(1) Introduction to Python

1. What is Python and Why Use It?

Python is an easy-to-use, high-level language that runs code line by line.

- **Easy Syntax**: Looks like regular English.
- **High-level**: You don't need to manage memory manually.
- **Interpreted**: Executes one line at a time — easier to find and fix bugs.

Key Benefits:

- Simple to learn
- Works on all operating systems (Windows, macOS, Linux)
- Free and open-source
- Strong online community
- Built-in tools for math, web, files, etc.
- Widely used in web apps, automation, data science, AI, etc.

2. Python's Background

- **Inventor**: Guido van Rossum
- **Released**: 1991
- **Inspired by**: ABC language and a comedy show ("Monty Python")
- **Versions**:
    - Python 2 – old and no longer supported
    - Python 3 – current and recommended

Python has become one of the top languages worldwide.

3. Why Choose Python?

- Clear and clean code
- Fewer lines than Java or C++
- Powerful libraries for all tasks (web, ML, GUI, etc.)
- Write once, run anywhere
- Great for beginners and experts
- Can be used for many purposes (websites, games, AI, etc.)

4. Getting Python Ready on Your Computer

- Download from: python.org
- After installation, check version with `python --version`

Popular Code Editors (IDEs):

1. **VS Code** – Easy to use and extend
2. **PyCharm** – Full-featured Python IDE
3. **Anaconda** – Great for data science, includes Jupyter

5. Writing Your First Program

1. Open your code editor
2. Type:

```
print("Hello, World!")
```

3. Save as `hello.py`
4. Run it from the terminal with:

```
python hello.py
```

---

(2) Writing Code Properly – Python Style (PEP 8)

PEP 8 Overview

PEP 8 is Python's official style guide to make code neat and readable.

Why Follow It?

- Clean-looking code
- Better teamwork
- Easy to understand

Key Style Rules:

- Use 4 spaces for indentation
- Keep lines short (under 79 characters)
- Use blank lines to split sections
- Name variables clearly (`student_name` not `sn`)
- Add spaces around operators: `x = y + z`
- Write helpful comments

Indentation and Comments

- Python uses **indentation** instead of curly braces:

```
if age > 18:
    print("Adult")
```

- **Comments**:
  - Single-line: `# comment here`
  - Inline: `print("Hi") # Greet the user`
  - Multi-line (rare): triple quotes

Naming Rules

Type	Rule	Example
Variable	lowercase_underscore	`user_name`
Function	lowercase_underscore	`get_total()`
Class	PascalCase	`StudentData`
Constant	ALL_CAPS	`PI = 3.14`
Private Var	Start with _	`_hidden_value`

Tips for Better Code

1. Follow PEP 8
2. Use descriptive names
3. Add comments where needed
4. Break code into functions
5. Don't repeat code (DRY principle)
6. Keep lines short and organized

Example:

```
def circle_area(radius):

    """Returns area of a circle"""
    PI = 3.14
    return PI * radius * radius

r = 5
print("Area:", circle_area(r))
```

---

(3) Core Python Concepts

Data Types in Python

- `int`: Whole numbers — `x = 5`
- `float`: Decimal numbers — `y = 3.14`
- `str`: Text — `name = "Alice"`
- `list`: Changeable list — `fruits = ["apple", "mango"]`
- `tuple`: Fixed list — `colors = ("red", "blue")`
- `dict`: Key-value pairs — `{"name": "Bob", "age": 20}`
- `set`: Unique items — `{1, 2, 3}`

Variables and Memory

- Variables are labels for data in memory.

```
name = "Alice"
age = 20
```

Python auto-assigns data types and handles memory for you.

Operators in Python

Arithmetic Operators

- `+`, `-`, `*`, `/`, `//`, `%`, `**`

Comparison Operators

- `==`, `!=`, `<`, `>`, `<=`, `>=`

Logical Operators

- `and`, `or`, `not`

Bitwise Operators (for bits, rarely used):

- `&`, `|`, `^`, `~`, `<<`, `>>`

---

(4) Conditional Statements

- `if`: Run code if a condition is true
- `if-else`: Either this or that
- `if-elif-else`: Check multiple conditions
- **Nested if**: `if` inside another `if`

Example:

```
age = 17
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

---

(5) Loops in Python

Types of Loops:

- `for`: Loop through items (like list, string)
- `while`: Run code while a condition is true

Control Flow in Loops:

- `break`: Stop the loop early
- `continue`: Skip current step
- `pass`: Placeholder (do nothing)

Example:

```
for fruit in ["apple", "banana"]:
```

```
    print(fruit)
```

---

(6) Generators and Iterators

Generators:

- Special functions using `yield`
- Remember their state between calls
- Save memory by generating values one at a time

```
def count_up():
    for i in range(3):
        yield i
```

Iterators:

- Objects you can loop over
- Use `__iter__()` and `__next__()` methods
- Lists, strings, sets are all iterable

---

(7) Functions and Methods

Defining a Function:

```
def greet(name):
    print("Hello", name)
```

Function Parameters:

- Positional: `greet("Alice")`
- Keyword: `greet(name="Alice")`
- Default: `def greet(name="Guest")`

Variable Scope:

- **Local**: Inside functions
- **Global**: Outside functions

Built-in Methods:

- `str`: `upper()`, `lower()`, `replace()`
- `list`: `append()`, `sort()`, `pop()`

---

(8) Loop Control Statements

- `break`: Ends the loop completely
- `continue`: Skips current round
- `pass`: Does nothing (used as placeholder

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

---

(9) String Handling

- Strings are collections of characters.
- You can:
    - **Join** strings: `"Hello " + "World"`
    - **Repeat** strings: `"ha" * 3`

Common Methods:

- `upper()`, `lower()`
- `strip()`
- `replace()`
- `find()`
- `split()`

Slicing:

```
s = "Python"
print(s[0:3])  # Output: Pyt
```

---

(10) Advanced Python (Functional Programming)

Functional Concepts:

- Functions can be treated like values.
- You can pass them around like data.

Useful Functions:

- `map()`: Apply function to every item
- `filter()`: Keep only items that meet a condition
- `reduce()`: Combine all items into one (needs `functools`)

Closures:

A function inside another function that remembers outer variables.

Decorators:

A function that adds features to another function without changing its code.

```
@my_decorator
def my_func():
    pass
```
