List Basics

Table of contents

01

Introduction

Introduction

A **list** in Python is a built-in data type that is used to store multiple items in a single variable.

Characteristics of Lists

- **Ordered**: The elements in a list maintain the order in which they were inserted.
- **Mutable**: You can modify elements in a list (add, remove, or change elements).
- **Allow Duplicates**: Lists can have repeated elements.
- **Can Contain Mixed Data Types**: Lists can hold integers, floats, strings, and even other lists or objects.

02

Creating a List

Creating a List

Lists are created using **square brackets [],** with elements separated by **commas**.

Syntax
`my_list = [item1, item2, … , item n]`

Creating a List - Examples

List of integers	`numbers = [10, 20, 30, 40]`
List of strings	`fruits = ["apple", "banana", "cherry"]`
List with mixed data types	`mixed = [1, "hello", 3.14, True]`
Nested list	`nested_list = [[1, 2, 3], ["a", "b", "c"]]`
Empty list	`empty_list = []`

03

Accessing a List

Accessing a List

You can **access** elements in a list using

- **Indexing**
- Slicing

Indexing (Positive & Negative)

Each element in a list has an index:

- **Positive indexing**: Starts from 0 for the first element.
- **Negative indexing**: Starts from -1 for the last element.

Python	Output

`my_list = ["apple", "banana", "cherry", "date"]`
 	
`# positive indexing`
`print(my_list[0])`
`print(my_list[2])`
 	
`# negative indexing`
`print(my_list[-1])`
`print(my_list[-3])`
	Apple
	Cherry
	
	Date
	banana

Accessing a List

You can **access** elements in a list using
- Indexing
- **Slicing**

Slicing

Slicing is used to access a **subset** of elements in a list.

Slicing Tricks:
- Slicing [start : stop]
- Slicing with step [start : stop : step]
- Omitting start (default is 0)
- Omitting stop (default is end)
- Reverse a list using slicing

Python	Output

`my_list = [10, 20, 30, 40, 50, 60]`
 	
`# [start:stop]`
`print(my_list[1:4])`
`# [start:stop:step]`
`print(my_list[::2])`
`# Omitting start`
`print(my_list[:3])`
`# Omitting stop`
`print(my_list[2:])`
`# Reverse a list using slicing`
`print(my_list[::-1])`
	`[20, 30, 40]`
	
	`[10, 30, 50]`
	
	`[10, 20, 30]`
	
	`[30, 40, 50, 60]`
	
	`[60, 50, 40, 30, 20, 10]`

04

Modifying a List

Modifying a List - Changing

- List is a mutable data type.
- The contents of list can be modified.
- Use index position to assign a new value to the list.

Syntax
`list1[i] = newvalue`

Modifying a List - Changing

- List items can be changed in **two** ways:
- **Changing Elements**
- Changing Range of Elements

Python	Output

`my_list = ["apple", "banana", "cherry"]`
`my_list[1] = "blueberry"`
`print(my_list)`
	`['apple', 'blueberry', 'cherry']`

Modifying a List - Changing

- List items can be changed in **two** ways:
- Changing Elements
- **Changing Range of Elements**

Python	Output

`my_list = ["apple", "banana", "cherry"]`
`my_list[1:3] = [“papaya“, “mango“]`
`print(my_list)`
	`['apple', 'papaya', 'mango']`

Modifying a List - Changing

Note:

- **Adding more items than you replace shifts existing ones forward.**
- Adding fewer items than you replace adjusts the list accordingly.

Python	Output

`my_list = ["apple", "banana", "cherry"]`
`my_list[1:2] = [“papaya“, “mango“]`
`print(my_list)`
	`['apple', 'papaya', 'mango', 'cherry']`

Modifying a List - Changing

Note:

- Adding more items than you replace shifts existing ones forward.
- **Adding fewer items than you replace adjusts the list accordingly.**

Python	Output

`my_list = ["apple", "banana", "cherry"]`
`my_list[1:3] = ['papaya']`
`print(my_list)`
	`['apple', 'papaya']`

05

Adding items to a List

Adding items to a List

There are three main ways to add items to a list:

- **append():** Adds a single element to the end of the list.
- **insert():** Adds an element at a specific index, shifting existing elements.
- **extend():** Adds multiple elements from another iterable (like a list).

Let’s Learn them **one – by – one**

Adding items to a List – append()

- Adds a **single** element to the **end** of the list.

Syntax
`my_list.append(new_item)`

Python	**Output**

`fruits = ["apple", "banana", "cherry"]`
`fruits.append("orange")`
`print(fruits)`
	`['apple', 'banana', 'cherry', 'orange']`

Adding items to a List – insert()

- Adds an element at a **specific index**, shifting existing elements.

Syntax
`my_list.insert(index, new_item)`

Python	**Output**

`subjects = ["Math", "Physics", "Biology"]`
`subjects.insert(1, "Chemistry")`
`print(subjects)`
	`['Math', 'Chemistry', 'Physics', 'Biology']`

Adding items to a List – insert()

Negative Indices:

If we insert at **-1**, it does **NOT** mean the last position but before the last item.

Python	Output

`subjects = ["Math", "Physics", "Biology"]`
`subjects.insert(-1, "Chemistry")`
`print(subjects)`
	`['Math', 'Physics', 'Chemistry', 'Biology']`

Adding items to a List – extend()

- Adds **multiple elements** from another iterable (like a list).

Syntax
`my_list.extend(iterable)`

Python	**Output**

`list1 = [1, 2, 3]`
`list2 = [4, 5, 6]`
 	
`list1.extend(list2)`
`print(list1)`
	`[1, 2, 3, 4, 5, 6]`

06

Removing items from a List

Removing items from a List

- Python lists allow us to **remove** elements in multiple ways, depending on whether we want to delete a specific value, an index, or all elements.
- The main methods for removing items are:

1. **remove():** Removes the first occurrence of a specific value.
2. **pop():** Removes the **last element** an element.
3. **clear():** Removes all elements.
4. **del:** Deletes elements at a specific index or a range of indices.

Removing items from a List – remove()

- Removes the **first occurrence** of a specific value.
- If the value is not found, it **raises an error**.

Syntax
`my_list.remove(value)`

Python	**Output**

`fruits = ["apple", "banana", "cherry", "banana"]`
`fruits.remove("banana")`
`print(fruits)`
	`['apple',`
	`'cherry',`
	`'banana']`

Removing items from a List – pop()

- Removes and returns an element.
- By default, removes the last element.
- Can remove an element at a specific index.

Syntax
`my_list.pop()`
`my_list.pop(index)`

Python	Output

`numbers = [10, 20, 30, 40]`
`removed_item = numbers.pop(1)`
`print(numbers)`
`print("Removed:", removed_item)`
	`[10, 30, 40]`
	`Removed: 20`

Removing items from a List – clear()

- Removes **all elements**, leaving an **empty list**.

Syntax
`my_list.clear()`

Python	**Output**

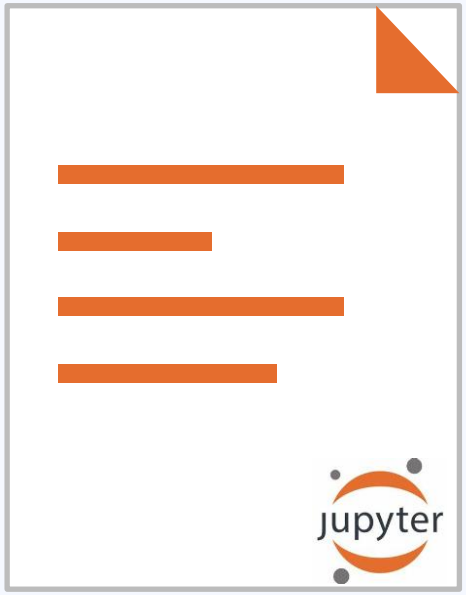
`items = ["pen", "notebook", "eraser"]`
`items.clear()`
`print(items)`
	`[]`

Removing items from a List – del

- Deletes elements at a **specific index** or a **range** of indices.
- Can also delete the **entire list**.

Python	Output

`numbers = [10, 20, 30, 40]`
 	
`del numbers[2]`
`print(numbers)`
 	
`del numbers[1:2]`
`print(numbers)`
 	
`del numbers`
	
	`[10, 20, 40]`
	`[10, 40]`

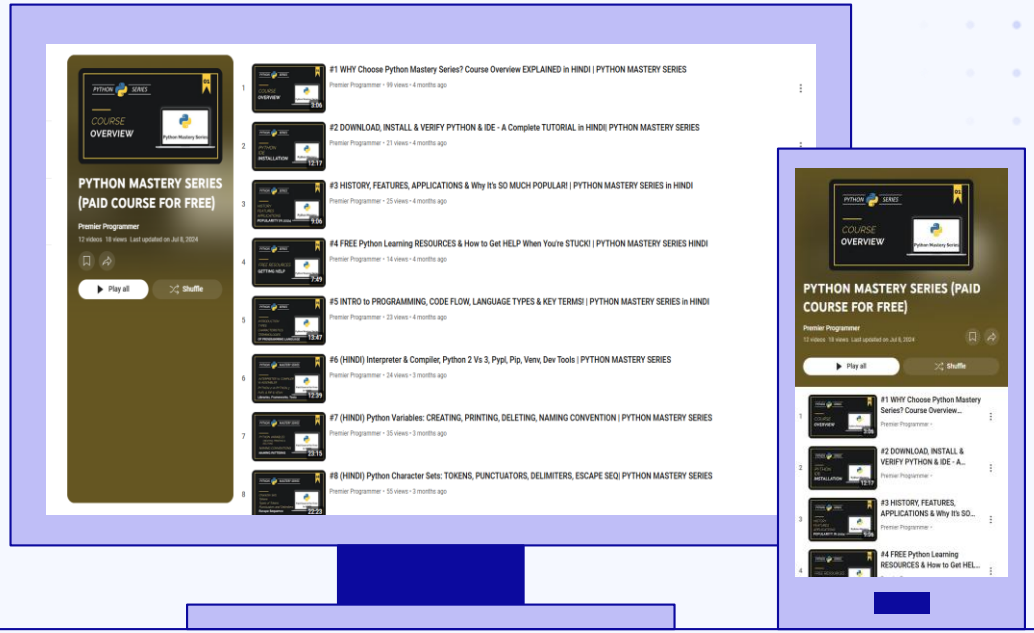


List Basics

Download Link in
Description
and
Pinned Comment

WATCH

Level up your coding with each episode in this focused Python series.



Next Video!

List Basics
Practice Set Solution