# ****What is OOP?****

**"OOP ka sabse bada faida yeh hai ke yeh aapko apna code chhote chhote hisso (objects) mein todne ka moka deta hai, jise aap asaani se samajh sakte hain, dobara use kar sakte hain, aur badal bhi sakte hain bina pura program kharab kiye.**

**, jisse humara code strong, flexible aur asaani se samajh aane wala ban jata hai**

**OOP matlab:** Program ko chhoti chhoti cheezon (objects) mein divide karna. Har cheez ka **apna kaam** aur **apne data** hote hain.

Matlab:

* Aap apne program ko chhote tukdo mein divide karte hain (jise hum objects kehte hain).
* Har tukda apni zimmedari leta hai.
* Jab bhi aapko koi cheez change karni hoti hai, to sirf us chhote tukde (object) ko badalna padta hai, baaki sab theek rehta hai.
* Is se programming asaan ho jati hai, bugs kam hote hain, aur aap acha program banate hain.

**Example:**  
Sochiye aap gaari bana rahe ho:

* Engine, Steering, Aur Wheels sab alag alag objects hain.
* Agar engine mein koi problem hai to sirf engine object ko fix karo, baki sab ko chhedo mat.

, OOP programming ko chhote chhote objects mein tod kar problems ko solve karne ka tareeqa hai, jisse code samajhna, maintain karna, aur dobara use karna bohat asaan ho jata hai."

# ****Key Principles of OOP****

## 🔐 1. **Encapsulation** — “Chhupao aur control do”

**Definition:** Data aur methods ko aik class ke andar band karna, aur bahar walon ko sirf zarurat ki cheez dikhana. Aur kuch cheezein private rakhna, taake koi galti se data ko seedha na badal sake.

### 👩‍🏫 Real Life:

Tumhari pencil box — us ke andar eraser, pencil, sharpener hain. But kisi ko pata nahi andar kya kya cheez hai — sirf tum use kar sakti ho.

**Example:**  
Car class mein speed ko private rakhna taake sirf functions ke zariye hi speed change ho.

Iska faida:

* Data safe rehta hai
* Sirf jo chahiye, woh hi bahar dikhai deta hai

## 🧠 2. **Abstraction** — “Sirf zarurat ki cheez dikhana”

Complex cheezon ko chhupa kar sirf zaroori cheezein dikhana. Jaise hum car chalate hain, lekin engine kaise kaam karta hai, woh humko nahi pata.

**Definition:** Detail chhupa do, sirf kaam dikhayo.

### 👩‍🏫 Real Life:

Tum light on karne ke liye switch dabati ho — andar wire kaise kaam kar rahi, woh nahi pata.

**Example:**  
Car ka drive() method call karo, engine khud start ho jata hai, aapko uska detail nahi pata.

Iska faida:

* Code simple lagta hai
* Complex details chhup jati hain

## 👪 3. **Inheritance** — “Warasat milna”

**Definition:** Aik class (child) dosri class (parent) ke functions aur data ko use kare bina dobara likhe.

### 👩‍🏫 Real Life:

Tum apne papa se bike chalaana seekh lo. Tumhe zero se nahi seekhna.

* Code reuse hota hai
* Code organize hota hai

## 🐾 4. **Polymorphism** — “Ek naam, kaam alag”

**Definition:** Alag alag objects apne tareeqe se kaam karte hain lekin ek common method use karte hain.

👩‍🏫 Real Life:

Tumhara mobile mein “Play” button sab media apps mein hota hai — but har app mein alag kaam karta hai.

* Code flexible hota hai
* Different objects ko ek jaisa treat kar sakte hain

# ****2. Basics of Classes and Objects****

# ****What is a Class?****

A class is a blueprint or template for creating objects.

Ek **class ek naqsha (blueprint)** hoti hai — jaise **ek design** jisse hum kai cheezen bana sakte hain.  
Jaise ek "Car" ka design sabke liye same hota hai, lekin har car ki **color, speed, ya model** alag ho sakti hai.

# ****What is an Object?****

Class se banaya gaya **real item** ko hum **object** kehte hain.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ****Defining Attributes and Methods****

Attributes = object ke **data**  
Methods = object ke **actions**

# ****The self Keyword****

self ka matlab hota hai: "ye object jisko use kar rahe ho".

Tum jab object create karti ho, self usi object ki **andar ki cheezen (attributes/methods)** ko access karne ke liye hota hai.

### ✅ **1. Class kya hoti hai?**

"Class ek blueprint hoti hai jisme hum attributes aur methods define karte hain.  
Iska kaam hota hai ek structure dena jis se hum objects bana saken.  
Jaise agar ek Car class banayi jaye, to usme color, speed jaise attributes aur accelerate(), brake() jaise methods ho sakte hain."

### ✅ **2. Object kya hota hai?**

"Object ek class ka instance hota hai.  
Jab hum class se koi cheez banate hain, use object kehte hain.  
Har object ka apna alag data hota hai.  
Example: agar Car class hai to my\_car = Car("Red", 60) ek object hai."

### ✅ **3. Attributes kya hote hain?**

"Attributes object ke andar ke data hote hain — jaise color, speed.  
Hum unko self.color ke through class ke andar access karte hain."

### ✅ **4. Methods kya hote hain?**

"Methods class ke andar likhe hue functions hote hain.  
Yeh object ke actions define karte hain.  
Jaise accelerate(), brake() etc."

### ✅ **5. Constructor kya hota hai? (** \_\_init\_\_ **)**

"\_\_init\_\_ ek special method hota hai jo object banate waqt automatic chalta hai.  
Iske andar hum object ke initial attributes set karte hain.  
Isme self hamesha first parameter hota hai."

* "Constructor ek special method hota hai jo automatically chal jata hai jab hum class ka object banate hain."

### ✅ **6. self keyword kya karta hai?**

"self ka use hum object ke current instance ko refer karne ke liye karte hain.  
Agar hume object ke attribute ya method ko access karna hai class ke andar, to hum self use karte hain."

### ✅ **7. Private attribute ya method kya hota hai?**

"Jab kisi method ya attribute ke naam se pehle double underscore \_\_ laga dete hain,  
to wo private ho jata hai.  
Private ka matlab hota hai ki use sirf class ke andar hi access kiya ja sakta hai,  
class ke bahar se nahi."

### ✅ **Last Summary Line for Viva:**

* **Class** ek design hai,
* **Object** us design ka product hai,
* **Attributes** data hote hain,
* **Methods** actions hote hain,
* **self** current object ko refer karta hai,
* Aur **private** cheezen sirf class ke andar hi access hoti hain."

### **2. Parameterized Constructor kya hai?**

* "Agar hum constructor ko kuch parameters dena chahte hain, jese naam, age, brand, to usay parameterized constructor kehte hain."
* "Ye parameters object ke attributes ko initialize karte hain."

# ****3. Constructors and Destructors****

# ****What is a Constructor?****

* "Agar hum *\_\_init\_\_* method define nahi karte, to Python apne aap ek default constructor de deta hai jo kuch nahi karta."
* "Hum ek aisa constructor bhi bana sakte hain jo bina parameters ke ho, jise default constructor kehte hain."

# ****What is a Destructors? (del method)****

* "Destructor ek special method hota hai jo tab chalta hai jab object destroy hone wala hota hai, matlab uska kaam khatam ho jata hai."
* "Python mein destructor ka naam *\_\_del\_\_* hota hai."
* "Iska use hum cleanup karne ke liye karte hain, jaise files close karna ya memory release karna."

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ****4. Class Attributes and Instance Attributes****

**Python mein do tarah ke attributes hote hain:**

1. **Class Attributes**
2. **Instance Attributes**

**Dono ka use object ke data ko store karne ke liye hota hai, lekin scope aur behavior alag hota hai."**

### 🧬 **1. Class Attributes:**

* **"Ye directly class ke andar define kiye jate hain, kisi method ke bahar."**
* **"Ye sabhi objects ke liye common hote hain — yani ek baar define karne se sabhi objects usko use kar sakte hain."**

**Example:**

class School:

school = "PAF"

### 👤 **2. Instance Attributes:**

**"Instance attributes har object ke liye unique or alag alag hote hain, hote hain, aur ye hum \_\_init\_\_ method ke andar define karte hain."**

**Example:**

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

* **"Yahan name aur age har dog ke liye unique value lete hain."**

### 🛠️ Accessing & Modifying Attributes:

* **"Class attribute ko hum class name se ya object se access kar sakte hain:"**

print(Dog.species) # Class name se

print(dog1.species) # Object se

* **"Lekin agar hum object se modify karte hain to ek naya instance attribute ban jata hai."**

### 🔍 \_\_dict\_\_ Attribute:

* **"\_\_dict\_\_ ek dictionary hoti hai jo object ya class ke saare attributes ko store karti hai."**
* **"Python mein har object aur class ke paas ek \_\_dict\_\_ hoti hai jo unke saare attributes ko dictionary form mein store karti hai."**

print(dog1.\_\_dict\_\_) # Instance attributes

print(Dog.\_\_dict\_\_) # Class attributes

## ✅ Key Takeaways (Viva mein line by line bolo):

1. "Class attributes sab objects ke liye common hote hain."
2. "Instance attributes har object ke liye unique hote hain."
3. "Object se class attribute ko modify karne par ek naya instance attribute ban jata hai."
4. "*\_\_dict\_\_* ka use karke hum object ya class ke attributes dekh sakte hain."

### 🔍 **dict Attribute:**

print(Dog.\_\_dict\_\_)

print(dog1.\_\_dict\_\_)

**5. Methods in Python Classes**

## **Methods in Python Classes**

**Python mein class ke andar hum functions define karte hain jinhe methods kehte hain. Methods teen tarah ke hote hain: instance method, class method, aur static method."**

## **1. Instance Methods**

**"Yeh sabse common method hota hai. Iska pehla parameter self hota hai, jo object ko refer karta hai.**  
**Isse hum object ke attributes ko access aur modify kar sakte hain."** yeh method self parameter leta hai aur specific object ka data access karta hai.

**Example:**

def display(self):

print(self.name)

## **2. Class Methods (@classmethod cls)**

**"Yeh method class ke upar kaam karta hai, na ki object ke. Iska pehla parameter cls hota hai jo class ko refer karta hai.**  
**Isse hum class-level attributes ko modify kar sakte hain."**

**Example:**

@classmethod

def update\_species(cls, new\_species):

cls.species = new\_species

## **3. Static Methods (@staticmethod (NO Paramter…..)**

**"Yeh method class ya object dono se independent hota hai. Iska koi self ya cls parameter nahi hota.Yeh mostly utility functions ke liye use hota hai."**

**Example:**

@staticmethod

def is\_adult(age):

# ****6. Encapsulation****

### 🎯 **Definition:**

**"Data (attributes) aur us par kaam karne wale methods ko ek hi unit (class) mein band karna aur direct access ko control karna."**

Yeh **data hiding** ka concept follow karta hai taake kisi bhi object ka internal data bina permission ke change na ho.

➡️ **Data ko chhupana (hide karna)** — yani:

* Har koi class ke andar ke data ko **seedha access** na kar sake
* Sirf **functions ke zariye** hi access mile
* Taake **galat tareeqe se data change** na ho

## 🎯 Mukhtasir:

| **Urdu Mein** |  |
| --- | --- |
| Data aur us par kaam karne wale functions | Ek class mein ikattha karna |
| Data ko chhupana |
| Sirf functions se kaam karwana |  |

### ✅ 1. **Public (عوامی / sab ke liye)**

📣 **Jo sab ko dikh sakta hai**

* **Sab jagah se access ho sakta hai**
* Koi special nishan nahi lagta
* Isko har koi class ke bahar se bhi use kar sakta hai:

### 🟡 2. **Protected (محفوظ / androni istemal)**

🔒 **Class aur uski subclass ke andar use hota hai**

* Naam ke aagay **\_ (ek underscore)** lagta hai
* **Bahar se access possible hai**, lekin kehna yeh hota hai: "Ye sirf andr ka kaam hai, bahar se na chedho!"

### 🔴 3. **Private (نجی / sirf class ke andar)**

🚫 **Sirf class ke andar hi access hota hai**

* Naam ke aagay **\_\_ (do underscore)** lagta hai
* Python isko **mangling** kar deta hai — yani naam ko andar hi change kar deta hai
* Bahar se seedha access mushkil banata hai

## **Access Modifiers: Public, Private, and Protected:-**

Python mein public, protected, aur private ka use **naming conventions** se hota hai (jaise: \_, \_\_).

**Q: Python mein access modifiers kaise kaam karte hain?**

Python mein access modifiers ke liye hum naming conventions use karte hain — jaise koi variable public ho to direct likhte hain, protected ho to ek underscore *\_* aur private ho to double underscore *\_\_* use karte hain.

### 📖 **Kahani: BankAccount ka Magic Box**

Socho aik **magic box** hai jiska naam hai BankAccount 🎁  
Uske andar teen cheezein hain:

1. account\_holder ➡️ **Public** (har koi dekh sakta hai)
2. \_balance ➡️ **Protected** (sirf family wale dekhte hain 👨‍👩‍👧‍👦)
3. \_\_pin ➡️ **Private** (sirf box khud hi janta hai 🔐)
4. Jab tum **naya account** banati ho, tum **naam aur paisa** deti ho.
5. Naam **public** hota hai (har koi dekh sakta hai).
6. Paisa **protected** hota hai (signal milta hai ke direct chhed-chhad na karo).
7. Pin **private** hota hai — kisi ko nahi batana! 🤫

## **Getter and Setter Methods**

**Q: Getter aur Setter ka role kya hai?**

getter se hum data ko safely read karte hain, aur setter se validation ke sath update karte hain — bina directly attribute ko access kiye.

# ****7. Inheritance****

**"Ek class (child/subclass) doosri class (parent/superclass) ke attributes aur methods ko inherit karti hai.**

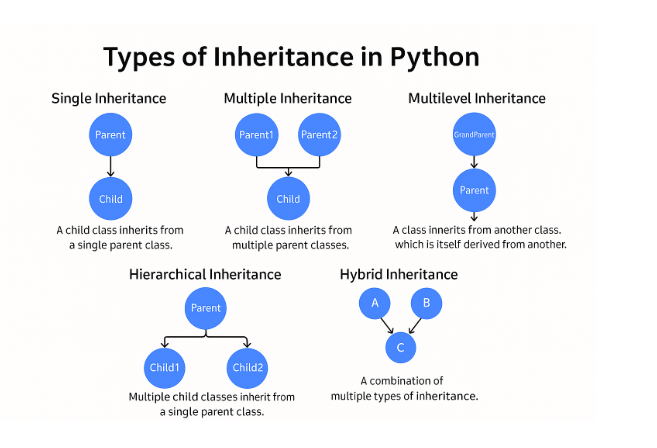
## **super() Function:**

**Q: Python mein super() ka kya kaam hai?**

super() ka use parent class ke methods ya constructor ko call karne ke liye hota hai, especially jab hum override kar rahe hote hain.

**Q: Method overriding kya hota hai?**

jab child class ek aisa method define karti hai jo already parent class mein defined hai, lekin uska behavior badal diya jata hai.



## ✅ **Types of Inheritance in Python (with Simple Explanation)**

Python mein mainly **5 types of inheritance** hote hain:

## **1. Single Inheritance**

👉 Ek **child class** ek hi **parent class** se inherit karti hai.

|  |
| --- |
| class Parent:  pass  class Child(Parent):  pass |

## **2. Multiple Inheritance**

👉 Ek **child class**, **do ya zyada parent classes** se inherit karti hai.

|  |
| --- |
| class Parent1:  pass  class Parent2:  pass  class Child(Parent1, Parent2):  pass |

### 3. **Multilevel Inheritance**

👉 Ek class, **ek aur child class** se inherit karti hai, jo khud ek **parent class** se inherit karti hai.

|  |
| --- |
| class GrandParent:  pass  class Parent(GrandParent):  pass  class Child(Parent):  pass |

🟢 **3-level hierarchy**: GrandParent → Parent → Child

### **4. Hierarchical Inheritance**

👉 **Ek parent class**, multiple **child classes** ko inherit karata hai.

|  |
| --- |
| class Parent:  pass  class Child1(Parent):  pass  class Child2(Parent):  pass |

🟢 **One parent → multiple children**

### 5. **Hybrid Inheritance**

👉 Jab **multiple types of inheritance ek saath** use hoti hain (mix of single, multiple, multilevel etc.)

|  |
| --- |
| class A:  pass  class B(A):  pass  class C:  pass  class D(B, C): # Hybrid = Multilevel + Multiple  pass |

# ****8. Polymorphism(Aik naam – multiple kaam)****

Polymorphism ka matlab hota hai ke ek hi method ya operator different classes mein different tareeqay se kaam kare.Yani **"ek hi interface, lekin different behavior."**

## 🧠 **Polymorphism** kya hota hai?

**Polymorphism** ka matlab hota hai:  
**“Ek hi naam, lekin alag behavior.”**  
💡 **Poly = Many** + **Morph = Forms**

Yani **ek hi method ya operator** alag-alag class ke objects ke liye **alag kaam karega**.

### 🎯 Real-Life Example:

Agar main bolun **"Drive!"**, to:

* Car chalegi 🚗
* Truck chalega 🚛
* Bike chalegi 🏍️

Sabko **Drive()** bola, lekin sabka **style alag tha** — **yeh hi polymorphism hai**!

## **Types of Polymorphism in Python**

### 🔸 1. **Method Overriding**

**Kya hota hai?**  
Jab subclass apne parent class ke method ko dobara define karta hai apne tareeqay se.

**Example:**

|  |
| --- |
| class Animal:  def speak(self):  return "Animal sound"  class Dog(Animal):  def speak(self):  return "Woof!"  class Cat(Animal):  def speak(self):  return "Meow!"  def animal\_sound(animal):  print(animal.speak())  animal\_sound(Dog()) # Output: Woof!  animal\_sound(Cat()) # Output: Meow |

### 🔸 2. **Operator Overloading**

**Kya hota hai?**  
Jab hum normal Python operators (like +, -) ko apni class ke objects ke liye custom behavior de dete hain.

**Example:**

|  |
| --- |
| class Point:  def \_\_init\_\_(self, x, y):  self.x = x  self.y = y  def \_\_add\_\_(self, other):  return Point(self.x + other.x, self.y + other.y)  def \_\_str\_\_(self):  return f"Point({self.x}, {self.y})"  p1 = Point(1, 2)  p2 = Point(3, 4)  p3 = p1 + p2  print(p3) # Output: Point(4, 6) |

### 🔸 3. **Duck Typing**

**Kya hota hai?**  
Python mein **type important nahi**, behavior important hota hai. Agar koi object quack() method rakhta hai, to Python use duck maan leta hai.

**Example:**

|  |
| --- |
| class Duck:  def quack(self):  return "Quack!"  class Person:  def quack(self):  return "I can quack like a duck!"  def make\_it\_quack(thing):  print(thing.quack())  make\_it\_quack(Duck()) # Output: Quack!  make\_it\_quack(Person()) # Output: I can quack like a duck! |

📝 **Explanation:**  
make\_it\_quack() function kisi bhi object ko accept karta hai jo quack() method rakhta hai — chahe wo Duck ho ya Person.

### ✅ **Simple Jawab:**

**Inheritance-based polymorphism** mein hum base class banate hain aur subclasses usay inherit karte hain, aur same method ko override karte hain.

**Duck typing** mein koi inheritance nahi hoti — agar object ke paas required method hai, to wo accept ho jata hai.

📌 **"Method ka naam same hona zaroori hota hai."**

### 📌 Ek Line mein farq:

**Inheritance mein rishta hota hai, duck typing mein sirf method ka hona zaroori hota hai.**

Agar viva mein pucha jaye:

### ❓"Duck typing mein kya zaroori hota hai?"

**Sirf method ka naam same hona zaroori hota hai, class ka type important nahi hota.**

## 🔑 **Key Takeaways (Viva ke liye)**

1. **Method Overriding** – subclass method ko override karta hai.
2. **Operator Overloading** – operators ka custom use.
3. **Duck Typing** – object ka behavior matter karta hai, uski class nahi.

**9. Abstraction– Sirf "Kya", "Kaise" nahi**

Abstraction ka matlab hota hai **complex cheezon ko simple tarike se represent karna**. Hum sirf **function kya karega** ye batate hain — **kaise karega**, ye baad mein decide hota hai.

### 🔒 **1. What is Abstraction? (Asan Alfaaz Mein)**

Imagine tum remote se TV chala rahi ho — tum sirf **button dabati ho** (start()), magar **andar kya wiring hai, kaise signals ja rahe hain**, yeh sab **tumhein nahi pata hona chahiye** — **bas kaam hona chahiye**.  
Yeh hi hota hai **abstraction** — focus on ***what*** it does, not ***how*** it does.

from abc import ABC, abstractmethod

Abstraction ke liye hum Python mein use karte hain:Isko har koi class ke bahar se bhi use kar sakta hai:Isko har koi class ke bahar se bhi use kar sakta hai:Top of Form

Bottom of Form

**ABC** = Abstract Base Class  
🏷️ **@abstractmethod** = Ye method **lazmi** har child class ko banana hoga

### 🧠 Final Line:

**Abstraction** ek "farmaan" ki tarah hota hai —  
"Jo bhi mera bacha ho (child class), usay yeh kaam lazmi karna hoga, lekin tareeqa us ka apna hoga!" 🎯

### ✅ 1. Base Class

* Yeh **sab classes ki maa** hai.
* Har class Python mein **object se hi shuru** hoti hai.

### ✅ 2. Default Methods

* Iske paas kuch **methods already bana hotay hain**, jaise \_\_init\_\_, \_\_str\_\_, \_\_repr\_\_, etc.
* Tumhari class agar yeh override nahi karti, to yeh object wale use ho jatay hain.

### ✅ **Real-life Example:**

TV remote ka button press karte ho to channel change hota hai — **aapko ye nahi pata hota ke andar kya wiring hai**.  
Yehi abstraction hai:  
**Kya karna hai – pata hai**  
**Kaise karna hai – chhupa hota hai**

## 🔸 **Python mein Abstraction kaise hoti hai?**

Python mein abstraction karne ke liye **abstract class** banayi jaati hai.

### Tools:

* ABC (Abstract Base Class)
* @abstractmethod decorator

from abc import ABC, abstractmethod

|  |
| --- |
| class Vehicle(ABC): # Abstract class  @abstractmethod  def start(self): # Abstract method  pass |

🔸 ABC → class ko abstract banata hai  
🔸 @abstractmethod → method mandatory bana deta hai  
🔸 **Abstract class ka object nahi bana sakte**

## 🔐 **Key Benefits (Viva Points)**

🔹 **Abstraction** coding ko simple aur secure banata hai  
🔹 **Abstract classes** blueprint jaise hote hain — khud se use nahi hotay  
🔹 **@abstractmethod** force karta hai child class ko specific method define karne  
🔹 Isse aap rule bana sakte ho ke "har class yeh kaam karegi"

## 🔑 Viva ke liye Key Lines:

* **“Abstraction mein hum sirf define karte hain ke ek method hona chahiye, lekin uska implementation subclass karega.”**
* **“Abstract class Python mein banane ke liye abc module use hota hai.”**
* **“@abstractmethod decorator method ko mandatory banata hai.”**
* **“Abstract class ka object direct create nahi kiya ja sakta.”**

## 🔹 **Kya hoti hai** object **class?**

Python mein object class **sabse pehli (root/base) class hoti hai**.  
Chahe aap koi built-in class use karo (jaise int, str) ya custom class banao — sab kuch **indirectly ya directly object class se inherit karta hai**.

## **Key Points of object class:**

### 1️⃣ Base Class

* Jab aap koi class define karte ho bina kisi parent ke:

## 🧠 Viva ke liye Bold Points:

* **“object class Python ki sabse root/base class hoti hai.”**
* **“Har class directly ya indirectly object se inherit karti hai.”**
* **“object class kuch built-in methods deti hai jaise \_\_init\_\_(), \_\_str\_\_(), \_\_eq\_\_(), etc.”**
* **“Python ka Unified Type System ensure karta hai ke sab kuch object ho, isliye behavior predictable hota hai.”**

## 🎯 **In Viva, Aise Samjha Sakte Ho:**

**"Python ka Unified Type System ka matlab hai ke har cheez — numbers, strings, lists, aur even functions — sab objects hote hain. Ye sab object class se inherit karte hain. Is wajah se unmein kuch common behavior hota hai jaise str(), type(), aur unke methods ko access karna dir() se. Is system se Python ka behavior consistent, flexible, aur dynamic hota hai."**

## ✅ **Benefits of the Unified Type System in Python — Simple Explanation**

### 🔸 1. **Simplicity (Asaani)**

* Har cheez object hai — alag-alag rules ya syntax nahi yaad rakhne padte.
* Aap ek hi tarike se har data type ke saath kaam kar sakte ho.

🔹 Example:

|  |
| --- |
| print(type(10)) # int  print(type("hello")) # str |

### 2. **Consistency (Ek Jaise Rules)**

* Sab types me similar behavior hota hai — same functions like type(), str(), dir() sab pe kaam karte hain.
* Chahe built-in ho ya custom class.

🔹 Example:

|  |
| --- |
| print(str(42)) # "42"  print(str([1, 2, 3])) # "[1, 2, 3]" |

### 3. **Extensibility (Apni Classes Banana Aasaan)**

* Aap **apni custom classes** bana sakte ho jo built-in types ki tarah behave karein.
* Python ke special methods (like \_\_str\_\_, \_\_len\_\_, \_\_getitem\_\_) isme help karte hain.

|  |
| --- |
| class MyClass:  def \_\_str\_\_(self):  return "I'm custom!"  obj = MyClass()  print(str(obj)) # I'm custom |

!

### 🔸 4. **Polymorphism (Ek Interface, Multiple Objects)**

* Agar 2 alag types ke objects same method follow karein, to aap unhe **interchangeably** use kar sakte ho.
* Yeh object-oriented concept Python mein naturally kaam karta hai.

🔹 Example:

|  |
| --- |
| print(len("hello")) # 5  print(len([1, 2, 3])) # 3 |

## ✨ **Conclusion:**

Python ka unified type system ye ensure karta hai ke sab kuch object ho, aur sabhi same rules follow karein. Isse programming simple, consistent, aur powerful ban jaati hai. Chahe aap kisi bhi type ke object ke saath kaam karo — aapka tarika ek jaisa hi rahega.

## 🧙‍♂️ **Special (Magic/Dunder) Methods — As if Python Has Magic!**

### 📦 1. \_\_init\_\_ → **Setup / Constructor**

|  |
| --- |
| class Toy:  def \_\_init\_\_(self, name):  self.name = name |

### 🏷️ 2. \_\_str\_\_ vs \_\_repr\_\_ → **Display Karna**

* \_\_str\_\_: For users (friendly)
* \_\_repr\_\_: For developers (technical)

|  |
| --- |
| * def \_\_str\_\_(self): return "Nice Toy!" * def \_\_repr\_\_(self): return "Toy('Nice Toy!')" |

### 📏 3. \_\_len\_\_ and \_\_getitem\_\_ → **Length aur indexing**

|  |
| --- |
| def \_\_len\_\_(self): return len(self.toys)  def \_\_getitem\_\_(self, index): return self.toys[index] |

### ⚖️ 4. \_\_eq\_\_, \_\_lt\_\_, etc → **Comparison Support**

|  |
| --- |
| def \_\_eq\_\_(self, other): return self.size == other.size |

### 🧙‍♀️ 5. Full Example – **MagicToyCatalog**

store = MagicToyCatalog()

store["RoboDog"] = (49.99, "RC dog") # \_\_setitem\_\_

print(store["RoboDog"]) # \_\_getitem\_\_

print("Dragon" in store) # \_\_contains\_\_

print(len(store)) # \_\_len\_\_

for toy in store: print(toy) # \_\_iter\_\_

print(store) # \_\_str\_\_

print(store(20)) # \_\_call\_\_

del store["RoboDog"] # \_\_delitem\_\_

🧠 **Final Tip for Viva:**

“Python ka unified type system aur special methods Python ko itna flexible aur readable banate hain ke custom classes bhi built-in types ki tarah behave karte hain — isse code short, powerful aur natural ban jaata hai!”

=================================================================================