Python Basics

1. OOPS Concept

Object Oriented Programming is a fundamental concept in Python.

- Class in Python
- Objects in Python
- Polymorphism in Python
- Encapsulation in Python
- Inheritance in Python
- Data Abstraction in Python

a) Class

A class is a collection of objects. Classes are blueprints for creating objects. A class defines a set of attributes and methods that the created objects (instances) can have.

b) Objects

An objects represents an real world entities. It represents a specific implementation of the class and holds its own data.

c) Polymorphism

Polymorphism allows methods to have the same name but behave differently based on the object's context.

d) Encapsulation

Encapsulation is the bundling of data (attributes) and methods (functions) within a class, restricting access to some components to control interactions.

e) Inheritance

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (called a child or derived class) to inherit attributes and methods from another class (called a parent or base class).

f) Abstraction

Abstraction hides the internal implementation details while exposing only the necessary functionality. It helps focus on "what to do" rather than "how to do it."

2. Exception Handing

Python Exception Handling handles errors that occur during the execution of a program. Exception handling allows to respond to the error, instead of crashing the running program.

- **try Block**: try block lets us test a block of code for errors. Python will "try" to execute the code in this block. If an exception occurs, execution will immediately jump to the except block.
- **except Block:** except block enables us to handle the error or exception. If the code inside the try block throws an error, Python jumps to the except block and executes it. We can handle specific exceptions or use a general except to catch all exceptions.
- **else Block:** else block is optional and if included, must follow all except blocks. The else block runs only if no exceptions are raised in the try block. This is useful for code that should execute if the try block succeeds.
- **finally Block:** finally block always runs, regardless of whether an exception occurred or not. It is typically used for cleanup operations (closing files, releasing resources).

Difference Between Exception and Error

- **Error**: Errors are serious issues that a program should not try to handle. They are usually problems in the code's logic or configuration and need to be fixed by the programmer. Examples include syntax errors and memory errors.
- **Exception**: Exceptions are less severe than errors and can be handled by the program. They occur due to situations like invalid input, missing files or network issues.

Logging in Python

- Logging in Python is used to track events (like errors or important actions) in a program, making it easier to debug and monitor applications.

```
import logging

logging.basicConfig(level=logging.DEBUG)
logging.debug("This is a debug message.")
logging.info("This is an info message.")
logging.warning("This is a warning message.")
logging.error("This is an error message.")
logging.critical("This is a critical message.")
```

Output:

```
DEBUG:root:This is a debug message.
INFO:root:This is an info message.
WARNING:root:This is a warning message.
ERROR:root:This is an error message.
CRITICAL:root:This is a critical message.
```

3. **Multi-threding**

Multithreading is defined as the ability of a processor to execute multiple threads concurrently. In a simple, single-core CPU, it is achieved using frequent switching between threads. This is termed **context switching.** In context switching, the state of a thread is saved and the state of another thread is loaded whenever any interrupt (due to I/O or manually set) takes place. Context switching takes place so frequently that all the threads appear to be running parallelly (this is termed **multitasking)**.

4. **Coding Standard**

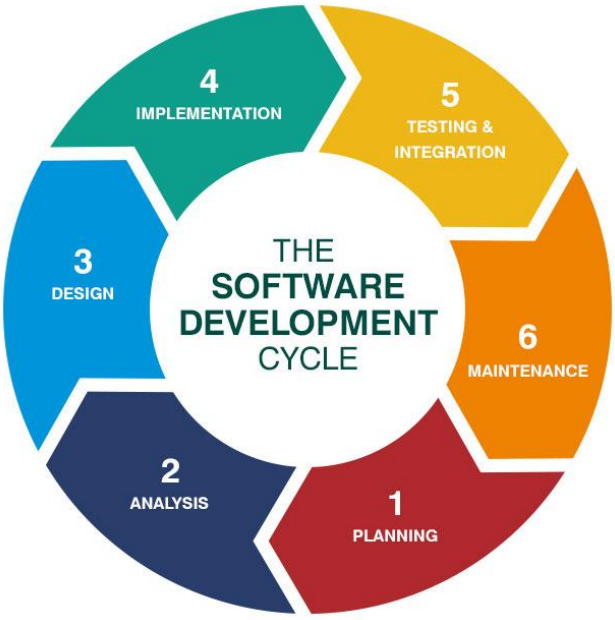
PEP 8

It is python enhancement proposal that provides style guidelines for writing clean and readable code in python.

Type	Naming Convention	Examples
Function	Use a lowercase word or words. Separate words by underscores to improve readability. This style is called snake case.	`function`, `python_function`
Variable	Use a lowercase single letter, word, or words. Separate words with underscores to improve readability.	`x`, `var`, `python_variable`
Class	Start each word with a capital letter. Don't separate words with underscores. This style is called camel case or Pascal case.	`Model`, `PythonClass`
Method	Use a lowercase word or words. Separate words with underscores to improve readability (snake case).	`class_method`, `method`
Constant	Use an uppercase single letter, word, or words. Separate words with underscores to improve readability.	`CONSTANT`, `PYTHON_CONSTANT`, `PYTHON_LONG_CONSTANT`
Module	Use a short, lowercase word or words. Separate words with underscores to improve readability.	`module.py`, `python_module.py`
Package	Use a short, lowercase word or words. Don't separate words with underscores.	`package`, `pythonpackage`

5. Software Development Life Cycle (SDLC)

SDLC is a process followed for software building within a software organization. The Software Development Life Cycle (SDLC) consists of seven essential phases: Planning, Requirements Analysis, Design, Coding, Testing, Deployment, and Maintenance, that ensure successful software development, from concept to continuous improvement.



Agile Model:

The meaning of Agile is swift or versatile. “Agile process model" refers to a software development approach based on iterative development. **Agile methods break tasks into smaller iterations, or parts** do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Phases of Agile Model:
Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.
2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.
3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.
4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.
5. Deployment: In this phase, the team issues a product for the user's work environment.
6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Scrum
SCRUM is an agile framework focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:
Product Owner – Responsible for defining what needs to be built and prioritizing the tasks (from the Product Backlog).
Scrum Master – Helps the team follow Scrum practices, removes obstacles, and ensures the team stays focused.
Development Team – The people who actually build and deliver the product (e.g., developers, designers, testers).

MOSCOW Principles

The **MoSCoW Principle** is a prioritization technique used in project management, particularly in Agile, to help stakeholders and teams decide what to focus on in terms of requirements or features.
MoSCoW stands for:

1. **Must Have** – These are the essential features or requirements that **must** be delivered for the project to be considered a success. If these are not included, the project will fail.
2. **Should Have** – These are important but not critical features. They are **highly desirable** and add significant value, but the project can still succeed without them if necessary.

3. **Could Have** – These are nice-to-have features that would enhance the project but are not essential. They can be **delayed** or skipped if time or resources are limited.
4. **Won't Have (This Time)** – These are features that are considered low priority or out of scope for the current project cycle. They might be reconsidered in the future.