Constructors in Python ¶

- Class functions that begins with double underscore **(__)** are called special functions as they have special meaning.
- Of one particular interest is the

  `__init__()`

  function. This special function gets called whenever a new object of that class is instantiated.
- This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to **initialize** all the variables.

Example:

In [13]:

```
class Bird():
    def __init__(self):
        self.x = 0
        self.y = 0
    def fly_up(self):
        self.y += 1
```

In [17]:

```
bird1 = Bird()

print(bird1.x , bird1.y)

bird1.fly_up()

print(bird1.x , bird1.y)
```

```
0 0
0 1
```

In [21]:

```
# Making multiple objects from a class
class Bird():
    def __init__(self):
        self.x = 0
        self.y = 0
    def fly_up(self):
        self.y += 1

birds = [Bird() for x in range(0,3)]
for i in birds:
    print(i)
```

```
<__main__.Bird object at 0x7f6a4037db00>
<__main__.Bird object at 0x7f6a4037da20>
<__main__.Bird object at 0x7f6a4037da58>
```

Refining the Bird class

Accepting paremeters for the init() method

In [22]:

```
class Bird():
    def __init__(self,x=0,y=0):
        #Each bird has position (x,y)
        self.x = x
        self.y = y
    def fly_up(self):
        self.y += 1


bird1 = Bird(10,20)
print(bird1.x , bird1.y)
bird1.fly_up()
print(bird1.x , bird1.y)
```

```
10 20
10 21
```

In [26]:

```
class Bird():
    def __init__(self,x=0,y=0):
        #Each bird has position (x,y)
        self.x = x
        self.y = y
    def fly_up(self):
        self.y += 1

# Making Group of birds
birds = []
birds.append(Bird())
birds.append(Bird(10,20))
birds.append(Bird(15,22))

# Printing position of each bird
for index,obj in enumerate (birds):
    print("Bird {} is flying at position: ({},{})".format(index,obj.x,obj.y))
```

```
Bird 0 is flying at position: (0,0)
Bird 1 is flying at position: (10,20)
Bird 2 is flying at position: (15,22)
```

Accepting paremeters in a method

In [30]:

```
class Bird():
    def __init__(self,x=0,y=0):
        #Each bird has position (x,y)
        self.x = x
        self.y = y
    def fly(self,x_increment =0,y_increment =0):
        self.x += x_increment
        self.y += y_increment

# Making Group of birds
birds = []
birds.append(Bird())
birds.append(Bird(10,20))
birds.append(Bird(15,22))

# Printing position of each bird
for index,obj in enumerate (birds):
    print("Bird {} is flying at position: ({},{})".format(index,obj.x,obj.y))

# Moving Birds
birds[0].fly(5,2)
birds[1].fly(4,5)
birds[2].fly(3,8)

# Printing updated position of each bird
print("Updated Positions Are:")
for index,obj in enumerate (birds):
    print("Bird {} is flying at position: ({},{})".format(index,obj.x,obj.y))
```

```
Bird 0 is flying at position: (0,0)
Bird 1 is flying at position: (10,20)
Bird 2 is flying at position: (15,22)
Updated Positions Are:
Bird 0 is flying at position: (5,2)
Bird 1 is flying at position: (14,25)
Bird 2 is flying at position: (18,30)
```

Adding a new method

New method performs that calculation, and then returns the resulting distance.

In [38]:

```
from math import sqrt
class Bird():
    def __init__(self,x=0,y=0):
        #Each bird has position (x,y)
        self.x = x
        self.y = y
    def fly(self,x_increment =0,y_increment =0):
        self.x += x_increment
        self.y += y_increment
    def get_distance(self, other_bird):
        # Calculates the distance from this bird to another bird,
        #  and returns that value.
        distance = sqrt((self.x-other_bird.x)**2+(self.y-other_bird.y)**2)
        return distance

# create two bird at different place
bird1 = Bird(10,20)
bird2 = Bird(50,25)

# show distance between them
distance = bird1.get_distance(bird2)
print("Distance between two birds is: ",distance)
```

```
Distance between two birds is:  40.311288741492746
```

Descructor

The destructor is defined using

```
__del__(self).
```

In [1]:

```
class TestClass:

    def __init__(self):
        print ("constructor")

    def __del__(self):
        print ("destructor")
```

In [10]:

```
obj = TestClass()
```

```
constructor
destructor
```

In [9]:

```
print(obj)
```

```
<__main__.TestClass object at 0x7f6a403fdb70>
```

In [11]:

```
del obj
```

```
destructor
```

In [12]:

```
obj  #shows error since obj is deleted
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-12-139c6c212d9f> in <module>()
----> 1 obj

NameError: name 'obj' is not defined
```

In [ ]:

```

```