

OBJECT  
= ORIENTED  
PROGRAMMING

# \* CREATING CLASSES:-

\* Class Dog (objects):  $\Rightarrow$  It is not compulsory to put ()  $\Rightarrow$  objects in parenthesis.  
def \_\_init\_\_(self):  $\Rightarrow$  We have to define it by giving double underscore & putting self means the constructor will do it.  
print('Nice You Have a Dog')  
def speak(self):  
pass

HM = dog()  $\Rightarrow$  HM is known as instance of type dog or class Dog  
AM = dog()  
 $\gg$  Nice You Have a Dog  
Nice You Have a Dog  
Let's add Attribute to it.

\* class Dog(object):  
def \_\_init\_\_(self, name):  
self.name = name  
def speak(self):  
print("Hi I am", self.name)

HM = Dog('HM')

AM = Dog('AM')

HM.speak()

AM.speak()

$\gg$  Hi I am HM

Hi I am AM

We can create how much method we want to create.

But they has to be ~~self~~ self.



# \* INNER INHERITANCE :-

\* class Dog (object):

```
def __init__(self, name, age):
```

```
    self.name = name
```

```
    self.age = age
```

```
def speak(self):
```

```
    print("Hi I am", self.name, "my age is", self.age, "years old")
```

class Cat (Dog):

now if we create a new class & we want some property of the old class, then we just have to put the class name in parenthesis. And everything will be sorted out.

```
def __init__(self, name, age, color):
```

```
    super().__init__(name, age)
```

```
    self.color = color
```

```
tim = Cat('tim', 5, 'blue')
```

```
tim.speak()
```

>>

Hi I am tim my age is 5 years old.

\* How to change something from Dog class that we don't want in Cat class.

\* class Dog(object):

[ Some as previous ]

```
def talk(self self):  
    print('Bark')
```

But we don't want "Bark" in Cat class then, what we do is.

class Cat(Dog):

[ Some previous ]

```
def talk(self):  
    print('Meow')
```

```
tim = Cat('Tim', 5, 'Blue')
```

~~tim = Cat~~

```
tim.talk()
```

So, anything we will ~~change~~ overwrite in cat class will ~~not~~ be print. ~~Not~~ dog. class.  
from the

So, it will overwrite that method.

⇒ Inheritance is simple means putting the attributes of different class.

More of Inheritance

# OVERLOADING METHODS :-

```
* class Point():
```

```
    def __init__(self, x=0, y=0):
```

```
        self.x = x
```

```
        self.y = y
```

```
        self.coords = (self.x, self.y)
```

```
    def move(self, x, y):
```

```
        self
```

```
        self.x += x
```

```
        self.y += y
```

```
    def __add__(self, p):
```

```
        return Point(self.x + p.x, self.y + p.y)
```

```
    def __sub__(self, p):
```

```
        return Point(self.x - p.x, self.y - p.y)
```

```
    def __mul__(self, p):
```

```
        return self self.x * p.x + self.y * p.y
```

```
    def __str__(self):
```

```
        return "(" + str(self.x) + ',' + str(self.y) + ")"
```

```
p1 = Point(3, 4)
```

```
p2 = point(3, 2)
```

```
p3 = point(1, 3)
```

```
p4 = point(0, 1)
```

```
p5 = p1 + p2
```

```
p6 = p4 - p1
```

```
p7 = p3 * p4
```

`print(p5, p6, p7)`

`>>`

`(6,6) (-3,-3) 4`



# Comparing Methods

# \*STATICS METHODS AND CLASS METHODS:-

class Dog:

```
def __init__(self, name):  
    self.name = name  
    self.dog.append(self)
```

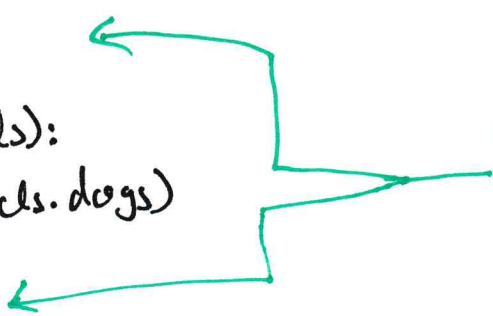
@classmethod

```
def num_dogs(cls):  
    return len(cls.dogs)
```

@staticmethod

```
def bark(n):  
    """ barks n times """  
    for _ in range(n):  
        print('bark')
```

these are known  
as decorators



~~Dog~~ Dog.bark(5)

>>

Bark

Bark

Bark

Bark

Bark

## \* PRIVATE AND PUBLIC CLASS:-

If we have anything like "\_" before that or dot underscore before, then it means typically they are PRIVATE.

Eg:- \_Private:

Now to run this we have created 2 pages or with mod.py & another with tutorial1.py. [You choose create what name you want to create.]

mod.py

```
class _Private:  
    def __init__(self, name):  
        self.name = name
```

class NotPrivate:

```
    def __init__(self, name):  
        self.name = name  
        self.priv = _Private(name)
```

```
    def _display(self):  
        print("Hello")
```

```
    def display(self):  
        print("Hi")
```

## tutorial 1.py

```
import mod
```

```
from mod import NotPrivate
```

```
test = NotPrivate('Hm')
```

```
test._display()
```

>> Hello

• if we put at the end line

```
test._display()
```

>> Hi

[ so, it means we can actually  
show private method as well.  
Just the thing is they are private