Python Programming

Unit 01 – Lecture 03: Tokens, Naming, Strings, and Numeric Types

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

- Identify Python tokens (keywords, identifiers, literals, operators)

Learning Outcomes

- Identify Python tokens (keywords, identifiers, literals, operators)
- Follow good naming conventions (snake_case)

Learning Outcomes

- Identify Python tokens (keywords, identifiers, literals, operators)
- Follow good naming conventions (snake_case)
- Use common string methods and operators

Learning Outcomes

- Identify Python tokens (keywords, identifiers, literals, operators)
- Follow good naming conventions (snake_case)
- Use common string methods and operators
- Format output using `format()` / f-strings

Learning Outcomes

- Identify Python tokens (keywords, identifiers, literals, operators)
- Follow good naming conventions (snake_case)
- Use common string methods and operators
- Format output using `format()` / f-strings
- Work with numeric types (`int`, `float`, `complex`)

Python Tokens (Big Picture)

- **Keywords**: reserved words (e.g., `if`, `for`, `def`)

Python Tokens (Big Picture)

- **Keywords**: reserved words (e.g., `if`, `for`, `def`)
- **Identifiers**: names you create (variables, functions)

Python Tokens (Big Picture)

- **Keywords**: reserved words (e.g., `if`, `for`, `def`)
- **Identifiers**: names you create (variables, functions)
- **Literals**: fixed values (`10`, `3.14`, `"hi"`)

Python Tokens (Big Picture)

- **Keywords**: reserved words (e.g., `if`, `for`, `def`)
- **Identifiers**: names you create (variables, functions)
- **Literals**: fixed values (`10`, `3.14`, `"hi"`)
- **Operators**: `+ - * / == <` etc.

Python Tokens (Big Picture)

- **Keywords**: reserved words (e.g., `if`, `for`, `def`)
- **Identifiers**: names you create (variables, functions)
- **Literals**: fixed values (`10`, `3.14`, `"hi"`)
- **Operators**: `+ - * / == <` etc.
- **Delimiters**: `( ) [ ] { } , :`

Identifiers and Naming Conventions

- Use **snake_case**: `total_marks`, `student_name`

Identifiers and Naming Conventions

- Use **snake_case**: `total_marks`, `student_name`
- Names should be meaningful and consistent

Identifiers and Naming Conventions

- Use **snake_case**: `total_marks`, `student_name`
- Names should be meaningful and consistent
- Do not use keywords as names (e.g., `for = 5` is invalid)

String Basics

- Strings are sequences of characters (immutable)

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

String Basics

- Strings are sequences of characters (immutable)
- Indexing and slicing works like lists

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

Common String Methods

- `lower()`, `upper()`, `title()`

Common String Methods

- `lower()`, `upper()`, `title()`
- `strip()` (remove leading/trailing spaces)

Common String Methods

- `lower()`, `upper()`, `title()`
- `strip()` (remove leading/trailing spaces)
- `split()` (string $\rightarrow$ list of words)

Common String Methods

- `lower()`, `upper()`, `title()`
- `strip()` (remove leading/trailing spaces)
- `split()` (string $\rightarrow$ list of words)
- `replace(old, new)`

Common String Methods

- `lower()`, `upper()`, `title()`
- `strip()` (remove leading/trailing spaces)
- `split()` (string $\rightarrow$ list of words)
- `replace(old, new)`
- `find(sub)` / `count(sub)`

String Operators

- Concatenation: `"Py" + "thon"` $\rightarrow$ `"Python"`

String Operators

- Concatenation: `"Py" + "thon"` $\rightarrow$ `"Python"`
- Repetition: `"py" * 3` $\rightarrow$ `"pypypy"`

String Operators

- Concatenation: `"Py" + "thon"` → `"Python"`
- Repetition: `"py" * 3` → `"pypypy"`
- Membership: `"th" in "python"` → `True`

Formatting Output

- `format()` method:

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


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

Formatting Output

- `format()` method:

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

- f-strings (recommended for readability):

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

Numeric Data Types

- `int`: integers (e.g., `5`, `-12`)

Numeric Data Types

- `int`: integers (e.g., `5`, `-12`)
- `float`: decimals (e.g., `3.14`)

Numeric Data Types

- `int`: integers (e.g., `5`, `-12`)
- `float`: decimals (e.g., `3.14`)
- `complex`: complex numbers (e.g., `2+3j`)

Numeric Data Types

- `int`: integers (e.g., `5`, `-12`)
- `float`: decimals (e.g., `3.14`)
- `complex`: complex numbers (e.g., `2+3j`)
- Convert types when needed: `int("10")`, `float("2.5")`

Demo: Cleaning and Formatting Strings

- File: `demo/string_formatting_demo.py`

Demo: Cleaning and Formatting Strings

- File: `demo/string_formatting_demo.py`
- Reads a name and marks, cleans extra spaces

Demo: Cleaning and Formatting Strings

- File: `demo/string_formatting_demo.py`
- Reads a name and marks, cleans extra spaces
- Prints a formatted student summary line

Checkpoint 1

Question: What is the output?

```
print("py" * 3)
```

Checkpoint 2

Question: What is the difference between `split()` and `join()`?

```
sentence = "Python is fun"
parts = sentence.split()
again = "-".join(parts)
```

Think-Pair-Share

You have input text with extra spaces:

- `" UPES Dehradun "`

Discuss: which methods would you use to clean it?

Key Takeaways

- Tokens build programs: keywords, identifiers, literals, operators

Key Takeaways

- Tokens build programs: keywords, identifiers, literals, operators
- Use clean naming: snake_case, meaningful names

Key Takeaways

- Tokens build programs: keywords, identifiers, literals, operators
- Use clean naming: snake_case, meaningful names
- Strings are immutable; methods return new strings

Key Takeaways

- Tokens build programs: keywords, identifiers, literals, operators
- Use clean naming: snake_case, meaningful names
- Strings are immutable; methods return new strings
- Use `split/join/strip` for text processing

Key Takeaways

- Tokens build programs: keywords, identifiers, literals, operators
- Use clean naming: snake_case, meaningful names
- Strings are immutable; methods return new strings
- Use `split/join/strip` for text processing
- f-strings and `format()` produce clean output

Exit Question

Format `3.14159` to **2 decimal places**.