Object Oriented Programming

CLASS: is a bluprint for creating objects.
OBJECTS:is an instance of a class which inherits its property from its class.

In [7]:
```python
class Dog:
    name = "Bruno"
    age = 15


d1 = Dog()
d2 = Dog()
d3 = Dog()
```

In [8]:
```python
d1.name, d2.name, d3.name
```

Out[8]:
```
('Bruno', 'Bruno', 'Bruno')
```

In [18]:
```python
class Dog:
    def __init__(self,name,age):

        self.name = name
        self.age = age

d1 = Dog('Bruno',15)
d2 = Dog('Charlie',10)
```

In [14]:
```python
d1.name
```

Out[14]:
```
'Bruno'
```

In [17]:
```python
d2.name
```

Out[17]:
```
'Charlie'
```

In [19]:
```
dir(d1)
```

Out[19]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'age',
 'name']
```

In [24]:
```
class Dog:
    def __init__(self,name,age):

        self.dog_name = name
        self.dog_age = age

d1 = Dog('Bruno',15)
d2 = Dog('Charlie',10)
```

In [25]:
```
d1.dog_name
```

Out[25]:
```
'Bruno'
```

self is a keyword which points to the current instance of a class

In [35]:
```python
class Dog:
    tail = True      # class variable
    def __init__(self,name,age):
        self.dog_name = name
        self.dog_age = age

    def bark(self):
        print('Bhau! Bhau!')

    def run(self,steps):
        print(f"{self.dog_name} ran {steps} steps")

d1 = Dog('Bruno',15)
d2 = Dog('Charlie',10)
```

In [31]:
```python
d1.bark()
```

```
Bhau! Bhau!
```

In [33]:
```python
d1.run(500)
```

```
Bruno ran 500 steps
```

In [34]:
```python
d2.run(400)
```

```
Charlie ran 400 steps
```

In [38]:
```python
d1.tail, d2.tail
```

Out[38]:
```
(True, True)
```

In [55]:
```python
# create a Employee class which  takes
# 4 inputs from the user:

class Employee:
    company = 'Tata'
    def __init__(self,f,l,a,p='Software Engineer'):
        self.fname = f
        self.lname = l
        self.age = a
        self.profession = p

    def printname(self):
        print(f"The employee name is {self.fname} {self.lname}")

    def promotion(self,profile):
        self.profession = profile
        print(f"The profession updated to {self.profession}")


e1 = Employee('Nishant','Singh',20)
e2 = Employee('Vijay','Deverakonda',30,'Data Scientist')
```

In [56]:
```
e1.profession
```
Out[56]:
```
'Software Engineer'
```

In [57]:
```
e2.profession
```
Out[57]:
```
'Data Scientist'
```

In [58]:
```
e1.printname()
```
```
The employee name is Nishant Singh
```

In [59]:
```
e2.printname()
```
```
The employee name is Vijay Deverakonda
```

In [60]:
```
e1.promotion("data engineer")
```
```
The profession updated to data engineer
```

In [62]:
```
e1.profession
```
Out[62]:
```
'data engineer'
```

Inheritance

Inheritance is a process of inheriting a parents property into its childs class
Types if inheritance in python :

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

In [80]:
```python
# simple inheritance

class Person:
    def __init__(self,fname,lname):
        self.fname = fname
        self.lname = lname

    def printinfo(self):
        print("First Name : ",self.fname)
        print("Last Name :", self.lname)

    def run(self,dis, speed = 2):
        print("Time taken",dis/speed)

class Employee(Person):
    def __init__(self,profile,fname,lname):
        self.profile = profile
        Person.__init__(self,fname,lname)

e1 = Employee('Engineer','Shravan','Kumar')
```

In [81]:
```python
e1.profile
```

Out[81]:
```
'Engineer'
```

In [82]:
```python
e1.fname
```

Out[82]:
```
'Shravan'
```

In [83]:
```python
e1.printinfo()
```

```
First Name :  Shravan
Last Name : Kumar
```

In [84]:
```python
e1.run(200)
```

```
Time taken 100.0
```

Method Overriding

In [86]:
```python
def sample():
    print("Hello")

def sample():
    print('Bye')

sample()
```

```
Bye
```

In [91]:
```
# simple inheritance

class Person:
    def __init__(self,fname,lname):
        self.fname = fname
        self.lname = lname

    def printinfo(self):
        print("First Name : ",self.fname)
        print("Last Name :", self.lname)

    def run(self,dis, speed = 2):
        print("Time taken",dis/speed)

class Employee(Person):
    def __init__(self,profile,fname,lname):
        self.profile = profile
        Person.__init__(self,fname,lname)

    def run(self, dis, speed = 5):
        print("Time taken",dis/speed)

e1 = Employee('Engineer','Shravan','Kumar')
```

In [92]:
```
e1.run(300)
```

```
Time taken 60.0
```

In [93]:
```
p1 = Person('Rahul','Singh')
p1.run(300)
```

```
Time taken 150.0
```

Multiple Inheritance

In [103]:
```
class Car:
    def __init__(self,brand,model):
        self.brand = brand
        self.model = model
class Electric:
    def __init__(self,battery):
        self.battery = battery

class HybridCar(Car,Electric):    # multiple Inheritance
    def __init__(self,b,m,bat):
        Car.__init__(self,b,m)
        Electric.__init__(self,bat)

hc1 = HybridCar('Tata','Nexon',"35000MaH")
```

In [104]:
```
hc1.battery
```
Out[104]:
```
'35000MaH'
```

In [105]:
```
hc1.model
```
Out[105]:
```
'Nexon'
```

Multilevel Inheritance

In [106]:
```
class Car:
    def __init__(self,brand,model):
        self.brand = brand
        self.model = model

    @staticmethod
    def printinfo():
        print("This is a simple car class")

class Electric(Car):
    def __init__(self,battery):
        self.battery = battery

class HybridCar(Electric):    # multiple Inheritance
    def __init__(self,bat):
        Electric.__init__(self,bat)

hc1 = HybridCar(35000)
```

In [107]:
```
hc1.battery
```
Out[107]:
```
35000
```

In [108]:
```
hc1.printinfo()
```
```
This is a simple car class
```

In [109]:
```
hc1.model
```
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\BHUPEN~1\AppData\Local\Temp/ipykernel_14984/330512167.py in <module>
----> 1 hc1.model

AttributeError: 'HybridCar' object has no attribute 'model'
```

Method Overloading

In [113]:
```
class A:
    def summ(self,a,b):
        print(a+b)

obj = A()
obj.summ(10,11)
```

```
21
```

In [114]:
```
class A:
    def summ(self,a=0,b=0):     # overloading
        print(a+b)

obj = A()
obj.summ(10,11)
```

```
21
```

In [115]:
```
obj.summ(10)
```

```
10
```

In [116]:
```
obj.summ()
```

```
0
```

In [119]:
```
class A:
    def summ(self,a=0,b=0):
        print(a+b)

class B(A):

    def summ(self,a,b):     # method overriding
        print(a+b)

obj = A()
obj.summ(10,11)
```

```
21
```

In [122]:
```
obj2 = B()

obj2.summ(10)     # obj2.summ() will only work with 2 arguments
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\BHUPEN~1\AppData\Local\Temp/ipykernel_14984/673925444.py in <module>
      1 obj2 = B()
      2 
----> 3 obj2.summ(10)     # obj2.summ() will only work with 2 arguments

TypeError: summ() missing 1 required positional argument: 'b'
```

A lot more to learn and explore!

Keep Exploring!

In [ ]:

```

```