PYTHON TUTORIAL FOR BEGINNERS

Source: www.youtube.com/@RishabhMishraOfficial

Chapter - 16

List in Python

- What is List
- Create Lists
- Access List: Indexing & Slicing
- Modify List
- List Methods
- Join Lists
- List Comprehensions
- Lists Iteration

List in Python

A **list** in Python is a collection of items (elements) that are **ordered**, **changeable** (mutable), and allow **duplicate** elements.

Lists are one of the most versatile data structures in Python and are used to store multiple items in a single variable.

Example:

```
fruits = ["apple", "orange", "cherry", "apple"]
print(fruits)
# Output: ['apple', 'orange', 'cherry', 'apple']
```

Create List in Python

You can **create lists** in Python by placing **comma-separated** values between **square brackets []**. Lists can contain elements of different data types, including other lists.

```
Syntax:  list_name = [element1, element2, element3, ...]
# List of strings
colors = ["red", "green", "blue"]
# List of integers
numbers = [1, 2, 3, 4, 5]
# Mixed data types
mixed = [1, "hello", 3.14, True]
# Nested list
nested = [1, [2, 3], [4, 5, 6]]
```

Accessing List Elements - Indexing

You can **access elements** in a list by referring to their **index**. Python uses **zero-based** indexing, meaning the first element has an index of **0**.

Syntax: `list_name[index]`

Example:

```
fruits = ["apple", "orange", "cherry", "apple", "mango"]
```

Index	0	1	2	3	4
	-5	-4	-3	-2	-1

```
# Access first element
print(fruits[0])   # Output: apple
# Access third element
print(fruits[2])   # Output: cherry
# Access last element using negative index
print(fruits[-1])  # Output: mango
```

List Slicing

Slicing allows you to **access a range of elements** in a list. You can specify the **start and stop indices**, and Python returns a new list containing the specified elements.

Syntax: `list_name[start:stop:step]`

Example: `numbers = [10, 20, 30, 40, 50, 60]`

Index	0	1	2	3	4	5
	-6	-5	-4	-3	-2	-1

```
# Slice from index 1 to 3
print(numbers[1:4])  # Output: [20, 30, 40]

# Slice from start to index 2
print(numbers[:3])  # Output: [10, 20, 30]

# Slice all alternate elements
print(numbers[0::2])  # Output: [10, 30, 50]

# Slice with negative indices
print(numbers[-4:-1])  # Output: [30, 40, 50]
```

```
# Reverse list
print(numbers[::-1])  # Output: [60,50,40,30,20,10]
```

Modifying List

Lists are **mutable**, meaning you can change their content after creation. You can **add**, **remove**, or **change** elements in a list.

```
# Initial list:  fruits = ["apple", "banana", "cherry"]

# Changing an element
fruits[1] = "blueberry"
print(fruits)    # Output: ['apple', 'blueberry', 'cherry']

# Adding an element
fruits.append("mango")
print(fruits)    # Output: ['apple', 'blueberry', 'cherry', 'mango']

# Removing an element
fruits.remove("cherry")
print(fruits)    # Output: ['apple', 'blueberry', 'mango']
```

List Methods

Python provides several **built-in methods** to **modify** and **operate** on lists. Eg:

Methods	Description
append()	Adds a single element to the end of the list.
extend()	Adds multiple elements (from another iterable) to the end of the list.
insert()	Inserts an element at a specified position.
remove()	Removes the first occurrence of a specified element.
pop()	Removes and returns an element at a specified index. If no index is specified, it removes and returns the last element.
clear()	Removes all elements from the list, resulting in an empty list.
index()	Returns the index of the first occurrence of a specified element.
count()	Returns the number of occurrences of a specified element.
sort()	Sorts the list in ascending order by default. You can also sort in descending order and specify custom sorting criteria.
reverse()	Reverses the elements of the list in place.
copy()	Returns a shallow copy of the list.

Join Lists

There are several ways to **join**, or **concatenate**, two or more lists in Python.

```
list1 = [1, 2]
list2 = ["a", "b"]

# One of the easiest ways are by using the + operator
list3 = list1 + list2
print(list3) # Output: [1, 2, 'a', 'b']

# using append method
for x in list2:
  list1.append(x)
# appending all the items from list2 into list1, one by one
print(list1) # Output: [1, 2, 'a', 'b']

# using extend method
list1.extend(list2)   # add elements from one list to another list
print(list1) # Output: [1, 2, 'a', 'b']
```

List Comprehensions

List comprehensions provide a **concise way to create lists**. They consist of brackets containing an expression followed by a for clause, and optionally if clauses.

```
# Syntax:
new_list = [expression for item in iterable if condition]

# Creating a list of squares:
squares = [x**2 for x in range(1, 6)]
print(squares)  # Output: [1, 4, 9, 16, 25]

# Filtering even numbers:
even_numbers = [x for x in range(1, 11) if x % 2 == 0]
print(even_numbers)  # Output: [2, 4, 6, 8, 10]

# Applying a function to each element:
fruits = ["apple", "banana", "cherry"]
uppercase_fruits = [fruit.upper() for fruit in fruits]
print(uppercase_fruits)  # Output: ['APPLE', 'BANANA', 'CHERRY']
```

List Comprehensions - Flatten a List

```
Flatten a Nested List - using List Comprehension
    def flatten_list(lst):
        return [item for sublist in lst for item in sublist]
    # Example
    nested_list = [[1, 2], [3, 4], [5, 6]]
    flattened = flatten_list(nested_list)
    print(flattened)
    # Output: [1, 2, 3, 4, 5, 6]
```

Iterating Over Lists

Iterating allows you to traverse each element in a list, typically using loops.

```
#Example: fruits = ["apple", "banana", "cherry"]

# Using for loop
for fruit in fruits:
    print(fruit)

# Using while loop
index = 0
while index < len(fruits):
    print(fruits[index])
    index += 1
```

Python Tutorial Playlist: **Click Here**

https://www.youtube.com/playlist?list=PLdOKnrf8EcP384Ilxra4UlK9BDJGwawg9