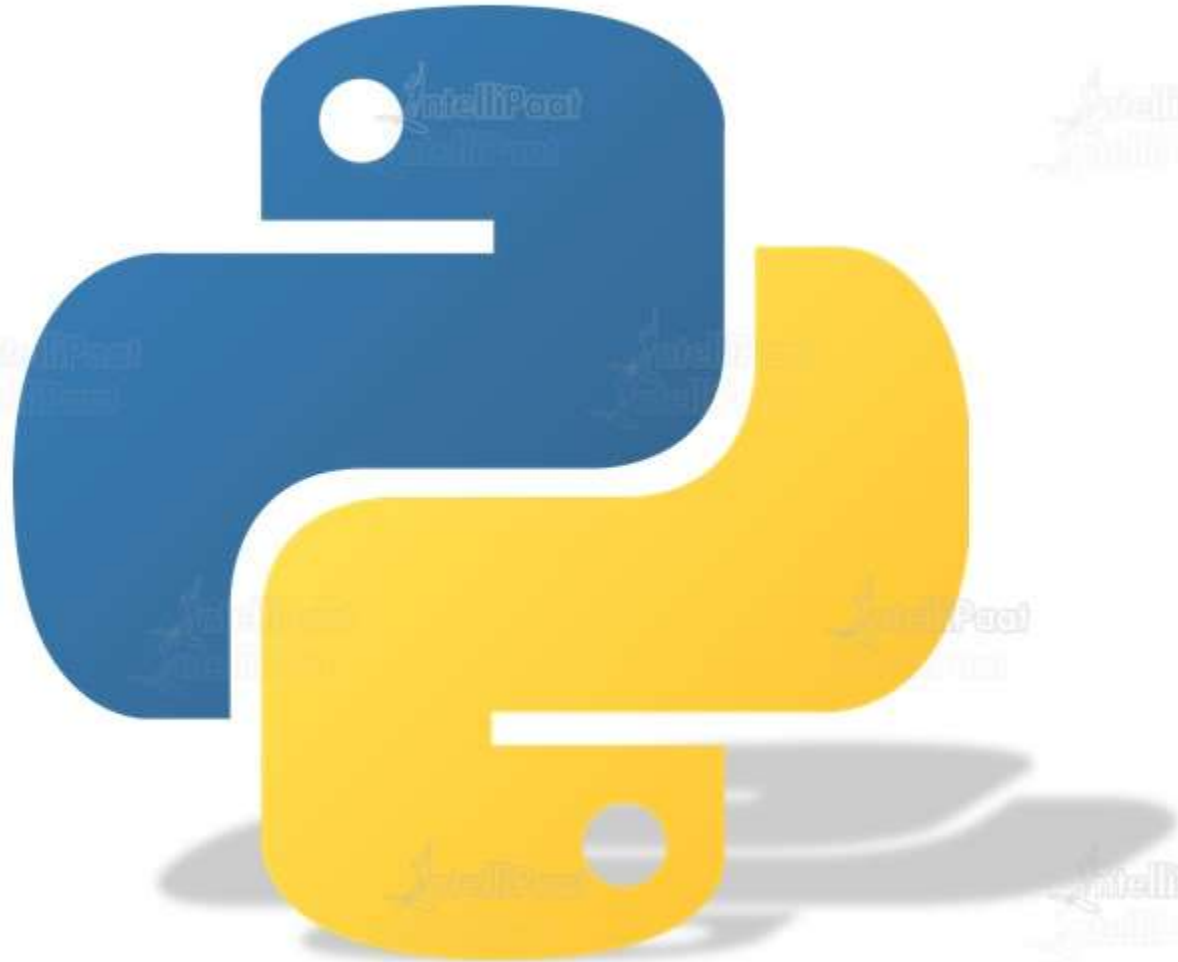
Data Science with Python

OOP in Python



Agenda

01 Introduction to OOPs

02 Real-world OOP example

03 OOPs – Classes and Objects

04 Magic Methods

05 Inheritance in Python

06 Encapsulation in Python

07 Polymorphism in Python

Introduction to OOPs

Introduction to OOPs

Object-Oriented Programming is a programming paradigm where you can use a real world entity which is called an **Object**.

Let us consider an example

- **Attribute**: Name, Age, Color
- **Behaviour**: Singing, Dancing

Parrot

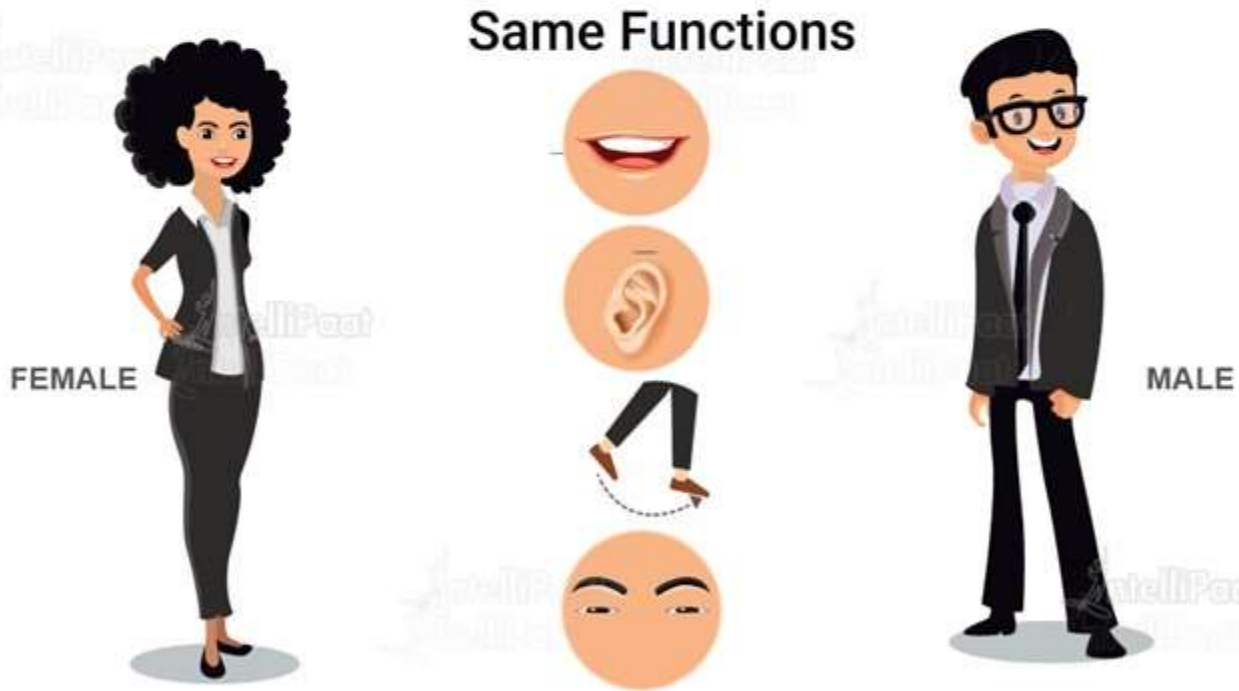
Introduction to OOPs

Basic Principle of OOPS

Real-world OOP example

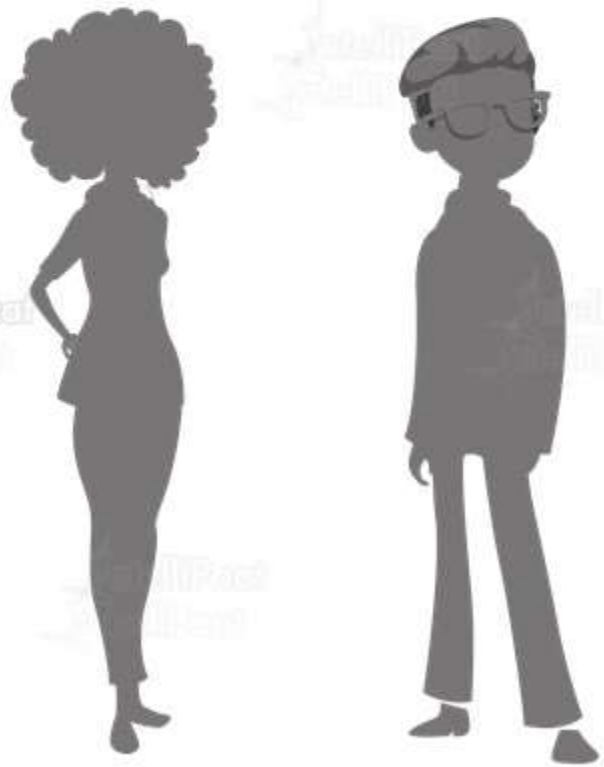
Real-world OOP example

Every Human Being is Classified into:



Real-world OOP example

Considering Human Being is a **class**

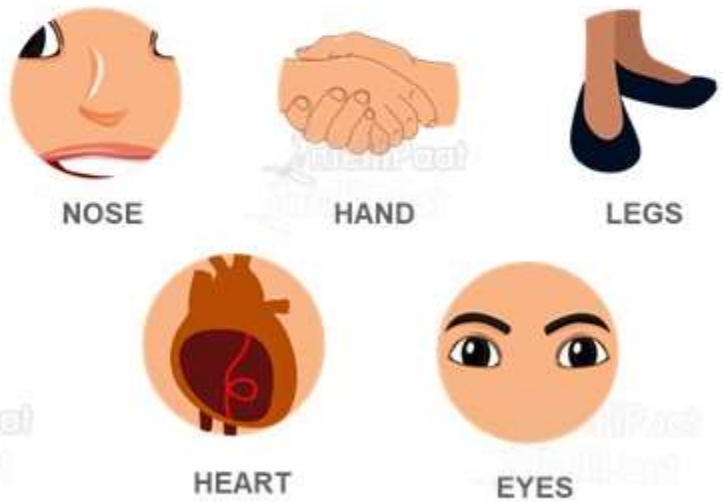


Real-world OOP example

Common body features and functions are **Class Attributes**

Every Human has:

Common Body Parts:



Every Human has:

Common Body Function:

Real-world OOP example

Male and Female are **inherited** from Class Human Being

FEMALE

MALE

Real-world OOP example

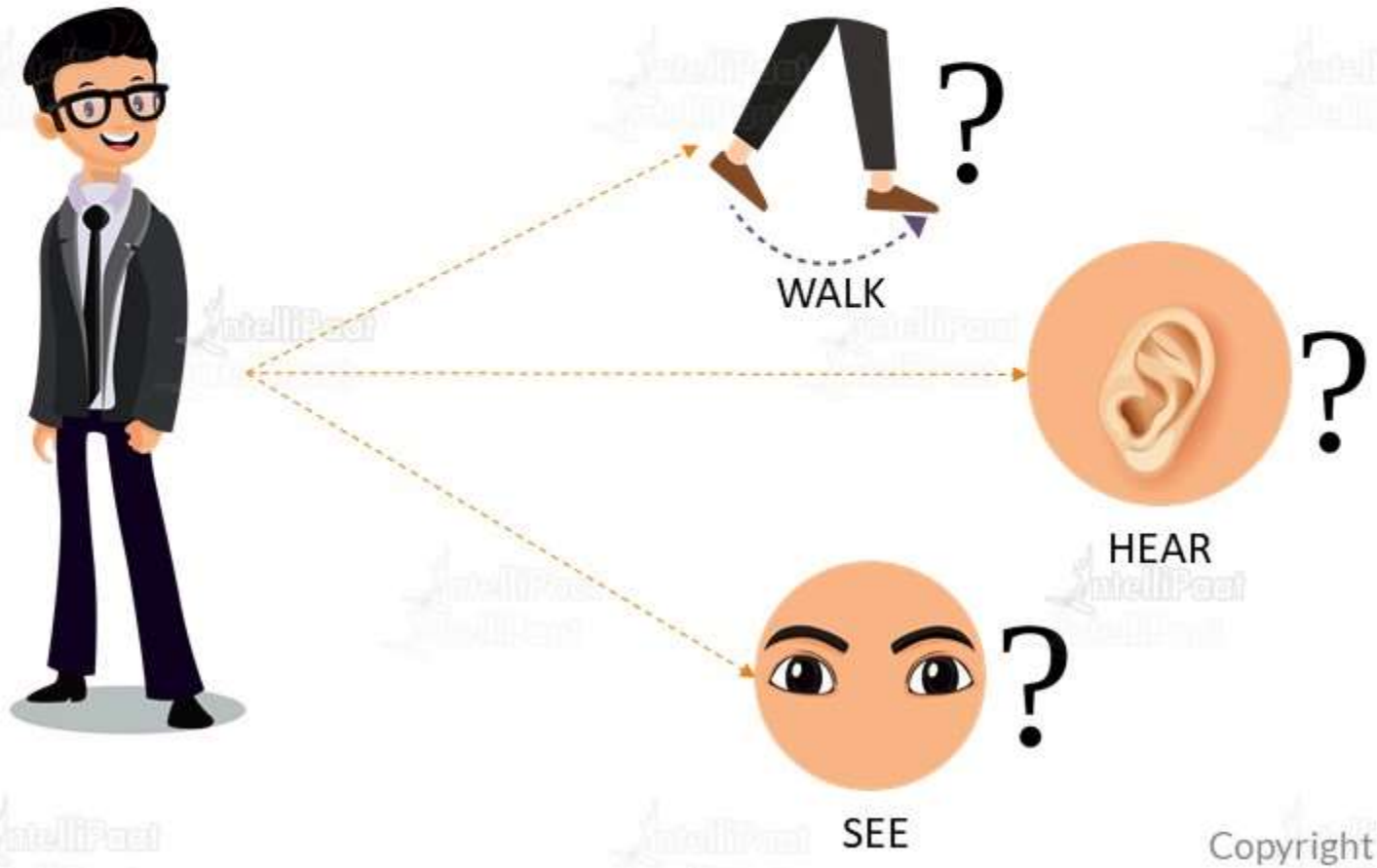
'**Name**' and '**Age**' are **object** of class MALE

Class: MALE
Name: Victor
Age: 24

- Objects have a physical existence
- Class is just a logical definition

Real-world OOP example

You don't know the detail of how you walk, listen or see.
i.e. **its hidden or Encapsulated**



Real-world OOP example

'She' can be a woman, wife, mother and a teacher at the same time which is many forms or **Polymorphism**

WOMAN

WIFE or MOTHER

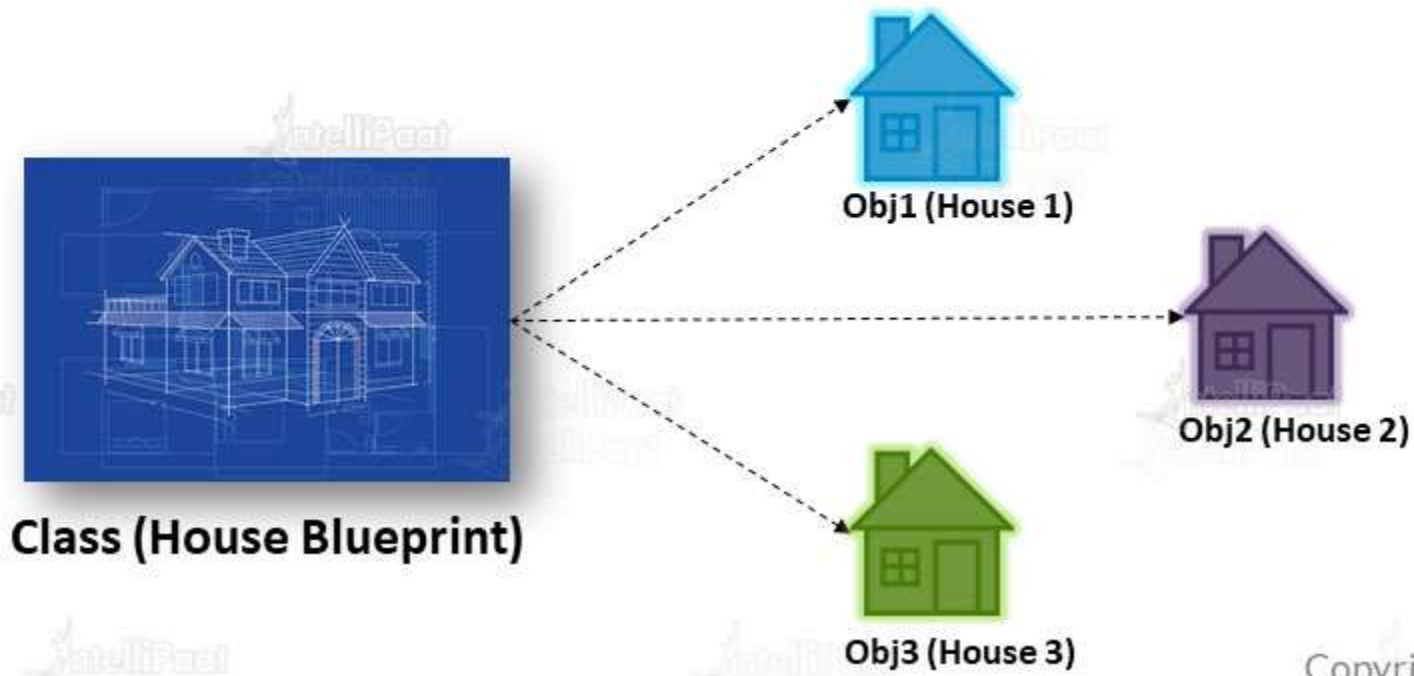
TEACHER

OOPs – Classes and Objects

OOPs – Classes and Objects

What are Objects and Classes?

Class is a blueprint for an object and the objects are defined and created from classes(blueprint)



OOPs – Classes and Objects

What are Objects and Classes?

01 Object is the basic unit of object-oriented programming

02 An object represents a particular instance of a class

03 There can be more than one instance of an object

04 Each instance of an object can hold its own relevant data

05 Objects with similar properties and methods are grouped together to form a Class

OOPs – Classes and Objects

What are Objects and Classes?

01 Object is the basic unit of object-oriented programming

02 An object represents a particular instance of a class

03 There can be more than one instance of an object

04 Each instance of an object can hold its own relevant data

05 Objects with similar properties and methods are grouped together to form a Class

OOPs – Classes and Objects

A class describe the structure of an object. It is made up of two things

Fields

A Field is simply a variable that is associated with a class which allows its object to store some data in it and can be accessed using the same object

Method

A Method is simply a function that is associated with an object of a class and can be called using that object

OOPs – Classes and Objects

A class describe the structure of an object. It is made up of two things

Fields

A Field is simply a variable that is associated with a class which allows its object to store some data in it and can be accessed using the same object

Method

A Method is simply a function that is associated with an object of a class and can be called using that object

OOPs – Classes and Objects

How to create a Object in Python?

Syntax

<obj-name> = NameOfClass()

Example

```
obj1 = ClassName()
obj1
```

```
<__main__.ClassName at 0x1a0eb1d3d48>
```

OOPs – Classes and Objects

How to create a Object in Python?

Syntax

<obj-name> = NameOfClass()

Example

```
obj1 = ClassName()
obj1
```

```
<__main__.ClassName at 0x1a0eb1d3d48>
```

Here obj1 is an object of class ClassName

OOPs – Classes and Objects

How to access Class Members?

Example

```
obj1 = ClassName()
obj2 = ClassName()
#Creating new instance attribute for obj2
obj2.variable = "I was just created"
print(obj1.variable)
print(obj2.variable)
print(ClassName.variable)
obj1.function()
```

```
I am a class Attribute
I was just created
I am a class Attribute
I am from inside the class
```

- Here obj1 and obj2 are object of class ClassName
- To access the members of a Python class, we use the dot operator.

OOPs – Classes and Objects

__init__() method in Python

Example

```
class Student(object):

    def __init__(self, name, branch, year):
            self.name = name
            self.branch = branch
            self.year = year
            print("A student object is created.")

    def print_details(self):
        print("Name:", self.name)
        print("Branch:", self.branch)
        print("Year:", self.year)
```

```
ob1= Student( "Paul","CSE",2019)
ob1.print_details()
```

```
A student object is created.
Name: Paul
Branch: CSE
Year: 2019
```

- __init__ is a special method in Python classes is a constructor method for a class
- __init__ is called when ever an object of the class is constructed

OOPs – Classes and Objects

Demo Test 1

Create two new vehicles called car1 and car2. Set car1 to be a red convertible worth $70,000.00 with a name of Ferrari, and car2 to be a blue van named JEEP worth $15,000.00.

```
# define the Vehicle class
class Vehicle:
        name = ""
        kind = "car“
        color = ""
        value = 100.00
        def description(self):
                desc_str = "%s is a %s %s worth $%.2f." %
                (self.name, self.color, self.kind, self.value)
                return desc_str
# your code goes here
……..
print(car1.description())
print(car2.description())
```

Demo: Classes and Objects

Magic Methods

Magic Methods

In Python, special methods are a set of predefined methods you can use to enrich your classes. They are easy to recognize because they start and end with double underscores.

__init__

__len__

__str__

Allows you to define a constructor which initializes an object

```
class Fruit:
        def __init__(self):
                print("I'm a fruit")
class Citrus(fruit):
        def __init__(self):
                super().__init__()
                print("I'm citrus")
lemon = Citrus()
```

Magic Methods

In Python, special methods are a set of predefined methods you can use to enrich your classes. They are easy to recognize because they start and end with double underscores.

__init__

__len__

__str__

It allows you to use len function on an object.

```
class Fruit:
        def __len__(self):
                return 10
print(len(Fruit()))
```

Magic Methods

In Python, special methods are a set of predefined methods you can use to enrich your classes. They are easy to recognize because they start and end with double underscores.

__init__

__len__

__str__

Allows you to define a string representation of an object

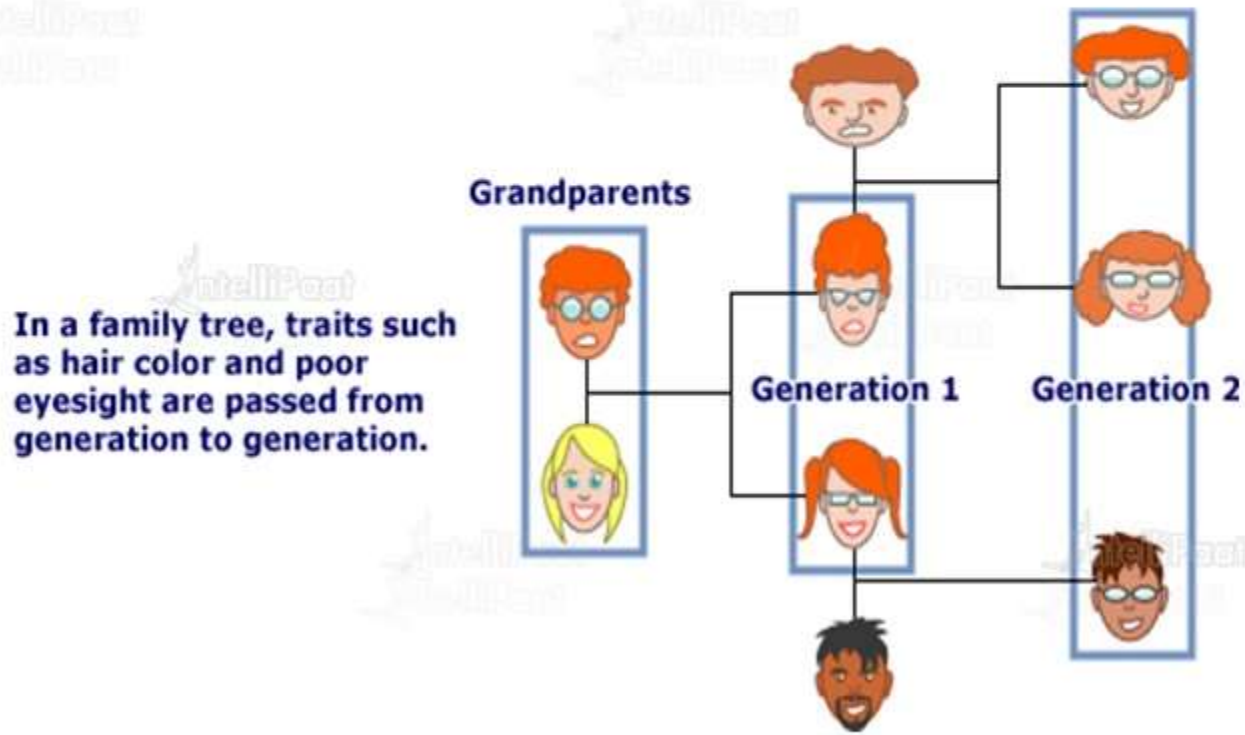
```
class Fruit:
        def __str__(self):
                return "I'm a fruit"
print(Fruit())
```

Demo: Magic Methods

Inheritance in Python

Inheritance in Python

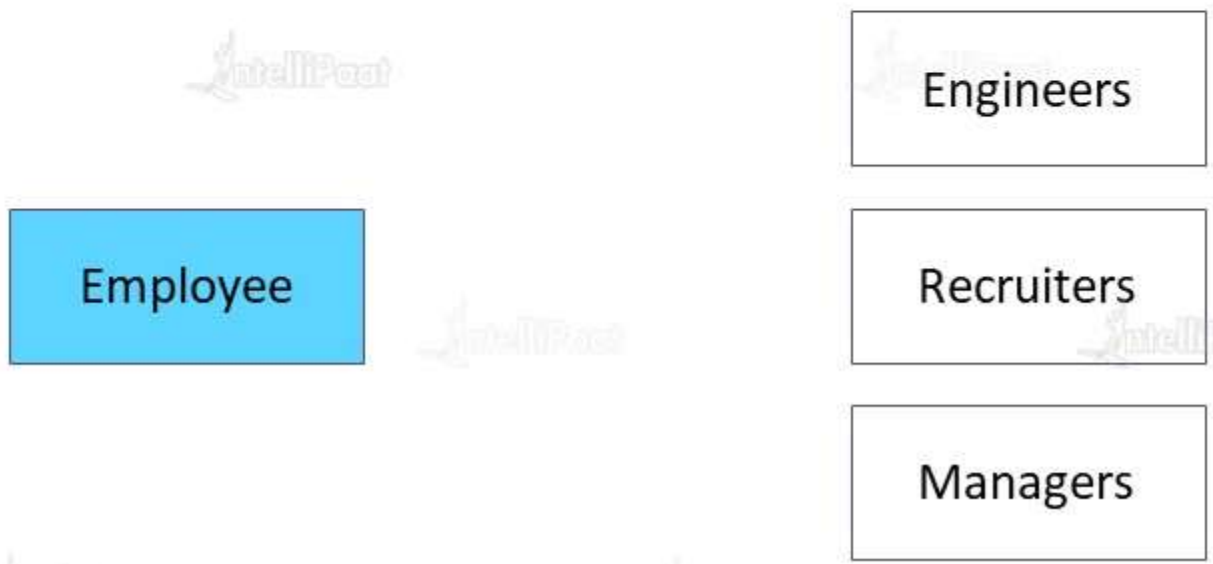
One class acquiring the property of another class. For example, you would have inherited few qualities from your parents.



Inheritance in Python

Suppose you have a class named Employee in your code base which stores information like id, name, age etc.

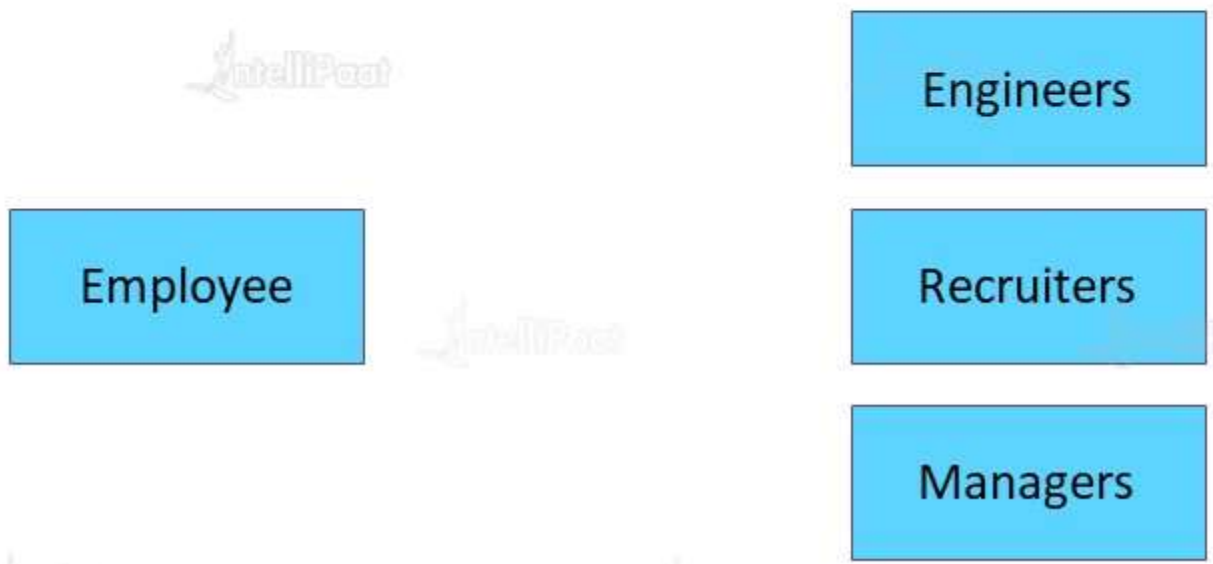
Now your codebase evolves and needs to store different kinds of employees like engineers, recruiters, managers etc.



Inheritance in Python

One way you could do that is by copying and pasting the code in employee class that in all the other classes of engineers, recruiters etc.

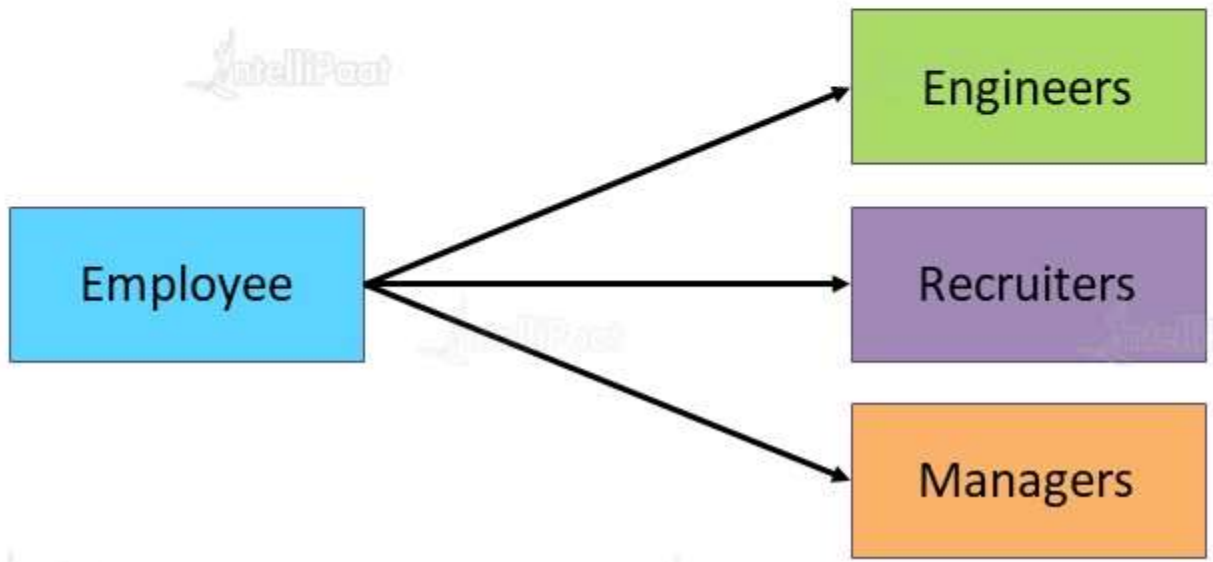
But then if we make any change to the employee class then all the other classes need to be updated



Inheritance in Python

On the other hand we can use inheritance to keep classes in sync by having other classes inherit from the employee class

These classes can now evolve with employee class and a developer only needs to make changes to the employee class



Inheritance in Python

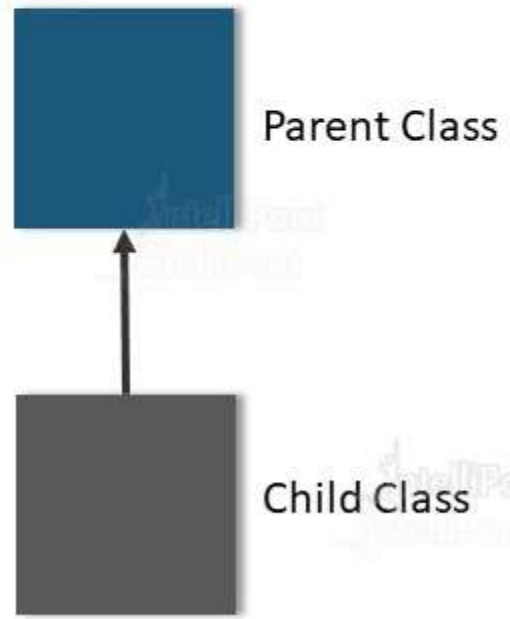
Different Types of Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Single class inherits from a class



Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

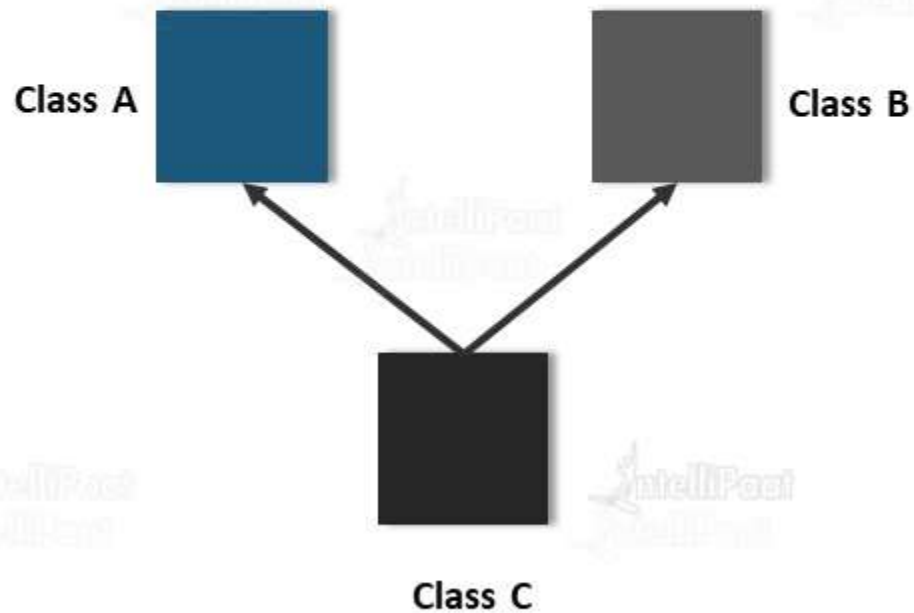
Single class inherits from a class

```
class fruit:
        def __init__(self):
                print("I'm a fruit")
class citrus(fruit):
        def __init__(self):
                super().__init__()
                print("I'm citrus")
Lemon = citrus()
```

Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

A class inherits from multiple classes



Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

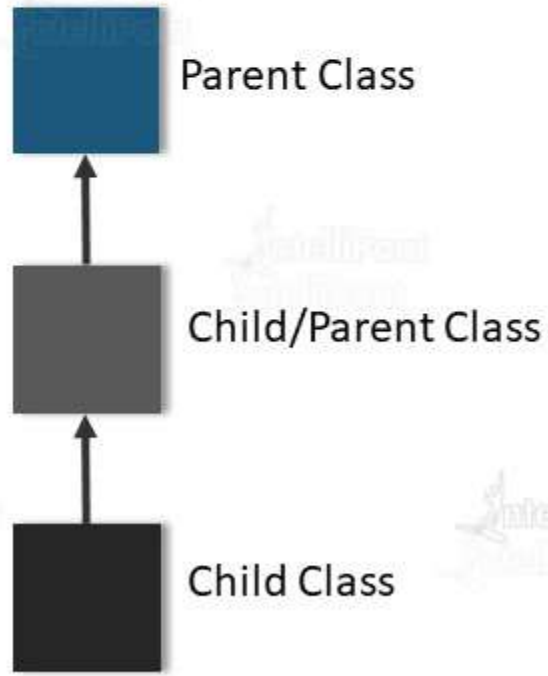
A class inherits from multiple classes

```
class A:
    pass
class B:
    pass
class C(A,B):
    pass
issubclass(C,A) and issubclass(C,B)
```

Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- **Multilevel Inheritance**
- Hierarchical Inheritance
- Hybrid Inheritance

One class inherits from a class, which will inherit from another class



Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

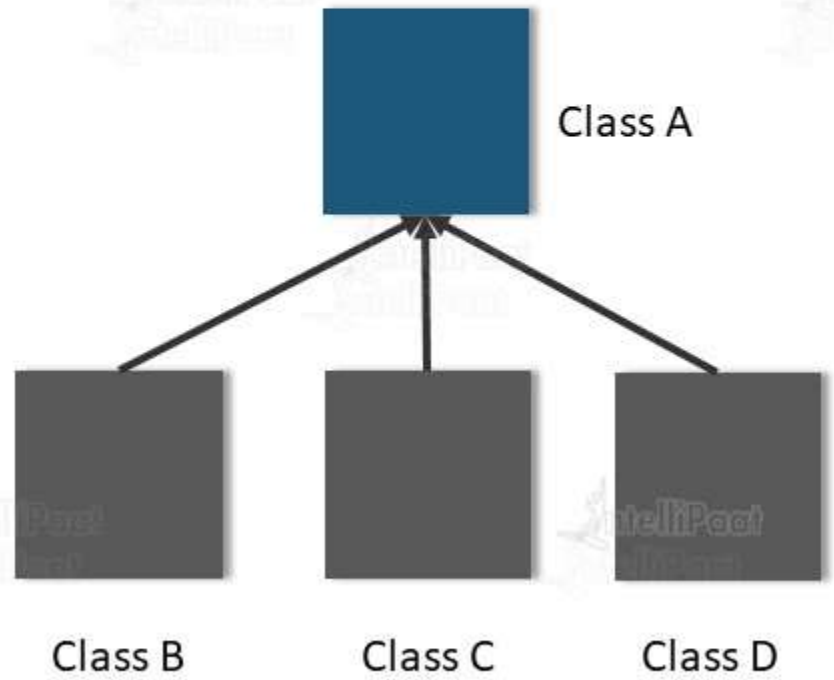
One class inherits from a class, which will inherit from another class

```
class A:
        x=1
class B(A):
        pass
class C(B):
        pass
cobj=C()
cobj.x
```

Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- **Hierarchical Inheritance**
- Hybrid Inheritance

More than one class inherits from a class



Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- **Hierarchical Inheritance**
- Hybrid Inheritance

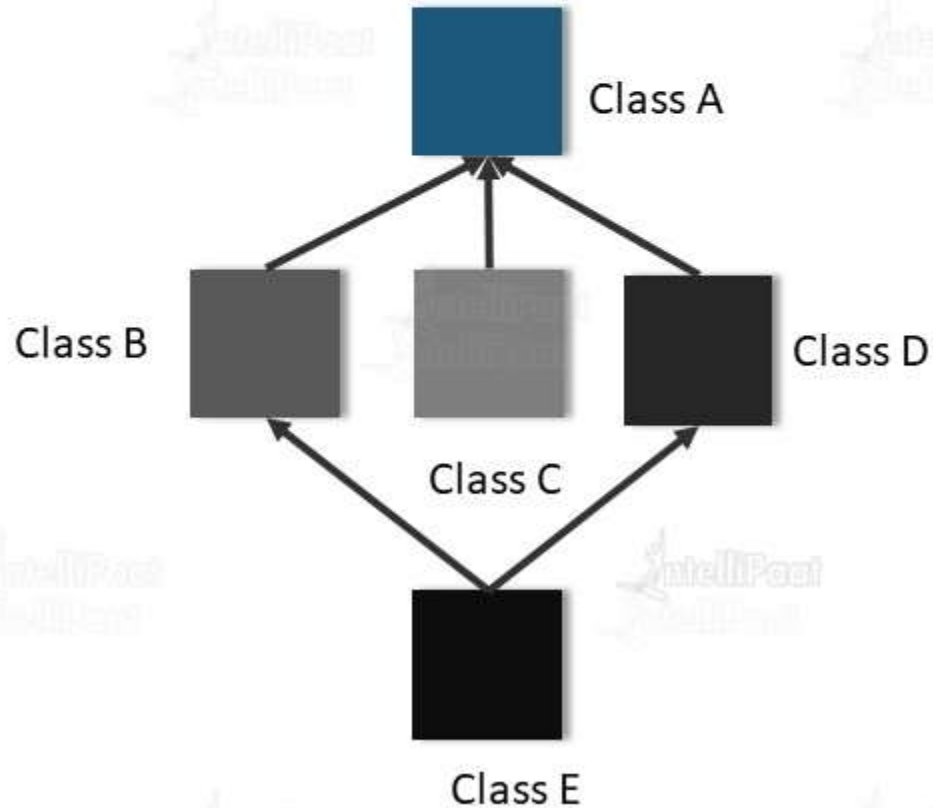
More than one class inherits from a class

```
class A:
        pass
class B(A):
        pass
class C(A):
        pass
issubclass(B,A) and issubclass(C,A)
```

Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Combination of any two kinds of inheritance



Inheritance in Python

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Combination of any two kinds of inheritance

```
>>> class A:
        x=1
>>> class B(A):
        pass
>>> class C(A):
        pass
>>> class D(B,C):
        pass
>>> dobj=D()
>>> dobj.x
```

Inheritance in Python

Inheritance Super Function

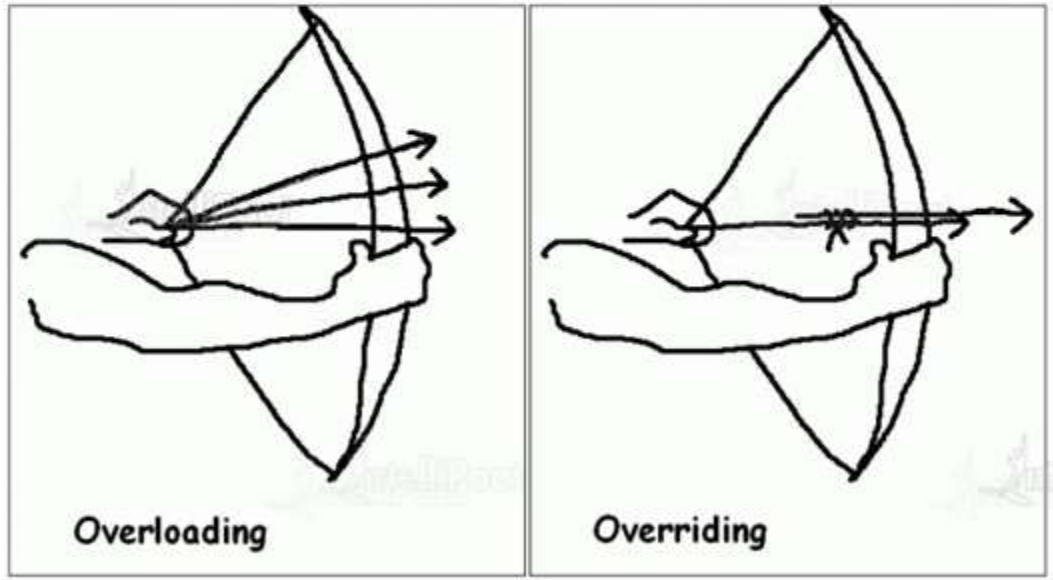
Used to call a method from the parent class

```
class Vehicle:
        def start(self):
            print("Starting engine")
        def stop(self):
            print("Stopping engine")
class TwoWheeler(Vehicle):
        def say(self):
            super().start()
            print("I have two wheels")
            super().stop()
Harley=TwoWheeler()
Harley.say()
```

Inheritance in Python

Overriding vs Overloading

Developers sometimes get confused between them



Inheritance in Python

Overloading

Overriding

Same function with different parameters

Why overload a function?

```
def add(a,b):
    return a+b
def add(a,b,c):
    return a+b+c
add(2,3)
```

```
TypeError: add() missing 1
required positional argument: 'c'
```

Inheritance in Python

Overloading

Overriding

Same function with different parameters

How to overload a function?

```
def add(instanceOf,*args):
    if instanceOf=='int':
        result=0
    if instanceOf=='str':
        result=''
    for i in args:
        result+=i
    return result
add('int',3,4,5)
```

Inheritance in Python

Overloading

Overriding

Subclass may change the functionality of a Python method in the superclass

Overriding a function

```
class A:
        def sayhi(self):
                print("I'm in A")
class B(A):
        def sayhi(self):
                print("I'm in B")
bobj=B()
bobj.sayhi()
```

Demo: Inheritance

Encapsulation in Python

Encapsulation in Python

Encapsulation = Abstraction + Data Hiding. Abstraction is showing essential features and hiding non-essential features to the user.

While writing a mail you don’t know how things are actually happening in the backend

Encapsulation in Python

Sometimes when writing code there are certain fields and methods we do not wish for others who are using the class to be able to access

For example when creating a class for accessing a cloud database the users should not be able to access methods that build the request to be sent over the network

Encapsulation in Python

To overcome this issue we use Encapsulation. So that all the code related to a particular task is in one location and only important functionality is exposed for users to use

To restrict user access we use construct called access modifier such as public, private and protected

Encapsulation in Python

Public

Protected

Private

The public access modifier is what is applied to all python fields and methods, so if not specified then everything is public

Encapsulation in Python

Public

Protected

Private

The protected access modifier is used to restrict access to fields and methods by anyone using an object of the class, but classes that inherit these fields and methods can access the

Encapsulation in Python

Public

Protected

Private

To specify if a something is protected in python you prefix its name with a single underscore e.g. _name, _get_data etc.

Encapsulation in Python

Public

Protected

Private

The private access modifier is used to restrict access to fields and methods by anyone using an object of the class, even if these things inherited they cannot be accessed by the class

Encapsulation in Python

Public

Protected

Private

To specify if a something is private in python you prefix its name with a double underscore e.g. __name, __get_data etc.

Encapsulation in Python

How to access a Private Method?

```
class Car:
    def __init__(self):
        self.__updateSoftware()

    def drive(self):
        print ('driving')

    def __updateSoftware(self):
        print ('updating software')

redcar = Car()
redcar.drive()
redcar._Car__updateSoftware()
```

Private method can be called using
redcar._Car__updateSoftware()

Encapsulation in Python

How to access a Private Method?

To change the value of a private variable, a setter method is used

```
def setMaxSpeed(self,speed):
        self.__maxspeed = speed


redcar = Car()
redcar.drive()
redcar.__maxspeed = 10  # will not
change variable because its private
redcar.setMaxSpeed(320)
redcar.drive()
```

Demo: Encapsulation

Polymorphism in Python

Polymorphism in Python

Functions with same name, but functioning in different ways

You behave differently in front of elders, and friends. A single person behaves differently at different time

Polymorphism in Python

For example suppose you have a class named Animal which has a method named speak which prints “I am an animal” to the screen

Then you have a class named Dog which inherits from Animal but overrides the speak methods to print “I am a dog” to the screen

```
class Animal:
    def speak():
        print("I am an Animal")

    def walk():
        print("I am an Walking")

class Dog(Animal):
    def speak(self):
        print("I am a Dog")
```

```
animal = Animal
dog = Dog()
animal.speak()
dog.speak()
```

```
I am an Animal
I am a Dog
```

Polymorphism in Python

Polymorphism with a function

Example

```
def in_the_pacific(fish):
    fish.swim()

sammy = Shark()

casey = Clownfish()

in_the_pacific(sammy)
in_the_pacific(casey)
```

Demo: Polymorphism

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor