

Python Programming

Unit 01 – Lecture 02: Basic Syntax, Comments, Dynamic Typing, Mutability

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

1 Core Concepts

2 Demo

3 Interactive

4 Summary

Learning Outcomes

- Write valid Python blocks using indentation

Learning Outcomes

- Write valid Python blocks using indentation
- Use comments to make code readable

Learning Outcomes

- Write valid Python blocks using indentation
- Use comments to make code readable
- Explain **dynamic typing** and inspect values with `type()`

Learning Outcomes

- Write valid Python blocks using indentation
- Use comments to make code readable
- Explain **dynamic typing** and inspect values with `type()`
- Distinguish **mutable** and **immutable** data types

Syntax Rule #1: Indentation

- Python uses indentation to define blocks (no { })

```
if 10 > 5:
    print("Yes")
    print("Still inside if")
print("Outside if")
```

Syntax Rule #1: Indentation

- Python uses indentation to define blocks (no { })
- A colon : starts a block (if/for/while/def)

```
if 10 > 5:  
    print("Yes")  
    print("Still inside if")  
print("Outside if")
```

Syntax Rule #1: Indentation

- Python uses indentation to define blocks (no { })
- A colon : starts a block (if/for/while/def)
- Consistent indentation matters (use 4 spaces)

```
if 10 > 5:  
    print("Yes")  
    print("Still inside if")  
print("Outside if")
```

Comments

- Single-line comment: # ...

```
rate = 0.08    # annual interest rate (8%)
```

Comments

- Single-line comment: # ...
- Use comments to explain **why**, not obvious **what**

```
rate = 0.08    # annual interest rate (8%)
```

Variables and Assignment

- Variables are created when you assign a value

```
x = 10
x = "ten"          # allowed
print(type(x))    # <class 'str'>
```

Variables and Assignment

- Variables are created when you assign a value
- You can reassign anytime (dynamic typing)

```
x = 10
x = "ten"          # allowed
print(type(x))    # <class 'str'>
```

Dynamic Typing (What it Means)

- The **value** has a type, not the variable name

```
x = 5
print(type(x))    # int
x = 5.0
print(type(x))    # float
```

Dynamic Typing (What it Means)

- The **value** has a type, not the variable name
- `type(x)` checks the current type of the value stored in `x`

```
x = 5
print(type(x))    # int
x = 5.0
print(type(x))    # float
```

Mutable vs Immutable

Type	Mutability
int, float, bool	immutable
str, tuple	immutable
list, dict, set	mutable

- Immutable: cannot change the object in place

Mutable vs Immutable

Type	Mutability
int, float, bool	immutable
str, tuple	immutable
list, dict, set	mutable

- Immutable: cannot change the object in place
- Mutable: can change contents without creating a new object

Example: Immutable String

```
s = "python"  
s2 = s.upper()  
print(s)      # python (original unchanged)  
print(s2)     # PYTHON
```

Example: Mutable List

```
a = [1, 2, 3]
a.append(99)
print(a)    # [1, 2, 3, 99]
```

Pitfall: Aliasing (Same List Object)

```
a = [1, 2, 3]
b = a           # b points to the same list
b.append(10)
print(a)        # [1, 2, 3, 10]
```

- Fix: copy the list using `a.copy()` or `a[:]`

Demo: Dynamic Typing and Mutability

- File: `demo/dynamic_typing_and_mutability.py`

Demo: Dynamic Typing and Mutability

- File: `demo/dynamic_typing_and_mutability.py`
- Shows:

Demo: Dynamic Typing and Mutability

- File: `demo/dynamic_typing_and_mutability.py`
- Shows:
 - reassignment changes type of a variable

Demo: Dynamic Typing and Mutability

- File: `demo/dynamic_typing_and_mutability.py`
- Shows:
 - reassignment changes type of a variable
 - `id()` intuition for objects

Demo: Dynamic Typing and Mutability

- File: `demo/dynamic_typing_and_mutability.py`
- Shows:
 - reassignment changes type of a variable
 - `id()` intuition for objects
 - aliasing and safe copies

Checkpoint 1

Question: Identify the error and fix it.

```
if 5 > 2
    print("OK")
```

Checkpoint 2

Question: Predict the output.

```
a = [10, 20]
b = a
b.append(30)
print(a)
```

Think-Pair-Share

Discuss with a partner:

- Why might immutable objects be safer in programs?
- Give one example where mutation can cause a bug.

Key Takeaways

- Indentation defines blocks; : starts a block

Key Takeaways

- Indentation defines blocks; : starts a block
- Python is dynamically typed (types belong to values)

Key Takeaways

- Indentation defines blocks; `:` starts a block
- Python is dynamically typed (types belong to values)
- Mutable types can change in place; immutable types cannot

Key Takeaways

- Indentation defines blocks; `:` starts a block
- Python is dynamically typed (types belong to values)
- Mutable types can change in place; immutable types cannot
- Aliasing can surprise you; use copies when needed

Exit Question

Name one mutable type and one immutable type in Python.