Class One Theory

- **Python** is high-level programming language, dynamic and interpreter language
- **Fewer Lines of Code**: Compared to languages like C, C++, or Java, Python can accomplish tasks in fewer lines, boosting productivity.
- **Interpreter System**: Python runs code line by line (interactive).

Basic Python Syntax

In Python, “syntax” refers to the set of rules that define how Python programs are written and interpreted. While some of these rules are common across programming languages

Statements and the Interactive Shell

Statements

- A **statement** in Python is a piece of code that performs a specific action. Examples include assignment statements, function calls, loops, and conditional statements.

```
x = 5             # Assignment statement
print(x)          # Function call statement
if x > 3:         # If statement
    print("Yes!")
```

- Python executes statements **one by one** from top to bottom, unless control flow (e.g., `if`, `for`, `while`, or function calls) changes that order.

The Interactive Shell

- When you run `python` (or `py`) without specifying a script file, you enter the **interactive shell** (REPL: Read-Eval-Print Loop).
- You can type statements or expressions, and Python immediately evaluates or executes them, then displays the result (if any):

```
>>> x = 10
>>> x * 2
```

```
20
```

- To exit the shell, type `exit()` or press `Ctrl+D` (on Linux/Mac) or `Ctrl+Z` then `Enter` (on Windows).

Indentation and Code Blocks

Significance of Indentation

- Unlike many languages that use curly braces `{}` or keywords (like `BEGIN` / `END`) to define code blocks, Python uses **indentation** (whitespace at the beginning of a line).
- Every block of code that belongs together (e.g., the body of a function, the body of an `if` statement) must be indented by the same amount of whitespace.
- **Consistency is Key**: While Python doesn't force you to use 2 or 4 spaces, the widely accepted standard (PEP 8) is to use **4 spaces** per indentation level.

Basic Example

```
if 5 > 2:
    print("Five is greater than two!")
    print("This line is also in the same block.")
print("This line is outside the if block, no indentation.")
```

- The two `print` statements are indented by 4 spaces, so they belong to the `if` block.
- The last `print` statement is not indented, so it's executed regardless of the `if` condition.

Indentation Errors

- If you use inconsistent indentation within the same block, Python raises an `IndentationError` .

```
if 5 > 2:
  print("One level of indentation")
    print("Inconsistent indentation, will cause an error!")
```

- Always maintain the same number of spaces in a block.

Line Structure, Multiple Statements, and Continuations

End of a Statement

- In Python, each **new line** generally ends a statement. No semicolons are required (unlike many other languages).
- **Optional**: You can use a semicolon (`;`) to separate statements on the same line, but this is **not** recommended:

```
x = 5; y = 10; print(x + y)  # Allowed, but not typical in Python.
```

- For cleaner code, prefer one statement per line.

Explicit Line Continuation

- If a statement is too long to fit on one line, you can use the **backslash** (`\`) to continue on the next line:

```
total_sum = 1 + 2 + 3 + 4 + \
            5 + 6 + 7 + 8
```

- However, Python also supports **implicit line continuation** inside parentheses, brackets, and braces:

```
total_sum = (
    1 + 2 + 3 +
    4 + 5 + 6
)
# No backslash needed because we are inside parentheses.
```

Blank Lines

- Blank lines are typically **ignored** by Python but are used to improve **readability**.
- Use blank lines to separate logical sections of your code.

6.4. Comments and Documentation

6.4.1. Single-Line Comments

- Single-line comments start with a `#` symbol, and Python ignores the rest of the line:

```
# This is a comment
x = 10  # This is an inline comment
```

Multi-Line Comments or Docstrings

- Python doesn't have a dedicated multi-line comment syntax like `/* */` in C.
- A common workaround is to use **triple-quoted strings** (`"""`) that are **not** assigned to a variable, effectively acting as a comment:

```
"""
This section of text
can be used as a multi-line comment
if it's not assigned to a variable.
"""
```

- For **documentation** (docstrings), triple-quoted strings **assigned** to a function, class, or module are used to describe what that block of code does:

```
def my_function():
    """
    This is a docstring. It should describe what my_function does.
    """
    print("Inside my_function")
```

- Python's built-in help system can read these docstrings when you use the `help()` function.

Variables, Identifiers, and Keywords

Identifiers

- An **identifier** is a name used to identify variables, functions, classes, modules, etc.
- **Rules**:
    1. Must start with a letter (A-Z or a-z) or an underscore `_` .
    2. Can contain letters, digits (0-9), or underscores.
    3. Case-sensitive: `age` ≠ `Age` .
    4. Avoid using Python **keywords** as identifiers.

Keywords

- Python reserves certain words for its own use. These **keywords** cannot be used as variable or function names:

```
False, True, None, and, as, assert, async, await, break, class, continue,
def, del, elif, else, except, finally, for, from, global, if, import, in,
is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield
```

- Python's keyword list can vary slightly by version, so to see which are current on your system:

```
import keyword
print(keyword.kwlist)
```

Variable Naming Conventions

- **PEP 8** guidelines recommend using **snake_case** for variable names:

```
my_var = 10
user_name = "Alice"
```

- For constants (values not supposed to change), use ALL_CAPS:

```
MAX_SIZE = 100
```

- Use names that are **descriptive** of what the variable holds.

Expressions, Operators, and Operands

Expressions

- An **expression** is any code segment that returns a value. It can contain variables, operators, literal values, and function calls.

```
2 + 3          # Expression => 5
x = 5
x * 2          # Expression => 10
```

Common Operators

1. **Arithmetic Operators**: `+`, `-`, `*`, `/`, `//` (floor division), `%` (modulus), `**` (exponent).

2. **Comparison Operators**: `==` , `!=` , `>` , `<` , `>=` , `<=` .
3. **Logical Operators**: `and` , `or` , `not` .
4. **Assignment Operators**: `=` , `+=` , `-=` , `*=` , `/=` , etc.
5. **Membership Operators**: `in` , `not in` .
6. **Identity Operators**: `is` , `is not` .

Printing and Getting User Input

Printing to the Console

- `print()` is a built-in function to output data to the console:

```
print("Hello, World!")
x = 5
print(x)
```

- By default, `print()` adds a newline at the end. You can change this behavior:

```
print("No newline here", end="")
```

- You can also print multiple items, separated by a space:

```
print("X =", x, "and Y =", y)
```

Reading Input

- Python uses the `input()` function to read user input from the console:

```
name = input("Enter your name: ")
print("Hello,", name)
```

- `input()` always returns a **string**; you may need to convert it to an integer or float if doing numeric operations.