**Python Learning Document**

**1. Data Types in Python**

Python provides several built-in data types that help in storing and managing different kinds of values. Understanding data types is fundamental for programming in Python.

• Numeric types: int, float, complex

• Sequence types: list, tuple, range

• Text type: str

• Set types: set, frozenset

• Mapping type: dict

• Boolean type: bool

• Binary types: bytes, bytearray, memoryview

Example:

x = 10 # int  
y = 3.14 # float  
z = "Hello" # string  
nums = [1,2,3] # list  
coords = (10,20) # tuple  
data = {"name":"Alice", "age":25} # dictionary  
flag = True # boolean

**2. Object-Oriented Programming (OOP) in Python**

Python supports object-oriented programming, which organizes code into classes and objects. OOP promotes modularity, reusability, and abstraction. The four main pillars of OOP in Python are encapsulation, inheritance, polymorphism, and abstraction.

**3. Constructors in Python (\_\_init\_\_)**

In Python, constructors are defined using the \_\_init\_\_() method. A constructor initializes the attributes of an object when it is created. The self keyword is used to refer to the instance of the class.

Example:

class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
s1 = Student("John", 21)  
print(s1.name, s1.age)

**4. Inheritance in Python**

Inheritance allows a class to acquire the properties and methods of another class. The class that inherits is called the child class, while the class being inherited from is called the parent class.

Example:

class Animal:  
 def speak(self):  
 print("This is an animal")  
  
class Dog(Animal):  
 def speak(self):  
 print("Bark")  
  
d = Dog()  
d.speak()

**5. Abstraction in Python**

Abstraction is the concept of hiding implementation details and showing only the functionality. In Python, abstraction is achieved using abstract base classes (ABC) from the abc module.

Example:

from abc import ABC, abstractmethod  
  
class Vehicle(ABC):  
 @abstractmethod  
 def start(self):  
 pass  
  
class Car(Vehicle):  
 def start(self):  
 print("Car started")  
  
c = Car()  
c.start()

**6. Polymorphism in Python**

Polymorphism allows the same function or method to behave differently based on the object that calls it. It enables code reusability and flexibility.

Example:

class Bird:  
 def fly(self):  
 print("Most birds can fly")  
  
class Penguin(Bird):  
 def fly(self):  
 print("Penguins cannot fly")  
  
b1 = Bird()  
p1 = Penguin()  
b1.fly()  
p1.fly()

**7. Importance of OOP in Python**

OOP in Python makes programming more structured and easier to manage, especially for large applications. By using classes and objects, developers can model real-world entities, reuse code effectively, and build maintainable systems.

• Encapsulation protects data from unauthorized access.

• Inheritance promotes code reuse and establishes relationships.

• Polymorphism improves flexibility in method usage.

• Abstraction simplifies complex systems by exposing only necessary details.

**8. Practical Applications of Python OOP**

OOP is widely used in real-world Python projects across different domains:

• Web development frameworks like Django and Flask use OOP concepts for handling models and views.

• Game development relies on OOP to represent characters, objects, and interactions.

• Data science projects use classes to structure datasets and models.

• Automation scripts benefit from OOP by making code modular and reusable.