Python Programming
Unit 02 – Lecture 02 Notes
Tuples and Immutability

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

Tuples are Python's "fixed" (immutable) sequence type. They are used for records that should not change (coordinates, configuration values, keys, etc.). In this lecture we learn how to create tuples, access elements, unpack values, and choose tuples vs lists.

2 Core Concepts

2.1 Why Tuples? (The Idea of Immutability)

A tuple is an **immutable sequence**. "Immutable" means:

- you cannot change its length (no append/remove), and
- you cannot replace an element (no `t[0] = ...`).

Why is that useful?

- It prevents accidental changes to important values.
- It communicates intent: "this is a fixed record."
- Tuples can be used as dictionary keys *if all their elements are hashable.*

2.2 Creating Tuples

```
point = (3, 4) # parentheses
packed = 1, 2, 3 # packing without parentheses
single = (5,) # singleton tuple needs a comma
from_list = tuple([1, 2, 3])
empty = ()
```

The comma is what makes a tuple. Parentheses mostly help with readability.

2.3 Indexing, Slicing, and Iteration

Tuples support most of the *non-mutating* sequence operations:

```
t = (10, 20, 30, 40)
t[0] # 10
t[-1] # 40
t[1:3] # (20, 30)
```

You can iterate normally:

```
for value in t:
    print(value)
```

2.4 Packing and Unpacking

Unpacking assigns tuple positions to variables:

```
point = (10, 20)
x, y = point
```

Star-unpacking collects the remaining values into a list:

```
values = (10, 20, 30, 40, 50)
a, b, *rest = values
# a = 10, b = 20, rest = [30, 40, 50]
```

Two high-frequency patterns:

- swapping: `a, b = b, a`
- multiple return values:

```
def min_max(nums):
    return min(nums), max(nums)

mn, mx = min_max([3, 1, 7, 2])
```

2.5 Tuple Methods (Small but Useful)

Tuples have only two methods:

- `count(x)`: how many times `x` appears
- `index(x)`: position of the first occurrence of `x`

Most other operations (like sorting) are done by converting to a list or using `sorted`.

2.6 Tuples vs Lists (How to Choose)

Feature	List	Tuple
Mutability	Mutable	Immutable
Syntax	`[]`	`()`
Best use	Changeable collection	Fixed record
As dict key	No	Yes (if hashable contents)

If you expect updates (append/remove/sort in place), use a list. If the values represent a fixed record, use a tuple.

3 Common Pitfalls

3.1 Singleton Tuples

`(5)` is just the integer `5` in parentheses. The comma creates a singleton tuple:

```
a = (5) # int
b = (5,) # tuple
```

3.2 "Tuples Are Immutable" (But Inner Objects Can Mutate)

The tuple structure cannot change, but a tuple can contain a mutable object. Example:

```
t = ([1, 2],) # tuple containing a list
# t[0] = [9] # not allowed (would replace an element)
t[0].append(3) # allowed (mutates the inner list)
```

This surprises beginners. The rule is: *the tuple does not change; the contained object can.*

4 Demo Walkthrough: Distance Between Two Points

Script: `demo/tuple_distance.py`

4.1 Math Behind the Demo

For points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, the Euclidean distance is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Using tuples for points is natural because a coordinate pair is a fixed record.

4.2 Suggested Run Steps

```
python demo/tuple_distance.py
```

Suggested student activity: change input points and predict whether the distance increases or decreases before running.

5 Quiz and Solutions

5.1 Checkpoint 1 (Why `(5)` Is Not a Tuple)

Question. Why does `(5)` not create a tuple?

Solution. In Python, the **comma** makes the tuple. Parentheses are optional in many cases. `(5)` is just an integer in parentheses. To create a singleton tuple, write `(5,)`.

5.2 Checkpoint 2 (Unpacking with `*rest`)

Question. Predict the result:

```
values = (10, 20, 30, 40, 50)
a, b, *rest = values
```

What are `a`, `b`, and `rest`?

Solution. `a = 10`, `b = 20`, `rest = [30, 40, 50]`.

5.3 Think-Pair-Share (Where Immutability Helps)

Prompt. Where in programs should immutability be preferred?

Sample response. Use immutable data for values that should never change during execution (e.g., a GPS coordinate in a log entry, a configuration setting, or a composite key in a dictionary). It reduces bugs because functions cannot accidentally modify those records.

5.4 Exit Question (Convert List $\leftrightarrow$ Tuple)

Question. Write one line to convert a list `L` to a tuple and back.
Solution. `t = tuple(L)` converts list to tuple, and `L = list(t)` converts back.

6 Practice Exercises (With Solutions)

6.1 Exercise 1: Use Tuples as Dictionary Keys

Task. Store distances between cities using a tuple key `("CityA", "CityB")`.
Solution.

```
dist = {}
dist[("Dehradun", "Delhi")] = 245
print(dist[("Dehradun", "Delhi")])
```

6.2 Exercise 2: Swap and Return Multiple Values

Task. Write a function that returns both the sum and product of two numbers.
Solution.

```
def sum_and_product(a, b):
    return a + b, a * b

s, p = sum_and_product(3, 4)
```

6.3 Exercise 3: Safe Update Workflow (Tuple $\rightarrow$ List $\rightarrow$ Tuple)

Task. Given a tuple of coordinates, update the first coordinate by 1.
Solution.

```
coords = (10, 20)
tmp = list(coords)
tmp[0] += 1
coords = tuple(tmp)
```

7 Further Reading

- `https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences`