Python Sets

Table of contents

01

Introduction

Introduction

- A **set** is an unordered, mutable collection of unique elements.
- Sets are primarily used for **membership testing** and **removing duplicates** from data.

CHARACTERISTICS:

- **Unordered** – Elements have no fixed order.
- **Unindexed** – No indexing or slicing like lists or tuples.
- **Unique Elements** – Duplicate values are automatically removed.
- **Mutable** – You can add or remove elements after creation.

02

Creating Set

Creating Set

Sets are created using **Curly Braces {} or set() Constructor**, with elements separated by **commas**.

Syntax

```
my_set = {item1, item2, … , item n}
```

Creating Set – Using {}

Sets are created using **Curly Braces {} or set() Constructor**, with elements separated by **commas**.

Python	Output
`my_set = {1, 2, 3, 4}` `print(my_set)`	`{1, 2, 3, 4}`

Creating Set – Using {}

Duplicate elements are ignored or removed.

Python	Output
`my_set = {1, 2, 2, 3}` `print(my_set)`	`{1, 2, 3}`

Creating Set – Using set()

- **set()** Constructor creates a set from an **iterable** (list, tuple, string, etc.).

Python

```
list_set = set([1, 2, 3, 4, 5])
char_set = set("hello")
```

Creating Set – Empty Set

For creating an **Empty Set**, use **set()** instead of **{}** to avoid confusion with an **empty dictionary**.

Python	Output

`empty_set = set()`
`print(type(empty_set))`
`---------------------------`
`wrong_set = {}`
`print(type(wrong_set))`
	`<class 'set'>`
	
	`<class 'dict'>`

Accessing Set

"Since sets are **unordered** and **unindexed**, you ***CANNOT*** access elements using an *index*."

03

Adding Elements

Adding Elements

- Sets are **mutable**, meaning they **can be modified** after creation.
- The **elements** within the set must be **immutable** (e.g., integers, strings, or tuples).
- Mutable elements like **lists** or **dictionaries** *cannot* be added to a set.
- To add a **single item** to a set, use the **add()** method.
- To include **multiple items** from another set into the current set, use the **update()** method.

Adding Elements – using add()

Syntax	
`set.add(element)`	
Python	**Output**

`my_set = {1, 2, 3}`
`my_set.add(4)`
`print(my_set)`
	`{1, 2, 3, 4}`

Adding Elements – using add()

Duplicates are ignored:

Syntax	
`set.add(element)`	
Python	**Output**

`my_set = {1, 2, 3}`
`my_set.add(3)`
`print(my_set)`
	`{1, 2, 3}`

Adding Elements – using update()

Syntax	
`set.update(iterable)`	
Python	**Output**

`my_set = {1, 2, 3, 4}`
`my_set.update([5, 6, 7])`
`print(my_set)`
 	
`my_set.update((8, 9))`
`print(my_set)`
 	
`my_set.update("abc")`
`print(my_set)`
	`{1, 2, 3, 4, 5, 6, 7}`
	
	`{1, 2, 3, 4, 5, 6, 7, 8, 9}`
	
	`{1, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 'c', 'b'}`

04

Removing Elements

Removing Elements

- Python provides several methods to **remove** elements from a set.

- They are:
  - .remove()
  - .discard()
  - .pop()
  - .clear()
  - del

- Let’s learn them one-by-one

Removing Elements – using remove()

- Removes a **specific element** from the set.
- Raises a **KeyError** if the element is **not found**.

Syntax	
`set.remove(element)`	
Python	**Output**

`my_set = {1, 2, 3, 4, 5}`
`my_set.remove(3)`
`print(my_set)`
`my_set.remove(10)`
`print(my_set)`
	`{1, 2, 4, 5}`
	
	`# Raises KeyError: 10`

Removing Elements – using discard()

- Also removes a **specific element** from the set.
- Does **NOT** raise an **error** if the element is **not found**.

Syntax	
`set.discard(element)`	
Python	**Output**

`my_set = {1, 2, 3, 4, 5}`
`my_set.discard(3)`
`print(my_set)`
`my_set.discard(10)`
`print(my_set)`
	`{1, 2, 4, 5}`
	
	`# No error, does nothing`

Removing Elements – using pop()

- **Removes** and returns a **random element** from the set.
- Since sets are **unordered**, you **cannot predict** which element will be removed.
- Raises **KeyError** if the set is **empty**.

Syntax	
`set.pop(element)`	
Python	**Output**

`my_set = {10, 20, 30, 40, 50}`
`removed_item = my_set.pop()`
`print(removed_item)`
	`# Random element removed`

Removing Elements – using clear()

- Removes **all elements** from the set, making it **empty**.

Syntax	
`set.clear()`	
Python	**Output**

`my_set = {1, 2, 3, 4, 5}`
`my_set.clear()`
`print(my_set)`
	`set()`

Removing Elements – using del

- Deletes the **entire set from memory**.
- After deletion, **trying to access** the set raises a **NameError**.

Syntax	
`del set_name`	
Python	**Output**

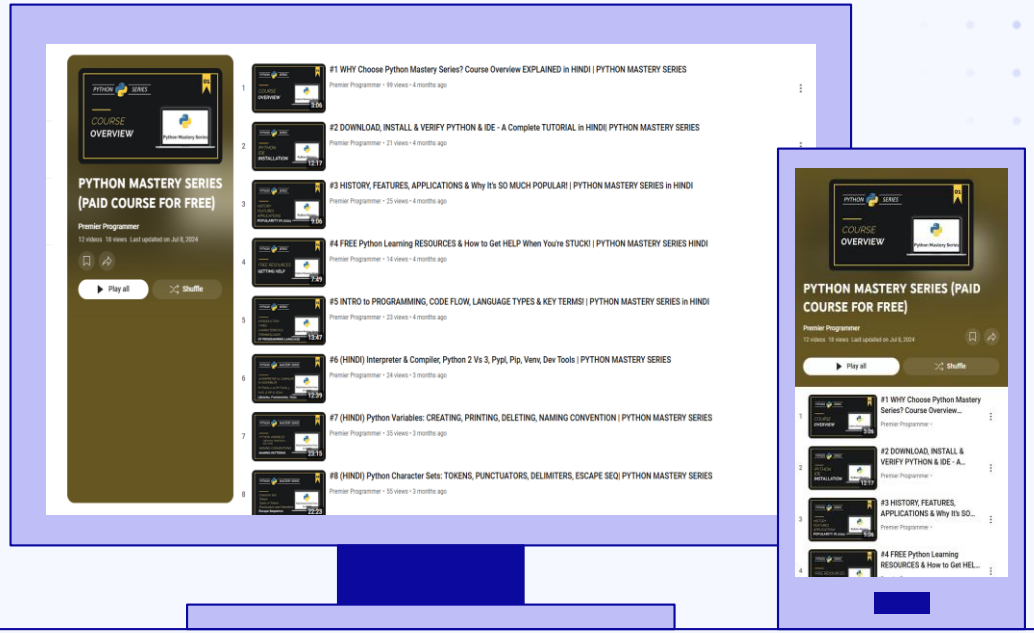
`my_set = {1, 2, 3}`
`del my_set`
`print(my_set)`
	`# Raises NameError: name 'my_set' is not defined`

Practice Set - 1

Download Link in **Description** and **Pinned Comment**

WATCH

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



Next Video!

Practice Set - 1
Solution