionary comprehensions:

**Example:**

```
squared_values = {key: value**2 for key, value in  
my_dict.items()}  
print(squared_values)
```

### 8. HOW DO YOU CHECK THE PRESENCE OF A KEY IN A DICTIONARY?

#### 1. Using the in Operator

The simplest and most Pythonic way is to use the in operator:

**Example:**

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

```
if 'b' in my_dict:
```

```
    print("Key 'b' is present in the dictionary.")
```

```
else:
```

```
    print("Key 'b' is not present in the dictionary.")
```

## **2. Using the get() Method**

The get() method returns the value for the specified key if the key is in the dictionary, otherwise it returns None:

**Example:**

```
value = my_dict.get('b')
```

```
if value is not None:
```

```
    print("Key 'b' is present in the dictionary.")
```

```
else:
```

```
    print("Key 'b' is not present in the dictionary.")
```

## **3. Using the keys() Method**

You can also use the keys() method to get a view object of all the keys in the dictionary and check for the key:

**Example :**

```
if 'b' in my_dict.keys():
```

*print("Key 'b' is present in the dictionary.")*

*else:*

*print("Key 'b' is not present in the dictionary.")*

#### **4. Handling KeyError Exception**

Another way is to handle the KeyError exception when trying to access a key:

**Example:**

*try:*




*value = my\_dict['b']*

*print("Key 'b' is present in the dictionary.")*

*except KeyError:*

*print("Key 'b' is not present in the dictionary.")*

## **9. WHY DO YOU USE THE ZIP () METHOD IN PYTHON?**

-  The zip() method in Python is a powerful and versatile tool used to combine multiple iterables (like lists, tuples, etc.) into a single iterable of tuples.
-  The zip() function is useful for combining, iterating, and manipulating multiple iterables in a clean and efficient way.
-  Here are some common reasons to use zip():

## 1. Parallel Iteration

zip() allows you to iterate over multiple iterables in parallel, pairing corresponding elements together:

**Example :**

```
names = ['Alice', 'Bob', 'Charlie']
```

```
scores = [85, 92, 78]
```

```
for name, score in zip(names, scores):
```

```
    print(f"{name} scored {score}")
```

## 2. Creating Dictionaries

You can use zip() to create dictionaries from two lists, one for keys and one for values:

**Example :**

```
keys = ['name', 'age', 'city']
```

```
values = ['Alice', 30, 'New York']
```

```
my_dict = dict(zip(keys, values))
```

```
print(my_dict)
```

## 3. Unzipping

zip() can also be used in reverse to unzip a list of tuples into separate lists:

**Example :**

```
pairs = [('a', 1), ('b', 2), ('c', 3)]
```

```
letters, numbers = zip(*pairs)
```

```
print(letters) # Output: ('a', 'b', 'c')
```

```
print(numbers) # Output: (1, 2, 3)
```

#### **4. Handling Unequal Lengths**

When the input iterables have different lengths, `zip()` stops creating tuples when the shortest input iterable is exhausted:

**Example :**

```
list1 = [1, 2, 3]
```

```
list2 = ['a', 'b']
```

```
zipped = list(zip(list1, list2))
```

```
print(zipped)
```

```
# Output: [(1, 'a'), (2, 'b')]
```

#### **5. Simplifying Code**

Using `zip()` can make your code more readable and concise, especially when dealing with multiple iterables.



## 10. HOW MANY BASIC TYPES OF FUNCTIONS ARE AVAILABLE IN PYTHON?

 In Python, there are three basic types of functions:

### 1. Built-in Functions

These are functions that are already defined in Python and are always available for use.

Example : `print()`, `len()`, `max()`, `min()`, and many more.

### 2. User-defined Functions

These are functions that you create yourself to perform specific tasks. You define them using the `def` keyword. For example:

**Example :**

```
def greet(name):  
    return f"Hello, {name}!"
```

### 3. Anonymous Functions (Lambda Functions)

These are small, unnamed functions defined using the `lambda` keyword. They are typically used for short, simple operations. For example:

**Example :**

```
square = lambda x: x ** 2  
print(square(5))
```

# *Output: 25*

## 11. HOW CAN YOU PICK A RANDOM ITEM FROM A LIST OR TUPLE?

- + You can pick a random item from a list or tuple in Python using the `random.choice()` function from the `random` module.
- + **Import the random module:** This module provides various functions to generate random numbers and select random items.
- + **Use `random.choice()`:** This function takes a non-empty sequence (like a list or tuple) as an argument and returns a randomly selected item from it.

**Example:**

```
import random
```

```
my_tuple = (1, 2, 3, 4, 5)
```

```
random_item = random.choice(my_tuple)
```

```
print(random_item)
```

## 12. HOW CAN YOU PICK A RANDOM ITEM FROM A RANGE?

You can pick a random item from a range in Python using the `random.randrange()` or `random.randint()` functions from the `random` module. Here's how you can do it:

### **Using `random.randrange()`**

The `random.randrange()` function returns a randomly selected element from the specified range. The range is exclusive of the stop value.

```
import random
```

```
# Example: Pick a random number from 0 to 9
```

```
random_number = random.randrange(0, 10)
```

```
print(random_number)
```

### **Using `random.randint()`**

The `random.randint()` function returns a random integer within the specified inclusive range.

```
import random
```

```
# Example: Pick a random number from 0 to 9
```

```
random_number = random.randint(0, 9)
```

```
print(random_number)
```

## Explanation

1. **Import the random module:** This module provides various functions to generate random numbers.
2. **Use random.randrange(start, stop):** This function generates a random number from the range [start, stop), meaning it includes the start value but excludes the stop value.
3. **Use random.randint(start, stop):** This function generates a random number from the range [start, stop], meaning it includes both the start and stop values.

## 13.HOW CAN YOU GET A RANDOM NUMBER IN PYTHON?

### Method 1: Using random.randint()

This function returns a random integer between the specified range (inclusive).

#### Example:

```
import random
```

```
# Generate a random integer between 1 and 10
```

```
random_number = random.randint(1, 10)
```

```
print(random_number)
```

### Method 2: Using random.random()

This function returns a random float between 0.0 and 1.0.

**Example :**

```
import random
```

```
# Generate a random float between 0.0 and 1.0
```

```
random_float = random.random()
```

```
print(random_float)
```

**Method 3: Using random.uniform()**

This function returns a random float between the specified range.

**Example:**

```
import random
```

```
# Generate a random float between 1.0 and 10.0
```

```
random_float = random.uniform(1.0, 10.0)
```

```
print(random_float)
```

**Method 4: Using random.choice()**

This function returns a random element from a non-empty sequence.

**Example:**

```
import random
```

```
# Generate a random element from a list  
my_list = [1, 2, 3, 4, 5]  
random_element = random.choice(my_list)  
print(random_element)
```

### **Method 5: Using random.sample()**

This function returns a specified number of unique elements from a sequence.

#### **Example:**

```
import random  
  
# Generate a list of 3 unique random elements from a list  
my_list = [1, 2, 3, 4, 5]  
random_sample = random.sample(my_list, 3)  
print(random_sample)
```

## **14. HOW WILL YOU SET THE STARTING VALUE IN GENERATING RANDOM NUMBERS?**

- ✚ To set the starting value (or seed) for generating random numbers in Python, you can use

the `random.seed()` function. Setting the seed ensures that you get the same sequence of random numbers each time you run the program, which is useful for reproducibility and debugging.

Here's how you can set the seed:

**Example :**

```
import random
```

```
# Set the seed
```

```
random.seed(42)
```

```
# Generate random numbers
```

```
print(random.randint(1, 10)) # Output will be the same  
every time you run the program
```

```
print(random.random())      # Output will be the same every  
time you run the program
```

```
print(random.uniform(1.0, 10.0)) # Output will be the same  
every time you run the program
```

**Explanation:**

1. **Import the random module:** This module contains functions for generating random numbers.

2. **Set the seed:** `random.seed(42)` sets the starting value for the random number generator. You can use any integer as the seed value.
  3. **Generate random numbers:** The random numbers generated after setting the seed will be the same each time you run the program.
- ✚ By setting the seed, you ensure that the sequence of random numbers is predictable and repeatable, which can be very helpful for testing and debugging.

## 15. HOW WILL YOU RANDOMIZES THE ITEMS OF A LIST IN PLACE?

- ✚ You can randomize the items of a list in place using the `random.shuffle()` function from Python's random module.
- ✚ Here's how you can do it:

**Example:**

```
import random
```

```
# Create a list of items
```

```
my_list = [1, 2, 3, 4, 5]
```



```
# Shuffle the list in place  
random.shuffle(my_list)
```

```
# Print the shuffled list  
print(my_list)
```