Hi guys, thanks for staying the same. This is lecture 11 and my name is Akshansh Rawat (+91 8384891269, akshanshofficail@gmail.com (mailto:akshanshofficail@gmail.com)). We are going to talk about the inheritance in OOP. You remember that we have already talked about the OOP in lecture number 10. This is the continuation of that.

Let's talk about the inheritance. You must have heard about the following quote-

class can't be inherited.

What does taht means? Class is something that can not be transferred to one generation to another. So inheritance means transferring something (most of the time properties/qualities and characterstics).

Let's take a realistic example- Suppose your dad (parent) has car and bike. You (child) will be eligible to use his property. Child is automatically eligible for using preperties of parent and can easily access (like - fortune).

Can you access the properties of your child? As per the laws, I don't think you can. That's the logic here too. Parent can't access what child has but child can access what parent has.

In inheritance, child class can acess all the properties of parent class but parent class can't access what child class has.

let's create a Class of Car Like we did in lecture 10.

In [201]:

```
#creating a class Car
class Car:
    #let's define init method with few common arguments
    def __init__(self,maker,model,year):
        self.maker=maker
        self.model=model
        self.year=year
        self.meter_reading=0
    #creating method for the name
    def proper_name(self):
        name= f"{self.maker} - {self.model}, {self.year}"
        return name.title()
    #creating another method to start the car
    def start(self):
        return f"{self.maker} =3 ==33 ===333"
    #creating another method to accelerate the car
    def speeding(self):
        return f"{self.maker} VRoom vrrommm > >> >>>> "
    #creating another method to apply break
    def breaking(self):
        return f"{self.maker} is applying break KHRRrrrr kchhhhh !!!!...Pfff"
    #creating a method to read (odo)meter
    def meter(self,mileage=0):
        #by default I passed 0 in mileage if you want
        #anything else you can pass it too
        self.mileage=mileage  #accessing mileage
        if self.mileage>=self.meter_reading:
            print (f"this car has {self.mileage} Km reading on meter right now.")

        else:
            print("Reading on meter can't be negative, please pass +ve valaue in me

    #a new method to add Km in the meter if you buy an old car in case
    def update_meter(self,km):
        self.km=km
        self.km+=self.mileage
        print (f"{self.maker}, {self.model} is an old car which had {self.mileage}
```

This is all we did in lecture 10, right. This was the class Car and all the functions/methods of that class Car.

Let's create another class

This class will be called battery and this is not the child class we were talking about. This is an independent class which will be needed later.

I am creating this class because I will create a child class of Electric Car that will access all the properties of parent class Car. Battery class will be needed there as Electric Car will need a battery. Electric car needs a batter, doesn't it? At least in my country it does.

In [202]:

```
class Battery:
    #this class doesn't depend on the parent class Car
    #we will clear this, just be with me and with code
    def __init__(self,battery_size=75):
        self.battery_size=battery_size

    #creating methods that tells ab out the battery
    def battery_info(self):
        return f"this car has an electric battery of {self.battery_size} KWh in the
```

Ladies and gentlemen, the moment we all were waiting for - : We are finally creating a child class

I'm gonna create a child class named ElectriCar and in which I'll inherit all the properties of class Car. To do so I've to pass parent class as an argument in child class.

Pass parent class as an argument to the child class so that child class can access parent class

In [203]:

```
class ElectricCar(Car):
    def __init__(self,maker,model,year):
        super().__init(maker,model,year)
        self.battery=Batter()
```

notice that super(). method is there;that is magic keyword that give our child class a superpower to access the properties of parent class. After super(). init method will be passed and all the arguments will be same as parent class. Note that no : in super().

super(). gives additional superpower to childclass to access properties of parent class

In [204]:

```
class ElectricCar(Car):
    def __init__(self,maker,model,year):
        super().__init__(maker,model,year)
        self.battery=Battery()

    def proper_name(self):
            return f"{self.maker} presents you all new {self.model}; {self.year}"
```

Note that proper_name method was also in the parent class. It will override it parent class method. Let's take a realistic example.

You have bike. You dad has car and bike. If you need car, there is no option- you've to take your dad's car (child is accessing parent property). In case, you need bike? What will you do? First you will use your own despite knowing that your dad has another bike too but you'll be using your bike. Because - "own feels own".

So we have done - 1) created a class named Car 2) created another class Battery 3) created another child class ElectricCar that will access Parent class Car and will use class Battery too

let's combined all code

In [205]:

```
#creating a class Car
class Car:
    #let's define init method with few common arguments
    def __init__(self,maker,model,year):
        self.maker=maker
        self.model=model
        self.year=year
        self.meter_reading=0
    #creating method for the name
    def proper_name(self):
        name= f"{self.maker} - {self.model}, {self.year}"
        return name.title()
    #creating another method to start the car
    def start(self):
        return f"{self.maker} =3 ==33 ===333"
    #creating another method to accelerate the car
    def speeding(self):
        return f"{self.maker} VRoom vrrommm > >> >>>> "
    #creating another method to apply break
    def breaking(self):
        return f"{self.maker} is applying break KHRRrrrr kchhhhh !!!!...Pfff"
    #creating a method to read (odo)meter
    def meter(self,mileage=0):
        #by default I passed 0 in mileage if you want
        #anything else you can pass it too
        self.mileage=mileage  #accessing mileage
        if self.mileage>=self.meter_reading:
            print (f"this car has {self.mileage} Km reading on meter right now.")

        else:
            print("Reading on meter can't be negative, please pass +ve valaue in me

    #a new method to add Km in the meter if you buy an old car in case
    def update_meter(self,km):
        self.km=km
        self.km+=self.mileage
        print (f"{self.maker}, {self.model} is an old car which had {self.mileage}


class Battery:
    #this class doesn't depend on the parent class Car
    #we will clear this, just be with me and with code
    def __init__(self,battery_size=75):
        self.battery_size=battery_size

    #creating methods that tells ab out the battery
    def battery_info(self):
        return f"this car has an electric battery of {self.battery_size} KWh in the


class ElectricCar(Car):
    def __init__(self,maker,model,year):
        super().__init__(maker,model,year)
        self.battery=Battery()
    def proper_name(self):
            return f"{self.maker} presents you all new {self.model}; {self.year}"
```

That's our all the code: Let's play with it now-

creating an instance/object of main class Car

In [206]:

```
car1=Car("Audi","a4",2018)
```

creating another car instance, this time I'll design Lamborghini, aventador, 2014

In [207]:

```
car2=Car("lamborghini","aventador",2014)
```

let's print each function/method of class car

using proper_name() method to know the name

In [208]:

```
car1.proper_name()
```

Out[208]:

```
'Audi - A4, 2018'
```

In [209]:

```
car2.proper_name()
```

Out[209]:

```
'Lamborghini - Aventador, 2014'
```

lets start the car by using start() method

In [210]:

```
car1.start()
```

Out[210]:

```
'Audi =3 ==33 ===333'
```

In [211]:

```
car2.start()
```

Out[211]:

```
'lamborghini =3 ==33 ===333'
```

let's accelerate our cars by using speeding() method

In [212]:

```
car1.speeding()
```

Out[212]:

```
'Audi VRoom vrrommm > >> >>>> '
```

In [213]:

```
car2.speeding()
```

Out[213]:

```
'lamborghini VRoom vrrommm > >> >>>> '
```

let's apply break by using breaking() method

In [214]:

```
car1.breaking()
```

Out[214]:

```
'Audi is applying break KHRRrrrr kchhhhh !!!!...Pfff'
```

In [215]:

```
car2.breaking()
```

Out[215]:

```
'lamborghini is applying break KHRRrrrr kchhhhh !!!!...Pfff'
```

Let's check meter of cars

In [216]:

```
car1.meter()
```

```
this car has 0 Km reading on meter right now.
```

note that I didn't passs any argument in this, if you look at the code you will come to know by default it was assigned to 0. If you pass anything it will automaticall use that readinf

In [217]:

```
car1.meter(45)
```

```
this car has 45 Km reading on meter right now.
```

In [218]:

```
car2.meter(60)
```

```
this car has 60 Km reading on meter right now.
```

now update the reading on the meter of both cars

In [219]:

```
car1.update_meter()
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-219-8aafcac97edf> in <module>
----> 1 car1.update_meter()

TypeError: update_meter() missing 1 required positional argument: 'km'
```

it asks you atleast 1 agrument from you as nothing was assigned to default, let's pass an argument

In [220]:

```
car1.update_meter(45)
```

```
Audi, a4 is an old car which had 45 km on it and now it has become 90
km
```

see car1 has alread 45 km on it and when you passed another 45, total reading has become 45+45 now

In [221]:

```
car2.update_meter(60)
```

```
lamborghini, aventador is an old car which had 60 km on it and now it
has become 120 km
```

so far we have used all the methods of the parent class Car

now dig into child class and see how it access parent class methods for it's own benefits

first I need to creat another instance for electric car. Let's create two instance/object

In [222]:

```
tesla1=ElectricCar("tesla","model-s",2017)
```

In [223]:

```
tesla2=ElectricCar("tesla","sedan", 2016)
```

using proper_name() method to know the name

In [224]:

```
tesla1.proper_name()
```

Out[224]:

```
'tesla presents you all new model-s; 2017'
```

In [225]:

```
tesla2.proper_name()
```

Out[225]:

```
'tesla presents you all new sedan; 2016'
```

note that parent class was not defining the proper_name is this way. So what did exactly happen?

child class ElectricCar has its own method proper_name() so it didn't use parent class method.

using independent Battery class in child class

In [226]:

```
tesla1.battery.battery_info()
```

Out[226]:

```
'this car has an electric battery of 75 KWh in the cabinet'
```

In [227]:

```
tesla2.battery.battery_info()
```

Out[227]:

```
'this car has an electric battery of 75 KWh in the cabinet'
```

look at the code, tesla1.battery.battery_info() means- tesla1 will look into Battery class and then look into batter_info inside that class

using start() method from parent class Car for child class ElectricCar

In [228]:

```
tesla1.start()
```

Out[228]:

```
'tesla =3 ==33 ===333'
```

In [229]:

```
tesla2.start()
```

Out[229]:

```
'tesla =3 ==33 ===333'
```

using speed method from parent class to start electric car

In [230]:

```
tesla1.speeding()
```

Out[230]:

```
'tesla VRoom vrrommm > >> >>>> '
```

using meter() from parent class to read meter of electric class

In [231]:

```
tesla1.meter()
```

```
this car has 0 Km reading on meter right now.
```

In [232]:

```
tesla2.meter()
```

```
this car has 0 Km reading on meter right now.
```

update meter method from parent class

In [233]:

```
tesla1.update_meter(5)
```

```
tesla, model-s is an old car which had 0 km on it and now it has becom
e 5 km
```

In [234]:

```
tesla2.update_meter(456)
```

```
tesla, sedan is an old car which had 0 km on it and now it has become
456 km
```

I hope you guys got it what I meant to say with this lecture note. Stay safe everyone, stay at home. Have a great day.

In [ ]: