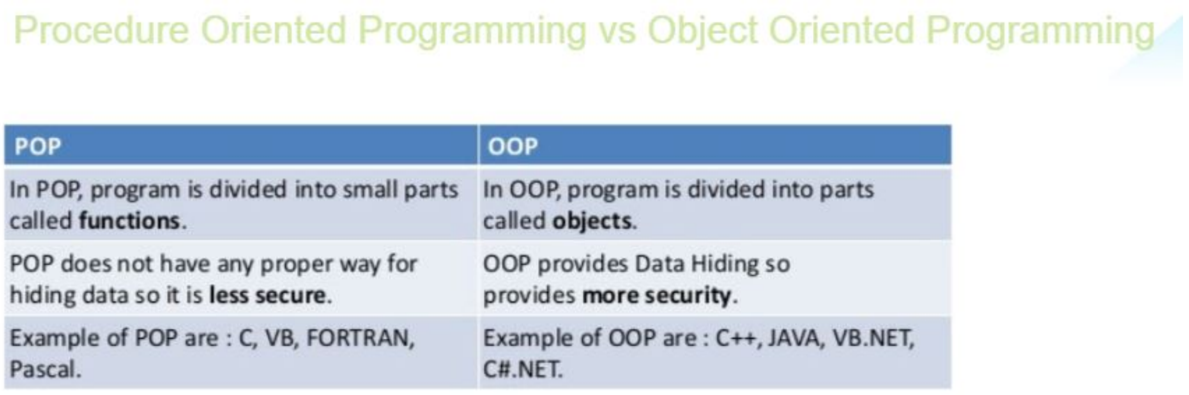
OBJECT ORIENTED PROGRAMMING

Object oriented programming is the methdology that promotes the effiecient way of programming using reusable compoents

Object oriented programming is divided into parts like objects.



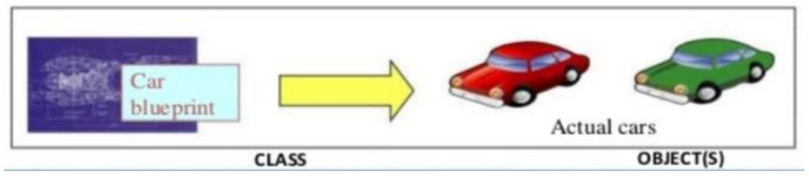
Procedure Oriented Programming vs Object Oriented Programming

POP	OOP
In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

CLASS

Class is a blueprint of objects

objects created by class can be simillar but not the same

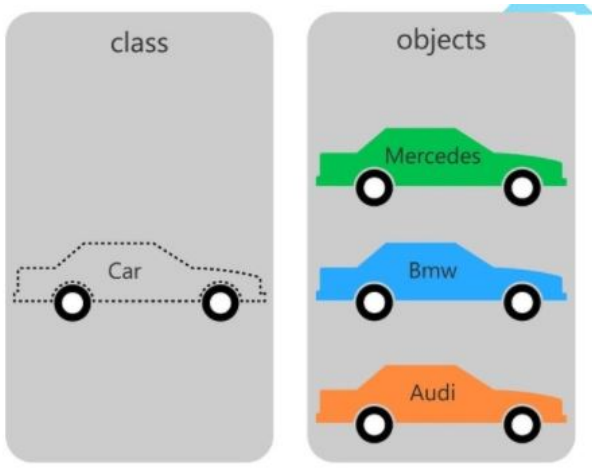


OBJECT

Objects is an intsance of class

In other words we can create multiple objects from the class

An object represents properties of an individual in the problem



In [2]:
```
#creation of  class

class employee():
    pass
```

In [3]:
```
#checking type on python

print(type(employee()))
```

```
<class '__main__.employee'>
```

ATTRIBUTES

- Here we created blueprint for the employee by creating class.
- Now we will create employees as object with some variables related to them called as **attributes**

In [6]:
```
# creating objects


emp1 = employee()
emp2 = employee()
```

- We can add attributes about the employee in two ways

1) Individually

2) In class

In [ ]:
```
#WAY 1 : Attributes adddditing for employee indiviualy
emp1.first = 'test1'
emp1.last = 'user'
emp1.email = 'test1.user@companymail.com'
emp1.pay = 30000
```

```
emp2.first = 'test2'
emp2.last = 'user'
emp2.email = 'test2.user@companymail.com'
emp2.pay = 30000

#checking of inforation
print(emp1.email)
print(emp2.email)
```

- if we add attributes like this it is very tedious and more or less similar to the solution to the bank problem using functions

- so instead we declare the attributes while creaing the object instance itself

- we can accept these attributes being passed in using the special **init** method
- Note: we will discuss the self keyword in some time

In [ ]:

```
#WAY 2 : attributes additing in class
class employee():

    def __init__(self,first,last,pay):
        self.first = first
        self.last = last
        self.pay = pay

emp1 =employee("user1","-test1",10000)
emp2 =employee("user2","-test2",20000)
emp3 =employee("user3","-test3",15000)
```

The `__init__` mehtod is called every time a new instance of a class is created It is also called as special method

- every time we create an instance it means we are creating an object of that class
- method is similar to a fucntion but inside a class
- special methods are inbuild funciton which has special meaning like the `__init__` methods which gets executed at the time of creation of a new object

---

ABOUT ACCESSING ATTRIBUTES:

- Acessing a attribute just like other built in objects in python

- we can write the object name follwed by a .(dot) then press tab key

- when accessing attributes we dont need () brackets after the attribute name

- when accessing method we need to add () brackets along with the parameters if any

- it is not compulsary to keep the name of the attribute the same as the one being passed in

---

---

self - keyword

- The self keyword tells the class that the attribbutes and the methods belong to that particular object
- in other word we are passing in the object itself to the methods
- whenever we call a method or an special method the self keyword is passed in by default and is hidden
- This can be verified by making a definig the inti method with no self

---

OPERATIONS ON ATTRIBUTES

- we can also do operation on the variables(attributes) being passed in or we can define an entirely new attributes which is not passed by the user.

In [8]:
```
class employee():
    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        self.email = first+'.'+last+'@companymail.com'
        self.post = "manager"
```

In [ ]:
```
emp1 = employee('test1','user',3000)
```

In [6]:
```
emp1.post
```

Out[6]:
```
'manager'
```

In [7]:
```
emp1.email
```

Out[7]:
```
'test1.user@companymail.com'
```

Functions(Methods)

- we can add more functions to perform multiple task

In [9]:
```
# Add different methods to make full name , post and emailid
# Here you can pass the company name as argument to make email id

class employee():
    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay



    def full_name(self):
        return self.first_name+' '+self.last_name


    def post(self):
        self.post = "manager"


    def email(self,company):
        self.company = company

        self.email = self.first_name+ "."+self.last_name+ "@"+self.company +".com"

        print(self.email)
```

In [12]:
```
emp1 = employee('test1','cat',3000)

emp2 =employee('test2','dog',1000)
```

In [13]:
```
emp2.email("xyz")    # method 1 of calling functions : object_name. function_name()
```

```
test2.dog@xyz.com
```

In [ ]:

REVISION

- 1) Understanding : OOP , class, object
- 2) creating class
- 3) Checking of class for its type
- 4) Creating objects
- 5) Attribution adding - a) Individually b) in class itself
- 6) Understanding : **init** , self
- 7) operations on attributes
- 8) additing methods - full name, post, email id

HOMEWORK

- 1) Add promotion method with customised promotion amt
- 2) Revise the concept of OOP, class and object

HOMEWORK SOULTION

In [14]:

```
class employee():
    def __init__(self,first,last,pay):
        self.first_name = first
        self.last_name = last
        self.salary = pay
        #self.email = first+'.'+last+'@companymail.com'


    def full_name(self):
        return self.first_name+' '+self.last_name


    def post(self):
        self.post = "manager"


    def email(self,company):
        self.company = company

        self.email = self.first_name+ "."+self.last_name+ "@"+self.company +".com"

        print(self.email)

    def promotion(self):

        hike = int(input("enter the  promotional amt"))

        self.salary_hike =int(self.salary* hike)

        print(self.salary_hike)
```

In [16]:

```
emp1 = employee("test","user",2000)
emp1.promotion()
```

```
enter the  promotional amt5
10000
```