Python Programming
Unit 01 – Lecture 02 Notes
Basic Syntax, Comments, Dynamic Typing, Mutability

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

In this lecture we build the **core habits** required to write correct and readable Python programs:

- indentation-based blocks,
- meaningful comments and naming,
- dynamic typing and inspecting values using `type()`,
- and the important idea of **mutability**.

2 Core Concepts

2.1 Indentation and Blocks

Python uses indentation to define a block of code. A block starts after a colon `:`.

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

Common mistakes:

- forgetting the colon,
- inconsistent indentation (mixing tabs and spaces),
- extra indentation where it is not needed.

2.2 Comments

Comments are for humans.

- Use comments to explain **why** a choice is made.
- Avoid commenting obvious code.

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

2.3 Variables and Dynamic Typing

In Python, the variable name is just a label. The **value** has the type. That is why you can reassign a variable to a new type:

```
x = 10
print(type(x)) # <class ’int’>
x = "ten"
print(type(x)) # <class ’str’>
```

Practical implication: dynamic typing makes coding fast, but you must be careful when reading user input (because `input()` is always a string).

2.4 `type()` and `id()`

`type(x)` tells you what kind of value is stored in `x`. `id(x)` gives an integer that represents the identity of the object in memory (you can treat it as “object address-like” for learning purposes).

```
a = 100
print(type(a))
print(id(a))
```

2.5 Mutable vs Immutable Types

Immutable: cannot be changed in-place (a new object is created). **Mutable:** can be modified after creation.

Immutable	Mutable
`int`, `float`, `bool`	`list`
`str`, `tuple`	`dict`
	`set`

2.5.1 Immutable Example (String)

```
s = "python"
t = s.upper()
print(s) # python
print(t) # PYTHON
```

`upper()` returns a new string. The original string `s` is unchanged.

2.5.2 Mutable Example (List)

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

`append` changes the same list object in-place.

2.6 Aliasing (A Common Beginner Bug)

Aliasing means two variables refer to the **same** object.

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

Fix: create an independent copy:

```
b = a.copy()
# or: b = a[:]
```

3 Demo Walkthrough

File: `demo/dynamic_typing_and_mutability.py`

What to observe

- The same variable name can store `int`, then `float`, then `str`.
- `id()` helps you notice when a new object was created.
- A list can be mutated, and aliasing can cause unexpected changes.

4 Interactive Checkpoints (with Solutions)

Checkpoint 1 Solution

Question: Identify the error and fix it.

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

Fix: add a colon and keep proper indentation:

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

Checkpoint 2 Solution

Question: Predict the output.

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

Answer: `[10, 20, 30]` because `a` and `b` refer to the same list.

5 Practice Exercises (with Solutions)

Exercise 1: Fix Indentation

Task: Correct the indentation to print `OK` only when the number is positive.

Solution:

```
n = int(input("Enter n: "))
if n > 0:
    print("OK")
```

Exercise 2: Type Checking

Task: Store your age as an integer and print its type.

Solution:

```
age = int(input("Enter age: "))
print(type(age))
```

Exercise 3: Demonstrate Mutability

Task: Create a list, add one item, and print before/after.

Solution:

```
items = [1, 2, 3]
print("Before:", items)
items.append(4)
print("After:", items)
```

Exercise 4: Avoid Aliasing

Task: Copy a list so that changing the copy does not change the original.

Solution:

```python
a = [5, 6, 7]
b = a.copy()
b.append(99)
print("a:", a)
print("b:", b)
```

6 Exit Question (with Solution)

Question: Name one mutable type and one immutable type.

Answer: mutable: `list`; immutable: `str` (many answers are possible).