Tuple Basics

Table of contents

01

Introduction

Introduction

- Tuple is one of the **built-in data types** in Python.
- It is a sequence of **comma separated items**.
- It is enclosed in **parentheses ()**.
- **Faster** than lists due to **immutability** (fixed memory allocation).

CHARACTERISTICS:

- **Immutable** → Once created, elements cannot be changed, added, or removed.
- **Ordered** → Items maintain their order and can be accessed via an index.
- **Allows Duplicates** → Can contain duplicate values.

02

Creating Tuple

Creating Tuple

Ways	Examples
Using Parentheses **()**	`my_tuple = (1, 2, 3, 4, 5)`
***Tuple with a **Single Element**	`my_tuple = (10,)`
Using the **tuple()** Constructor	`my_tuple = tuple([1, 2, 3, 4, 5])`
Creating an **Empty** Tuple	`empty_tuple = ()`
Creating a **Nested** Tuple	`nested_tuple = (1, (2, 3), (4, 5, 6))`
Mixed Data Types	`my_tuple = ("a", 12, True, 3.14)`

*** For a single-element tuple, a trailing comma , is required to differentiate it from a regular variable. Without the comma, it is just an integer.**

03

Accessing Tuple

Accessing Tuple

Indexing → Access elements using their index, starting from 0.

Python

```
my_tuple = (10, 20, 30)
print(my_tuple[1])
```

Output

```
20
```

Accessing Tuple

Negative Indexing → Use negative indices to access elements from the end.

Python

```
my_tuple = (10, 20, 30)
print(my_tuple[-1])
```

Output

```
30
```

Accessing Tuple

Slicing → Extract a portion of the tuple using slicing [start : stop : step].

Python

```
my_tuple = (10, 20, 30)
print(my_tuple[0:2])
```

Output

```
(10, 20)
```

Accessing Tuple

Accessing Nested Tuples → Use multiple indices for nested structures.

Python

```
nested_tuple = ((1, 2), (3, 4))
print(nested_tuple[1][0])
```

Output

```
3
```

04

Updating Tuple

Updating Tuple

Since **tuples are immutable**, they **cannot be modified directly** after creation. However, there are some workarounds to "update" a tuple.

Updating Tuple

1. **Convert to List and Modify**

Convert the **tuple** to a **list**, make changes, and **convert** it back to a **tuple**.

Python	Output

`my_tuple = (1, 2, 3)`
 	
`# Convert to list`
`temp_list = list(my_tuple)`
 	
`# Modify element`
`temp_list[1] = 20`
 	
`# Convert back to tuple`
`my_tuple = tuple(temp_list)`
`print(my_tuple)`
	`(1, 20, 3)`

Updating Tuple

2. Concatenation (Creating a New Tuple)

Add new elements by concatenating tuples.

Python

```
my_tuple = (1, 2, 3)
new_tuple = my_tuple + (4, 5)
print(new_tuple)
```

Output

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

05

Unpacking Tuple

Unpacking Tuple

1. Basic Unpacking (known number of values in the tuple) :

Assign tuple elements to **variables**.

Python

```
my_tuple = (1, 2, 3)
a, b, c = my_tuple
print(a, b, c)
```

Output

```
1 2 3
```

Unpacking Tuple

2. Extended Unpacking (unknown number of values in the tuple) :

Use an **asterisk (*)** to collect the remaining values as a list.

Python

```
my_tuple = (1, 2, 3, 4, 5)
a, *middle, b = my_tuple
print(a, middle, b)
```

Output

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

Unpacking Tuple

2. Extended Unpacking (unknown number of values in the tuple) :

When unpacking a tuple, the number of variables on the left must match the number of elements in the tuple. If they don't, Python raises a ValueError.

Too Many Variables (More Variables Than Tuple Elements)

```
ValueError: too many values to unpack (expected 2)
```

Too Few Variables (Fewer Variables Than Tuple Elements)

```
ValueError: not enough values to unpack (expected 4, got 3)
```

Unpacking Tuple

3. Ignoring Values with _:

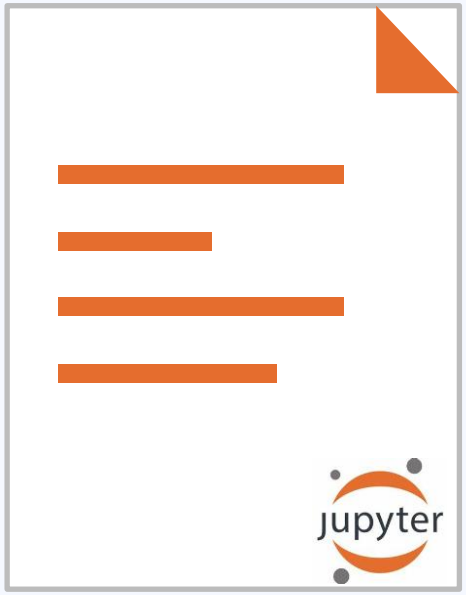
Use an **underscore (_)** to ignore element(s).

Python

```
my_tuple = (10, 20, 30)
a, _, c = my_tuple
print(a, c)
```

Output

```
10 30
```

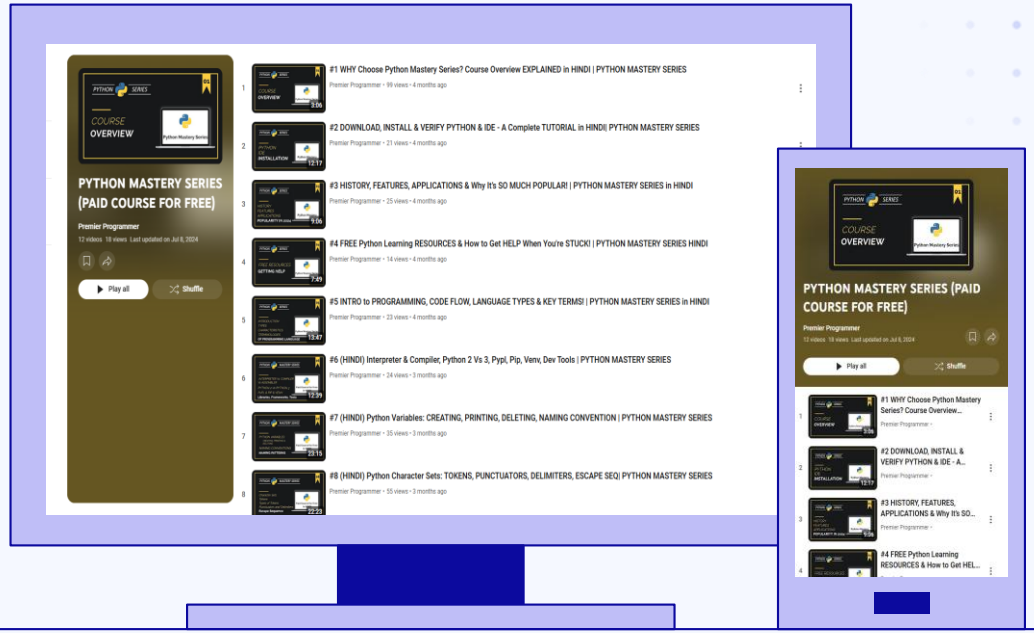


Practice Set - 1

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