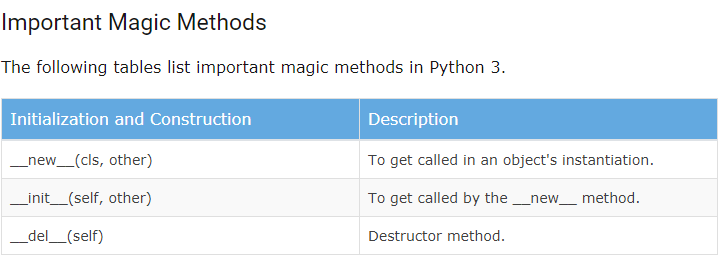
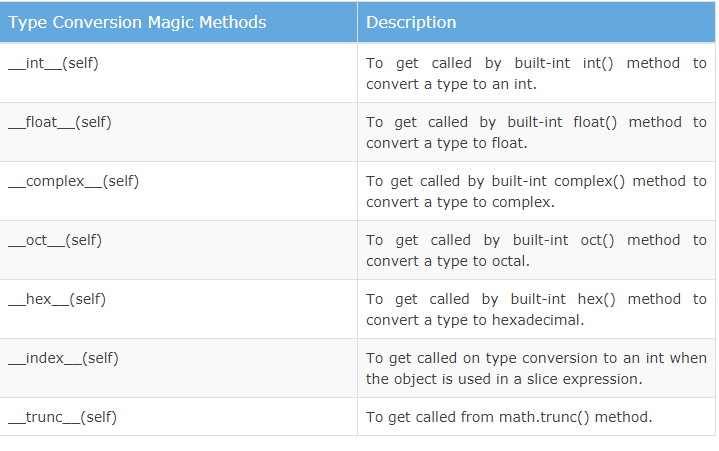
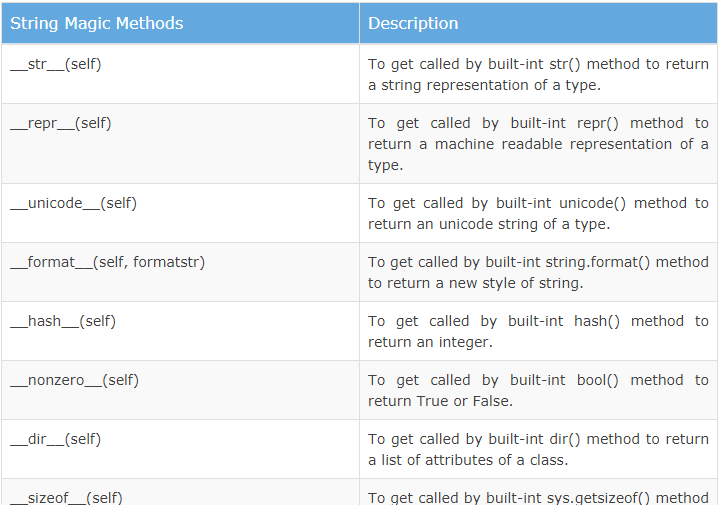
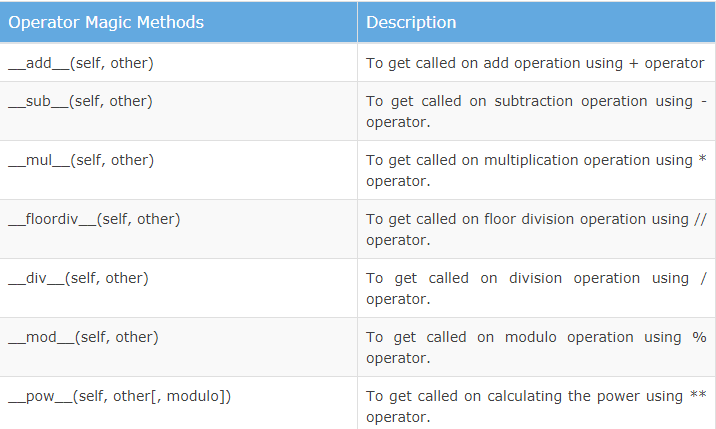
Magic method

Dunder or **magic methods in Python** are the **methods** having two prefix and suffix underscores in the **method** name. Dunder here means “Double Under (Underscores)”. These are commonly used for operator overloading. Few examples for **magic methods** are: \_\_init\_\_, \_\_add\_\_, \_\_len\_\_, \_\_repr\_\_ etc.









\_\_new\_\_() method

In Python the \_\_new\_\_() magic method is implicitly called before the \_\_init\_\_() method. The \_\_new\_\_() method returns a new object, which is then initialized by \_\_init\_\_().

class employee:

    def \_\_new\_\_(cls):

        print ("\_\_new\_\_ magic method is called")

        inst = object.\_\_new\_\_(cls)

        return inst

    def \_\_init\_\_(self):

        print ("\_\_init\_\_ magic method is called")

        self.name='Satya'

e1=employee()

## \_\_str\_\_() method

Another useful magic method is \_\_str\_\_(). It is overridden to return a printable string representation of any user defined class. We have seen str() built-in function which returns a string from the object parameter. For example, str(12) returns '12'. When invoked, it calls the \_\_str\_\_() method in the int class.

class employee:

    def \_\_init\_\_(self):

        self.name='Swati'

        self.salary=10000

    def \_\_str\_\_(self):

        return 'name='+self.name+' salary=$'+str(self.salary)

e1=employee()

print(e1)

## \_\_add\_\_() method

the magic method \_\_add\_\_() is overridden, which performs the addition of the points attributes of the two objects. The \_\_str\_\_() method returns the object's string representation.

class Point:

    def \_\_init\_\_(self, x = 0, y = 0):

        self.x = x

        self.y = y

    def \_\_str\_\_(self):

        return "({0},{1})".format(self.x,self.y)

    def \_\_add\_\_(self,other):

        x = self.x + other.x

        y = self.y + other.y

        return Point(x,y)

#creating first point object

p1=Point(10,20)

#creating second point object

p2=Point(30,40)

p3=p1+p2

print('Point1 =',p1)

print('Point2= ',p2)

print('Sum of Points= ',p3)

## \_\_ge\_\_() method

The following method is added in the distance class to overload the >= operator.

class Point:

    def \_\_init\_\_(self, x = 0, y = 0):

        self.x = x

        self.y = y

    def \_\_ge\_\_(self,other):

        val1=self.x+self.y

        val2=other.x+other.y

        if val1>=val2:

            return True

        else:

            return False

#creating first point object

p1=Point(10,20)

#creating second point object

p2=Point(30,40)

print(p1>=p2)