Python Programming

Unit 02 – Lecture 02: Tuples and Immutability

Tofik Ali

School of Computer Science, UPES Dehradun

February 14, 2026

Repository: `https://github.com/tali7c/Python-Programming`

Quick Links

Core Concepts Demo Interactive Summary

Agenda

Learning Outcomes

- Create tuples and use indexing and slicing

Learning Outcomes

- Create tuples and use indexing and slicing
- Apply tuple packing and unpacking

Learning Outcomes

- Create tuples and use indexing and slicing
- Apply tuple packing and unpacking
- Compare tuples with lists and choose appropriately

Learning Outcomes

- Create tuples and use indexing and slicing
- Apply tuple packing and unpacking
- Compare tuples with lists and choose appropriately
- Use tuples in real-world scenarios

Why Tuples?

- Immutable sequences (cannot be changed)

Why Tuples?

- Immutable sequences (cannot be changed)
- Useful for fixed records and safe data

Why Tuples?

- Immutable sequences (cannot be changed)
- Useful for fixed records and safe data
- Can be used as dictionary keys (if elements are hashable)

Why Tuples?

- Immutable sequences (cannot be changed)
- Useful for fixed records and safe data
- Can be used as dictionary keys (if elements are hashable)
- Communicates intent: "this should not change"

Creating Tuples

- Literal: `point = (3, 4)`

Creating Tuples

- Literal: `point = (3, 4)`
- Packing: `t = 1, 2, 3`

Creating Tuples

- Literal: `point = (3, 4)`
- Packing: `t = 1, 2, 3`
- Singleton requires a comma: `single = (5,)`

Packing and Unpacking

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

values = (1, 2, 3, 4)
a, b, *rest = values
```

- Unpacking maps tuple positions to variables

Packing and Unpacking

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

values = (1, 2, 3, 4)
a, b, *rest = values
```

- Unpacking maps tuple positions to variables
- `*rest` collects remaining values

Worked Example: Swap and Multi-Return

```
a, b = 5, 9
a, b = b, a      # swap

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

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

- Tuples are a common way to return multiple values

Tuple Methods and Operations

- Methods: `count`, `index`

Tuple Methods and Operations

- Methods: `count`, `index`
- Supports indexing, slicing, concatenation, repetition

Tuple Methods and Operations

- Methods: `count`, `index`
- Supports indexing, slicing, concatenation, repetition
- Example: `(1,2) + (3,4)`

Tuples vs Lists

Feature	List	Tuple
Mutability	Mutable	Immutable
Syntax	[]	()
Use case	Changeable data	Fixed records
Hashable	No	Yes (if items hashable)

Nested Tuples and Conversion

- Tuples can contain other tuples

```
coords = ((0, 0), (1, 1), (2, 2))
coords_list = list(coords)
coords_tuple = tuple(coords_list)
```

Nested Tuples and Conversion

- Tuples can contain other tuples
- Convert between list and tuple as needed

```
coords = ((0, 0), (1, 1), (2, 2))
coords_list = list(coords)
coords_tuple = tuple(coords_list)
```

Pitfall: Singleton Tuples and "Immutable"

- `(5)` is an integer in parentheses; use `(5,)` for a tuple

```
t = ([1, 2],)       # tuple containing a list
t[0].append(3)      # allowed: the list mutates
```

Pitfall: Singleton Tuples and "Immutable"

- `(5)` is an integer in parentheses; use `(5,)` for a tuple
- Tuples are immutable, but they can contain mutable objects

```
t = ([1, 2],)       # tuple containing a list
t[0].append(3)      # allowed: the list mutates
```

Common Use Cases

- Coordinates and geometry data

Common Use Cases

- Coordinates and geometry data
- Dictionary keys (composite keys)

Common Use Cases

- Coordinates and geometry data
- Dictionary keys (composite keys)
- Returning multiple values from functions

Demo: Distance Between Points

- Store coordinates as tuples

Script: `demo/tuple_distance.py`

Demo: Distance Between Points

- Store coordinates as tuples
- Unpack and compute distance

Script: `demo/tuple_distance.py`

Demo: Distance Between Points

- Store coordinates as tuples
- Unpack and compute distance
- Keep data immutable

Script: `demo/tuple_distance.py`

Demo: Distance Between Points

- Store coordinates as tuples
- Unpack and compute distance
- Keep data immutable
- Extension: compute distances for multiple point pairs

Script: `demo/tuple_distance.py`

Checkpoint 1

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

1. Parentheses alone do not create tuples

Checkpoint 1

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

1. Parentheses alone do not create tuples
2. A trailing comma is required

Checkpoint 1

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

1. Parentheses alone do not create tuples
2. A trailing comma is required
3. `(5)` is just an integer in parentheses

Checkpoint 2

Predict the result:

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

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

Think-Pair-Share

Where in programs should immutability be preferred? Discuss one example from daily software use.

Summary

- Tuples are immutable sequences

Summary

- Tuples are immutable sequences
- Packing/unpacking simplifies assignment and returns

Summary

- Tuples are immutable sequences
- Packing/unpacking simplifies assignment and returns
- Use tuples for fixed records and keys

Summary

- Tuples are immutable sequences
- Packing/unpacking simplifies assignment and returns
- Use tuples for fixed records and keys
- Convert between lists and tuples when needed

Exit Question

Write one line to convert a list `L` to a tuple and back.