Python Programming Fundamentals – Extended Edition (V2)

1. Introduction to Python

Python is a high-level, interpreted language focused on readability and productivity. It supports multiple paradigms including procedural, object-oriented, and functional programming.

2. Variables and Data Types

Python uses dynamic typing.

Common data types include:

- int, float, str, bool

- list, tuple, set, dict

- bytes, bytearray

- complex numbers

Python also supports custom data structures via classes.

3. Operators

Python supports arithmetic, comparison, logical, bitwise, membership, and identity operators.

Examples:

- Arithmetic: +, -, *, /, %, **

- Logical: and, or, not

- Membership: in, not in

4. Control Flow

Python uses indentation for code blocks.

Statements include:

- if/elif/else

- for loops

- while loops

- break, continue, pass

5. Functions (Extended)

Functions are defined with `def`.

They support:

- Positional parameters

- Keyword parameters

- Default parameters

- *args for variable-length arguments

- **kwargs for keyword argument packing

Python also supports higher-order functions and closures.

6. File Handling (Extended)

Python interacts with files using the `open()` function.

Modes:

- 'r' (read), 'w' (write), 'a' (append)

- 'rb', 'wb' for binary access

Always use context managers:

with open("data.txt", "r") as f:

content = f.read()

7. Error and Exception Handling

Errors are managed using try-except blocks.

You can handle multiple exceptions, raise custom exceptions, and use finally blocks for cleanup.

8. Object-Oriented Programming (Extended)

Python supports classes, objects, inheritance, polymorphism, abstraction, and encapsulation.

Example:

class Person:

def __init__(self, name):

self.name = name

def greet(self):

print(f"Hello, {self.name}")

Additional OOP features:

- Class methods and static methods

- Private and protected attributes

- dataclasses for boilerplate reduction

9. Modules and Packages (Extended)

A module is a Python file, and a package is a directory containing modules.

Python’s import system supports:

- Built-in modules

- Third-party modules (installed using pip)

- Custom modules

Common modules:

os, sys, math, datetime, json, random, pathlib

10. Decorators (Advanced)

Decorators modify or enhance the functionality of functions.

Example:

def log(func):

def wrapper():

print("Calling function...")

func()

return wrapper

@log

def greet():

print("Hello")

11. Generators and Iterators

Generators use `yield` to produce values lazily.

Useful for memory-efficient data processing.

12. List Comprehensions and Generator Expressions

Python provides concise syntax for creating lists, sets, and dictionaries.

Example:

squares = [x*x for x in range(10)]

13. Virtual Environments

Virtual environments isolate project dependencies.

Commands:

python -m venv env

env/Scripts/activate (Windows)

14. Concurrency (Threading, Multiprocessing, AsyncIO)

Python supports three concurrency models:

- Threading: lightweight, limited by GIL for CPU tasks

- Multiprocessing: true parallelism

- AsyncIO: best for I/O-bound tasks

15. Python Standard Library Deep Dive

Key modules:

- collections: Counter, defaultdict, deque

- itertools: combinations, permutations, infinite iterators

- functools: lru_cache, partial, reduce

- logging: structured logging

16. JSON and Data Parsing

Python provides JSON encoding/decoding using json module.

Example:

data = json.loads('{"a":1}')

json_str = json.dumps(data)

17. Working with APIs (Basics)

Use requests library to send HTTP requests.

Example:

import requests

response = requests.get("https://api.example.com")

18. Conclusion

Python is a powerful, flexible, and beginner-friendly language. Its vast ecosystem makes it suitable for web development, automation, data science, machine learning, and backend systems.