Python Programming
Unit 01 – Lecture 03 Notes
Tokens, Naming, Strings, and Numeric Types

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

This lecture focuses on the **building blocks** of Python code:

- tokens (keywords, identifiers, literals, operators),
- naming conventions for readable programs,
- strings (methods, operators, formatting),
- and basic numeric data types (`int`, `float`, `complex`).

2 Core Concepts

2.1 Python Tokens

A **token** is a small unit of a program. Common token categories:

- **Keywords:** reserved words such as `if`, `for`, `while`, `def`.
- **Identifiers:** names you create (variables, functions, classes).
- **Literals:** fixed values like `10`, `3.14`, `"hello"`.
- **Operators:** symbols/words that perform operations: `+`, `-`, `*`, `and`, `==`.
- **Delimiters:** punctuation used in syntax: `( ) [ ] { } , :`.

2.2 Identifiers and Naming Conventions

Rules (practical):

- Start with a letter or underscore; remaining characters can include digits.
- Identifiers are case-sensitive: `Name` and `name` are different.
- Do not use keywords as identifiers.

Conventions (recommended):

- Use **snake_case** for variables and functions: `total_marks`, `calculate_cgpa`.
- Choose meaningful names. Prefer `marks` over `m`.

2.3 Strings: Values, Indexing, Slicing

A **string** is a sequence of characters, written with quotes.

```
s1 = "Python"
s2 = 'UPES'
```

Strings support indexing and slicing:

```
s = "python"
s[0] # 'p'
s[-1] # 'n'
s[1:4] # 'yth'
```

Note: strings are **immutable**. Methods usually return a new string.

2.4 Common String Methods

Useful beginner methods:

- `lower()`, `upper()`, `title()`
- `strip()` removes leading/trailing whitespace
- `split()` splits into a list (by spaces by default)

- `replace(old, new)` replaces occurrences
- `find(sub)`, `count(sub)`

```
text = " UPES Dehradun "
clean = text.strip()
print(clean) # "UPES Dehradun"
print(clean.lower()) # "upes dehradun"
print(clean.split()) # ["UPES", "Dehradun"]
```

2.5 String Operators

Strings support some operators:

- **Concatenation** using `+`
- **Repetition** using `*`
- **Membership** using `in`

```
print("py" * 3) # pypypy
print("th" in "python") # True
```

2.6 Formatting Output: `format()` and f-strings

Formatting is important for readable output (grade sheets, reports, logs).

`format()` method

```
name = "Rohit"
cgpa = 7.0
print("Name: {}, CGPA: {:.1f}".format(name, cgpa))
```

`:.1f` means "one digit after the decimal point".

f-strings

f-strings are short and readable:

```
print(f"Name: {name}, CGPA: {cgpa:.1f}")
```

2.7 Numeric Data Types

Main numeric types in this unit:

- `int`: integers (no decimals)
- `float`: real numbers with decimals
- `complex`: numbers like `2+3j`

Type conversions (very common):

```
x = int("10") # 10
y = float("2.5") # 2.5
```

3 Demo Walkthrough

File: `demo/string_formatting_demo.py`

Goal

Read user text, clean extra spaces, compute a small summary, and print a nicely formatted line.

Run

```
python demo/string_formatting_demo.py
```

4 Interactive Checkpoints (with Solutions)

Checkpoint 1 Solution

Question: output of `print("py" * 3)`?
Answer: `pypypy`

Checkpoint 2 Solution

Question: difference between `split()` and `join()`?
Answer:

- `split()` breaks a string into a list.
- `join()` joins a list of strings into a single string.

```
sentence = "Python is fun"
parts = sentence.split() # ["Python", "is", "fun"]
again = "-".join(parts) # "Python-is-fun"
```

5 Practice Exercises (with Solutions)

Exercise 1: Clean a String

Task: Read a string and print it without leading/trailing spaces.
Solution:

```
s = input("Enter text: ")
print(s.strip())
```

Exercise 2: Count Vowels

Task: Count vowels in an input string (a, e, i, o, u; case-insensitive).
Solution:

```
s = input("Enter text: ").lower()
vowels = "aeiou"
count = 0
for ch in s:
    if ch in vowels:
        count += 1
print("Vowel count =", count)
```

Exercise 3: Format to Two Decimals

Task: Print a floating value with 2 decimal places.

Solution (f-string):

```
x = 3.14159
print(f"{x:.2f}")
```

6 Exit Question (with Solution)

Question: Format `3.14159` to 2 decimal places.

Answer: `3.14`. One way:

```
print(f"{3.14159:.2f}")
```