# SIG Python Class Notes

---

**Lecture 1: Introduction to Python Programming**

Python is one of the most popular and versatile programming languages used in the tech industry today. It is widely appreciated for its clean syntax, ease of readability, and the fact that it supports multiple programming paradigms including procedural, object-oriented, and functional programming. The primary goal of this session was to provide students with a foundational understanding of the Python language and set the stage for further development in the domain of programming and problem-solving.

**Topics Covered:**

- **What is Python?**
  Python is a high-level, interpreted programming language designed by Guido van Rossum and released in 1991. Its syntax is inspired by English, making it highly readable and beginner-friendly. It is open-source and available on all major platforms (Windows, macOS, Linux).
- **Why Learn Python?**
  Python is used in web development (Django, Flask), data science (pandas, NumPy), artificial intelligence (TensorFlow, scikit-learn), scripting, automation, game development, and more. It is the first choice for both beginners and professionals.
- **Installing Python and Setting up an IDE:**
  - Download and install Python from python.org
  - Recommended IDEs: **Visual Studio Code**, **Jupyter Notebook**, **PyCharm**, or **Thonny** for beginners
  - Overview of the interactive shell and script mode
- **Basic Building Blocks of Python:**
  - **Keywords:** Reserved words like if, else, for, def, class, etc.

    - **Identifiers:** Names given to variables, functions, etc. Must start with a letter or underscore
    - **Variables and Dynamic Typing:** Variables can hold data without declaring their type. Python handles memory and data types automatically.
- **Input and Output Functions:**
    - input() is used to take input from the user in the form of a string.
    - print() is used to display output on the screen. It can also format strings using f-strings or .format().
- **Basic Data Types in Python:**
    - **int** (e.g., 5, 100, -32)
    - **float** (e.g., 3.14, -0.1, 2.0)
    - **str** (e.g., "Hello", 'Python')
    - **bool** (e.g., True, False)
- **Type Conversion & Type Checking:**
    - Conversion using functions: int(), float(), str(), bool()
    - Checking data type: type(variable_name)

---

**Lecture 2: Operators and Control Flow Statements**

This session explored how to make decisions and repeat actions in Python programs. We introduced the concept of operators and how they are used in arithmetic, comparisons, and logical operations. Students also learned about conditional execution using if, elif, and else statements and how to implement repetitive logic through loops.

**Operators in Python:**

- **Arithmetic Operators:** Used for basic mathematical operations
  +, -, *, /, //, %, **
  - / results in float division
  - // results in integer (floor) division
  - ** is used for exponentiation
- **Relational (Comparison) Operators:**
  ==, !=, >, <, >=, <=
  These operators return True or False based on comparison results.
- **Logical Operators:**
  and, or, not
  These work with boolean values and combine multiple conditions.
- **Assignment Operators:**
  =, +=, -=, *=, /=, //=, **=, %=
  These assign values to variables and update them simultaneously.

**Conditional Statements:**

- Python uses if, elif, and else to handle decision-making logic.
- The syntax is indentation-based and does not use braces.
- Nested conditions are supported for multiple levels of checks.
  Example:

```
if a > b:
   print("a is greater")
elif a == b:
   print("a and b are equal")
else:
   print("b is greater")
```

**Loops:**

- **for loop:**
  Commonly used with range() to iterate over sequences and numbers.
  Example: for i in range(5):
- **while loop:**
  Repeats a block of code as long as a condition is True.
  Used when the number of iterations is unknown in advance.
- **Loop Control Statements:**
  - break: Terminates the loop immediately
  - continue: Skips the current iteration and goes to the next
  - pass: Acts as a placeholder, does nothing

---

**Lecture 3: Python Collections and Built-in Data Types**

This class focused on four essential built-in data types used to store multiple values in Python: List, Tuple, Set, and Dictionary. Each of these data types has unique properties and use-cases. Students learned how to create them, access elements, and use associated built-in methods.

**List**

- **Definition:** Ordered, mutable collection of elements enclosed in square brackets [].
- **Use Case:** Best when order matters and the content might change.

**Important Methods:**
append(), extend(), insert(), remove(), pop(), clear(), sort(), reverse(), index(), count()

**Tuple**

- **Definition:** Ordered, immutable collection enclosed in parentheses ().
- **Use Case:** When the data should not be modified (e.g., fixed values, coordinates).

**Important Methods:**
count(), index()

**Set**

- **Definition:** Unordered collection of unique, mutable elements enclosed in {}.
- **Use Case:** Useful for membership testing and removing duplicates.

**Important Methods:**
add(), update(), remove(), discard(), clear(), union(), intersection(), difference()

**Key Difference – remove() vs discard():**

- remove(x): Raises KeyError if x is not found
- discard(x): Does not raise an error if x is not present

**Dictionary**

- **Definition:** Unordered collection of key-value pairs, enclosed in {}. Keys must be unique and immutable.

**Common Methods:**
keys(), values(), items(), get(), update(), pop(), clear()

Example:

person = {'name': 'Alice', 'age': 25}

print(person.get('name'))  # Output: Alice

---

**Important Terms to Define When Teaching Data Types in Python**

1. **Mutable**
    - **Definition: Objects whose values can be changed after creation.**
    - **Examples: List, Set, Dictionary**
2. **Immutable**
    - **Definition: Objects whose values cannot be changed after creation.**
    - **Examples: Tuple, String**
3. **Ordered**
    - **Definition: Elements maintain the order in which they were inserted.**
    - **Examples: List, Tuple, Dictionary (from Python 3.7+)**
4. **Unordered**
    - **Definition: Elements are not stored in any particular order.**
    - **Example: Set**
5. **Indexed**

- **Definition: Elements can be accessed using an index (starting from 0).**
- **Examples: List, Tuple, String**

6. **Non-Indexed**
   - **Definition: Elements cannot be accessed by position; accessed via keys or not at all.**
   - **Examples: Set, Dictionary (accessed via keys)**
7. **Iterable**
   - **Definition: An object that can be looped over (used in a for-loop).**
   - **Examples: All major collections in Python (List, Tuple, Set, Dictionary, String)**
8. **Hashable**
   - **Definition: An object with a fixed hash value throughout its lifetime (used as keys in dictionaries or elements in sets).**
   - **Examples: String, Tuple (if elements are immutable)**
9. **Homogeneous (in some languages) vs. Heterogeneous**
   - **Not strictly enforced in Python, but:**
     - **Homogeneous: All elements are of the same type.**
     - **Heterogeneous: Mixed-type elements (common in Python Lists and Tuples)**
10. **Dynamic Typing**
    - **Definition: Variable types are determined at runtime.**
    - **Python allows this, making it flexible but also potentially error-prone.**

---