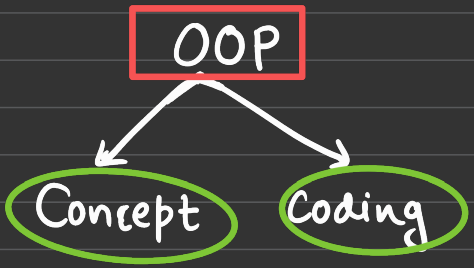

DECODE
AI

1.10.1

# Mastering OOPs in Python

→ Important for both Software Engineer and AI/ML engineer

→ My Interview experience (OOPs)
- ↳ NVIDIA → problem solving round
- ↳ Amazon, Google → dsa round
- ↳ Qualcomm, Samsung Research, Nutanix




→ OOPs usecase as an AI/ML Engineer

① Industry standards data pipeline is not like Notebook Procedural style programming

② Not all deployments are on AWS/GCP/Server.

Some deployments are on-device (low-end/high-end)

↳ Bespoke AI

③ DL Framework Pytorch – is primarily object-oriented.

④ Real world business problems are modelled as object-oriented

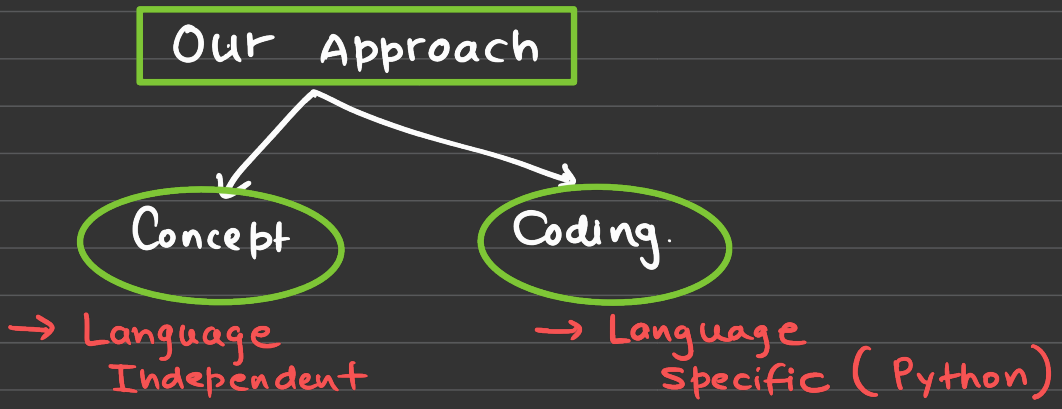
You don't just build models. You also
- Read documentation
- Understand business
- Understand other's code.
- build pipeline as per Industry Standards.

# Mastering OOPs in Python


## Table of Content

⑨ Data abastraction in Python

⑩ Static Concept and Copy Constructor




Like Subscribe

1.10.2

# Procedural Vs Object-oriented

Problem: Design a Student Management system.

1. Add Student details
2. Display Student details.

```python
# Procedural version using global list
students = []

def add_student(name, age, grade):
    students.append({'name': name, 'age': age, 'grade': grade})

def display_students():
    for s in students:
        print(f"Name: {s['name']}, Age: {s['age']}, Grade: {s['grade']}")

# Usage
add_student("Alice", 14, "8th")
add_student("Bob", 15, "9th")
display_students()
```

```python
# Object Oriented Programming Approach
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}")

class StudentManager:
    def __init__(self):
        self.students = []

    def add_student(self, name, age, grade):
        student = Student(name, age, grade)
        self.students.append(student)

    def display_all(self):
        for student in self.students:
            student.display()

# Usage
manager = StudentManager()
manager.add_student("Alice", 14, "8th")
manager.add_student("Bob", 15, "9th")
manager.display_all()
```

# Benefits of OOPs in above example

| Benefit | Explanation |
| --- | --- |
| 🔐 **Encapsulation** | Student data ( `name` , `age` , `grade` ) is tied with behavior ( `display()` ) in one object. |
| 📦 **Modularity** | `Student` and `StudentManager` are separate, reusable components. |
| 🧪 **Testability** | You can test `Student` and `StudentManager` independently. |
| 🔁 **Reusability & Extension** | Can subclass `Student` (e.g., `HighSchoolStudent` ) to extend behavior. |
| 👥 **Multiple Instances** | Can create multiple independent `StudentManager` objects for different classes/schools. |
| 💡 **Clear Real-world Mapping** | `Student` models a real student naturally, making the code easier to understand. |

1.10.3

# Introduction to OOPs in Python


→ Object-oriented Programming (OOP) is a **programming paradigm** that organizes code using **Classes** and **objects**.

Style or approach to solving problems using code.

→ Any real-world entity that possesses some **attributes** and **behaviours** can be modelled as an Object.

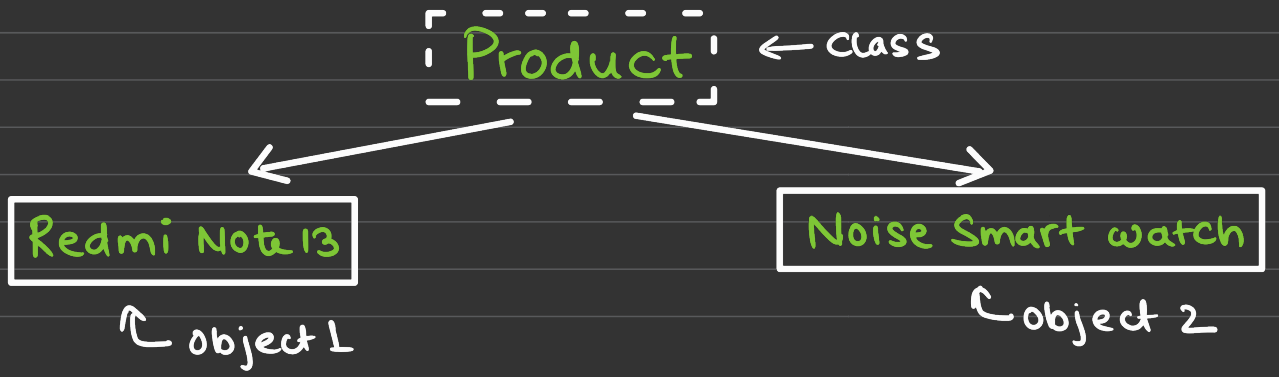
→ And the **blueprint** of the object is called a Class.

→ And object can also be called an instance of a class

# Terminology

1. Object → Real world entity
2. Class → Blueprint of an object.
3. Attributes → Member Variables or Attributes
4. Behaviours → Member functions or Methods.

## Analogy : E-Commerce WEBSITES

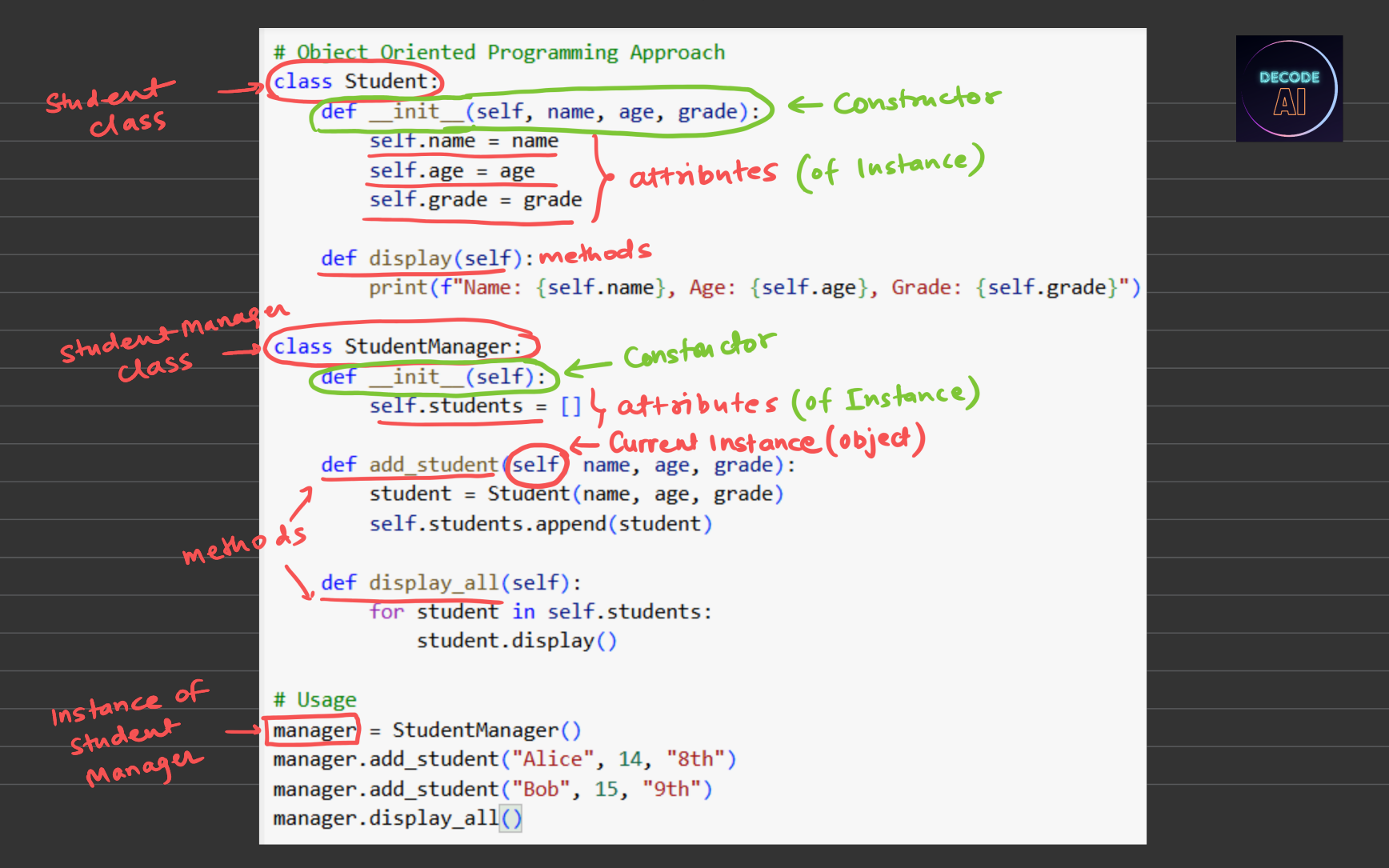


# Flipkart Product

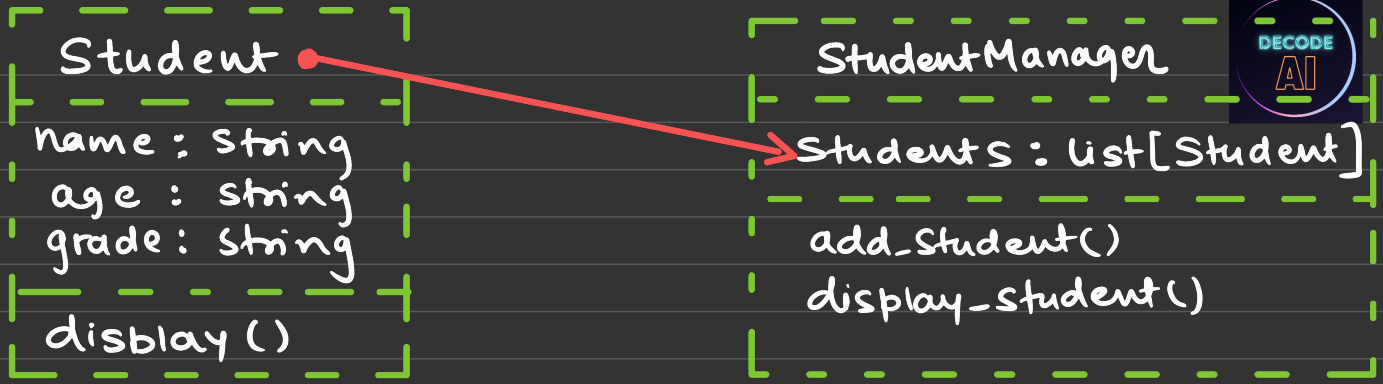
Product class ⇒

```
name : string
price : float
stock : int
rating : float
review : list[string]
Category: String
```

```
add_to_cart(user)
remove_from_cart(user)
is_available()
calculate_delivery_charge()
get_average_rating()
get_review()
add_review(rating, comment)
display_details()
display_additional_details()
```




```python
# Object Oriented Programming Approach
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}")

class StudentManager:
    def __init__(self):
        self.students = []

    def add_student(self, name, age, grade):
        student = Student(name, age, grade)
        self.students.append(student)

    def display_all(self):
        for student in self.students:
            student.display()

# Usage
manager = StudentManager()
manager.add_student("Alice", 14, "8th")
manager.add_student("Bob", 15, "9th")
manager.display_all()
```




* Problem : you need to manage a record of students taking part in different clubs → Music, Dance, Coding, Sports.

MusicClubStudentManager → Ram, Monu

DanceClubStudentManager → Sonu, Shivani

CodingClubStudentManager → Ram, Monu, Sonu, Shivani

SportsClubStudentManager → Ram

→ If we see above example, grouping Students into multiple Subgroups, managing student data etc becomes much easy using Concept of Classes and objects compared to

→ global variable implementation using List and dictionaries.

Hands-on Time

Like Subscribe

1.10.13

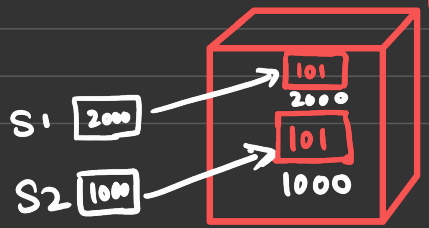
# Copy Constructor in Python

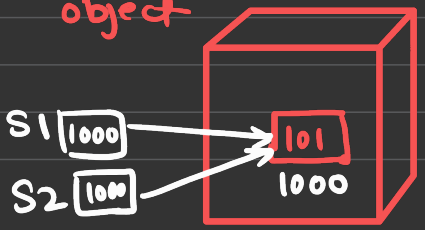
→ C++

↳ automatically available in C++ but not built-in in Python.
(built-in)

↳ C++ Support automatic on assignment copy constructor Call but not python.

| C++/Java | Python |
|---|---|
| Student S1(101); | S1 = Student(101) |
| Student S2=S1; | S2 = S1 |
| ↑ Copy constructor is called | ← No copy, Refer to some object |



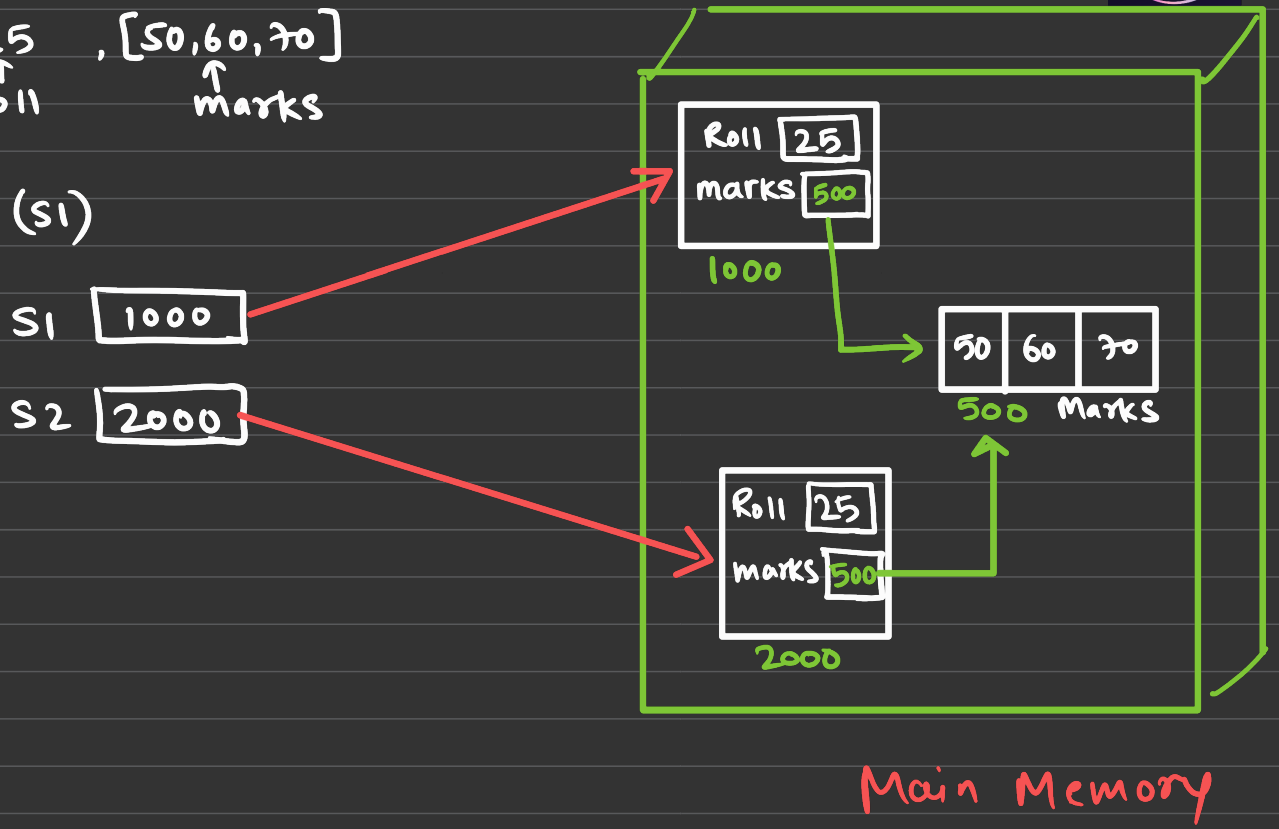



# * Shallow Copy

S1 = Student( 25 , [50,60,70] )

Roll — marks

S2 = Copy.Copy(S1)

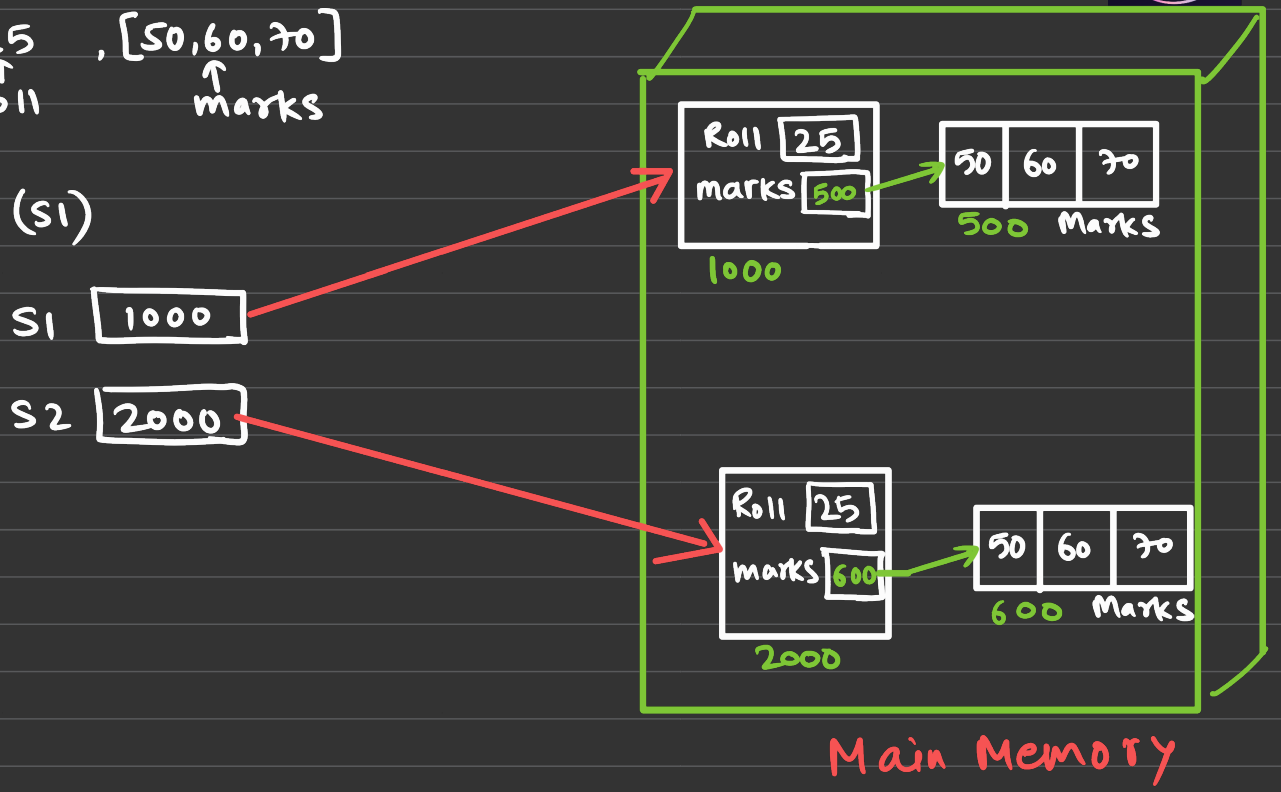


# * Deep Copy

S1 = Student( 25 , [50,60,70] )

Roll — marks

S2 = copy.copy(S1)



* Copy Constructor implementation

① __copy__() method #auto called on copy.copy(obj)

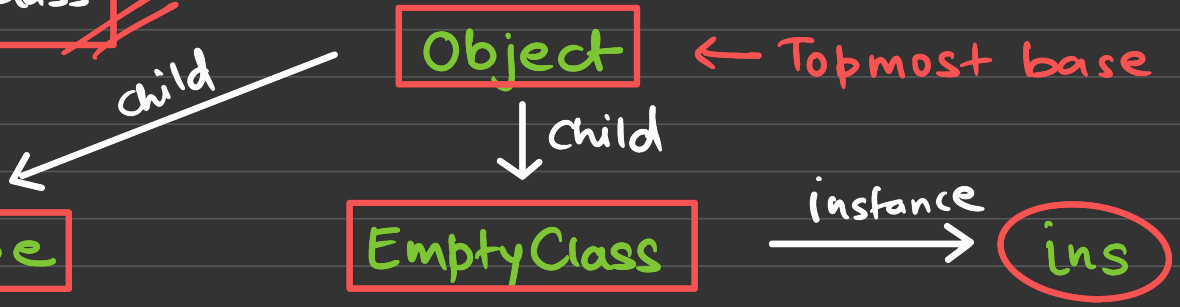
② Copy. Copy (obj)

③ Copy. deepCopy (obj)

Like Subscribe

# Facts about Classes in Python

~~Object~~
~~type~~
~~Empty class~~




→ ins is an instance of EmptyClass. ins is also an instance of object class

→ Since everything in Python is an object (instance).

→ EmptyClass is a class but it is also an instance (object) of object class

→ type is a class.

type(EmptyClass) ⇒ <class 'type'>

→ All user defined class (eg. EmptyClass) are instance of type

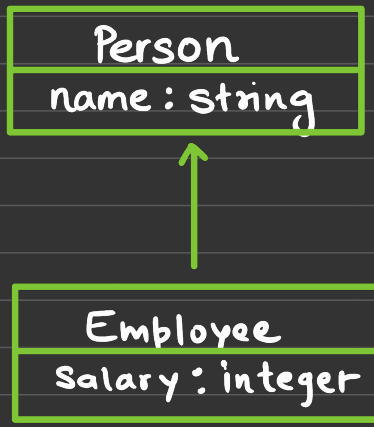
isinstance(type, type) ⇒ True

→ Type is an instance of itself.

Like Subscribe

# Inheritance

## ① Single Inheritance

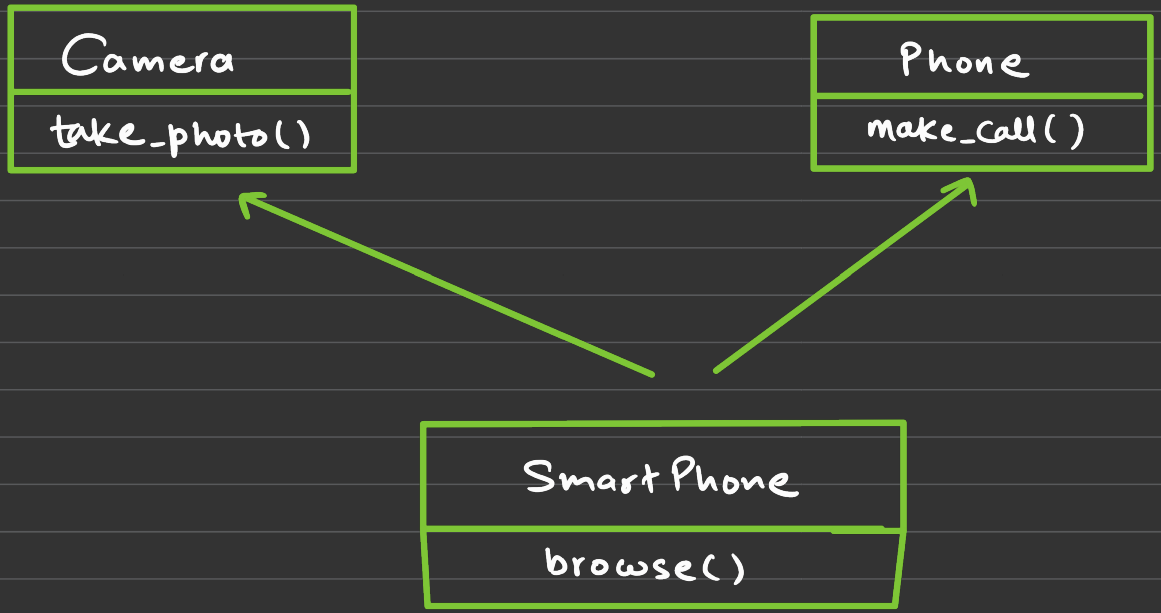



→ Employee inherits from person, adding a salary attribute

## UML diagram

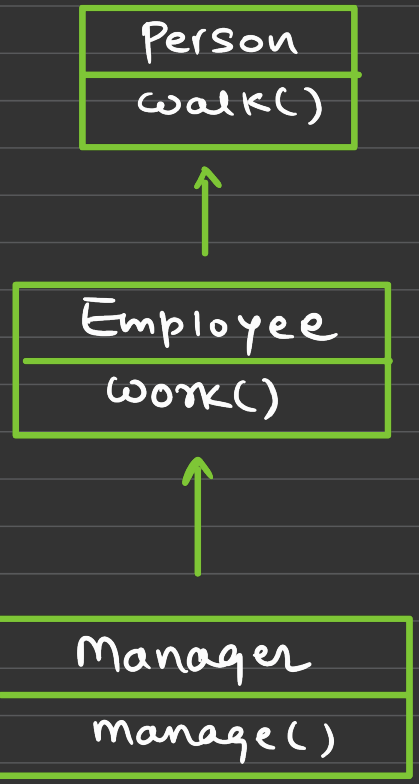
Unified modelling language

## ② Multiple Inheritance

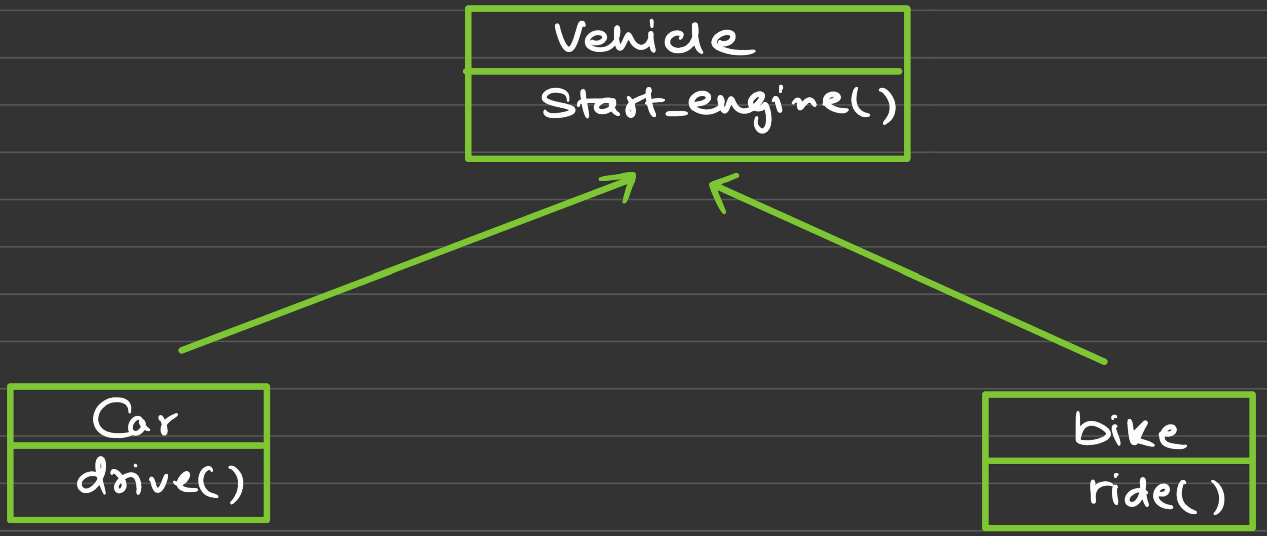



→ A Smartphone can act as Camera and phone.

## ③ Multilevel Inheritance

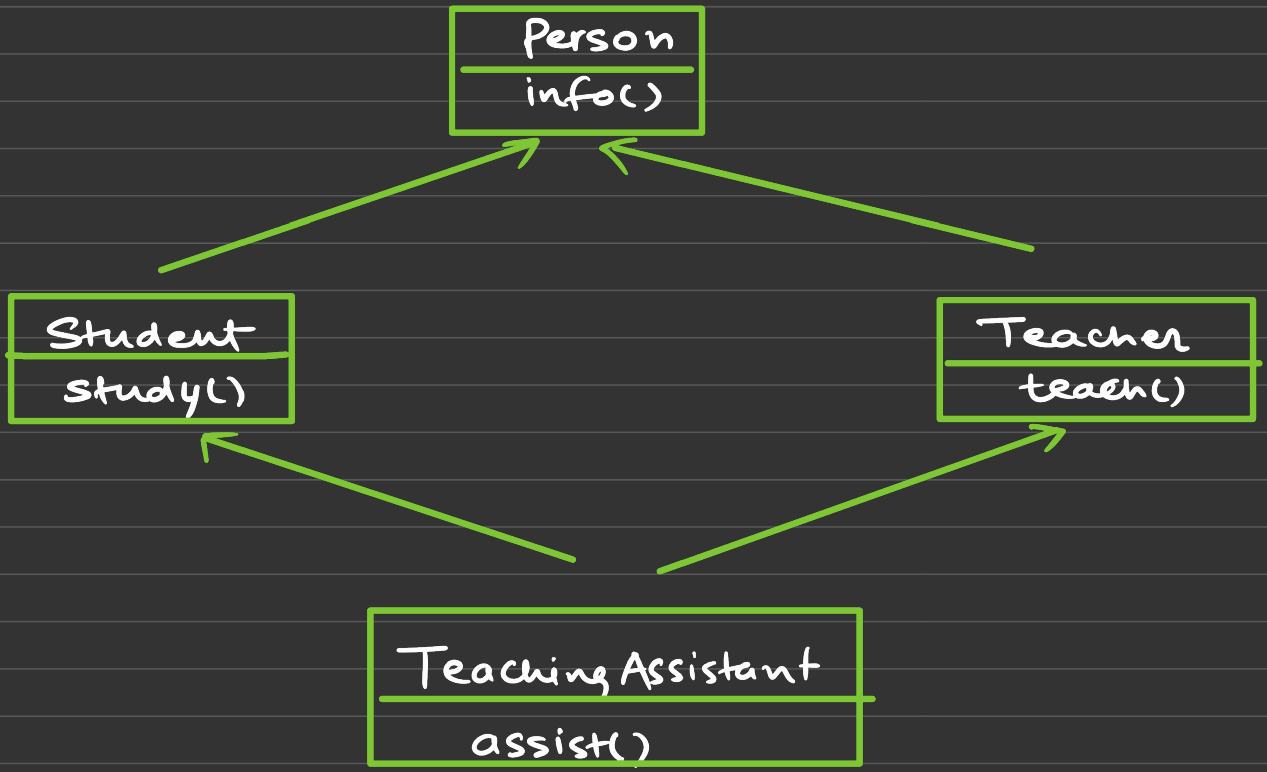


## ④ Hierarchical Inheritance




→ A vehicle can be car, bike or bus.

⑤ Hybrid Inheritance → Combination of more than 1 types of Inheritance



Like  Subscribe