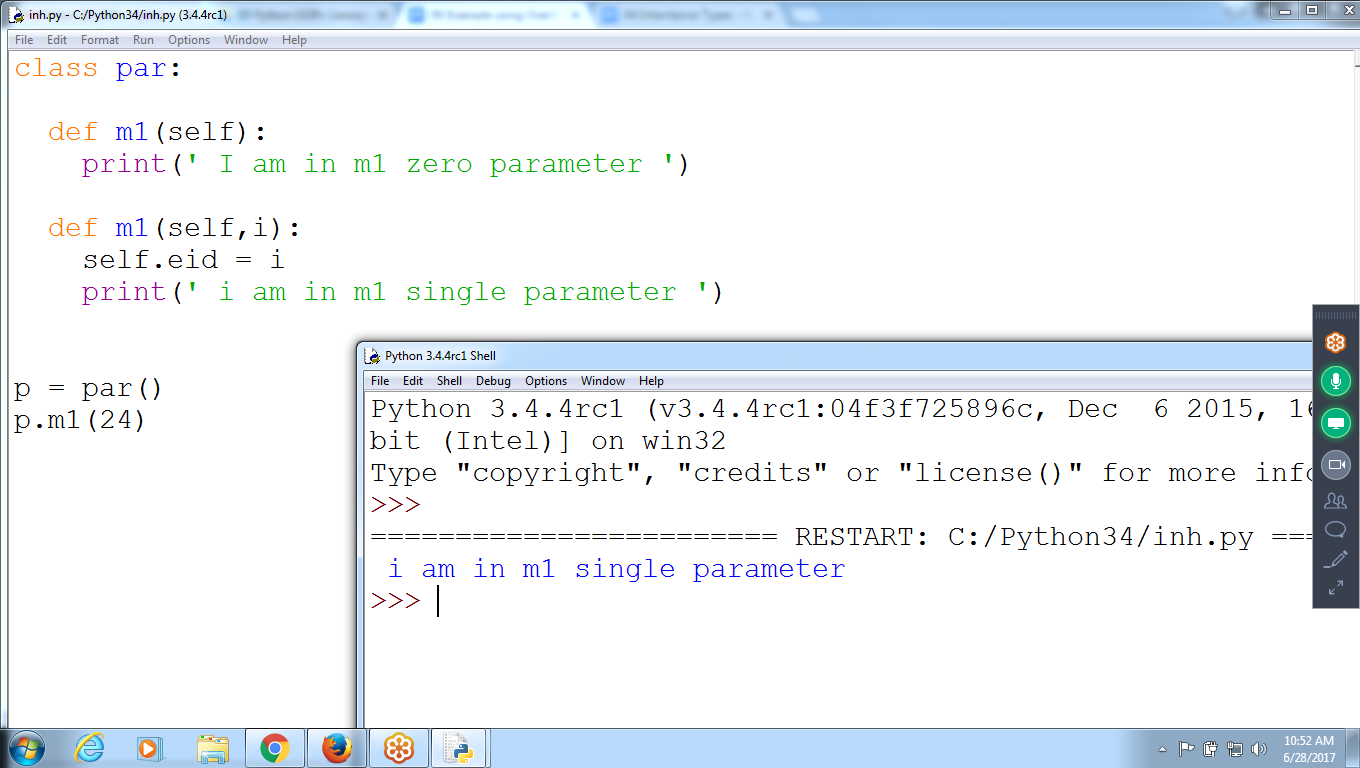
**Polymorphism :**: Same Method name, multiple Definitions

Within the class multiple Definitions



class par:

def m1(self):

print(' I am in m1 zero parameter ')

def m1(self,i):

self.eid = i

print(' i am in m1 single parameter ')

p = par()

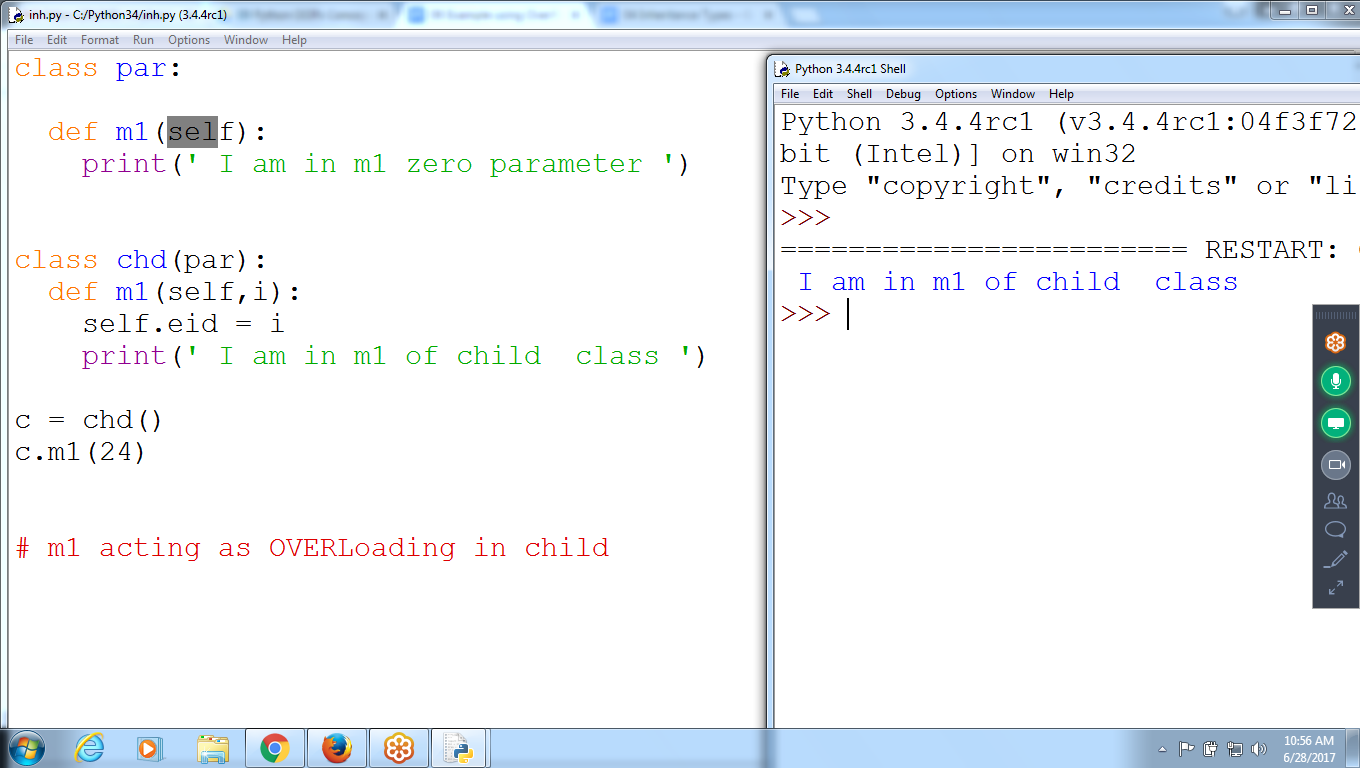
p.m1(24)

**Overloading ::**

Same Method name : Multiple Definitions inherited from parent is **Overloaded**

Same Method Name multiple Definitions with in same class :: **Polymorphism**

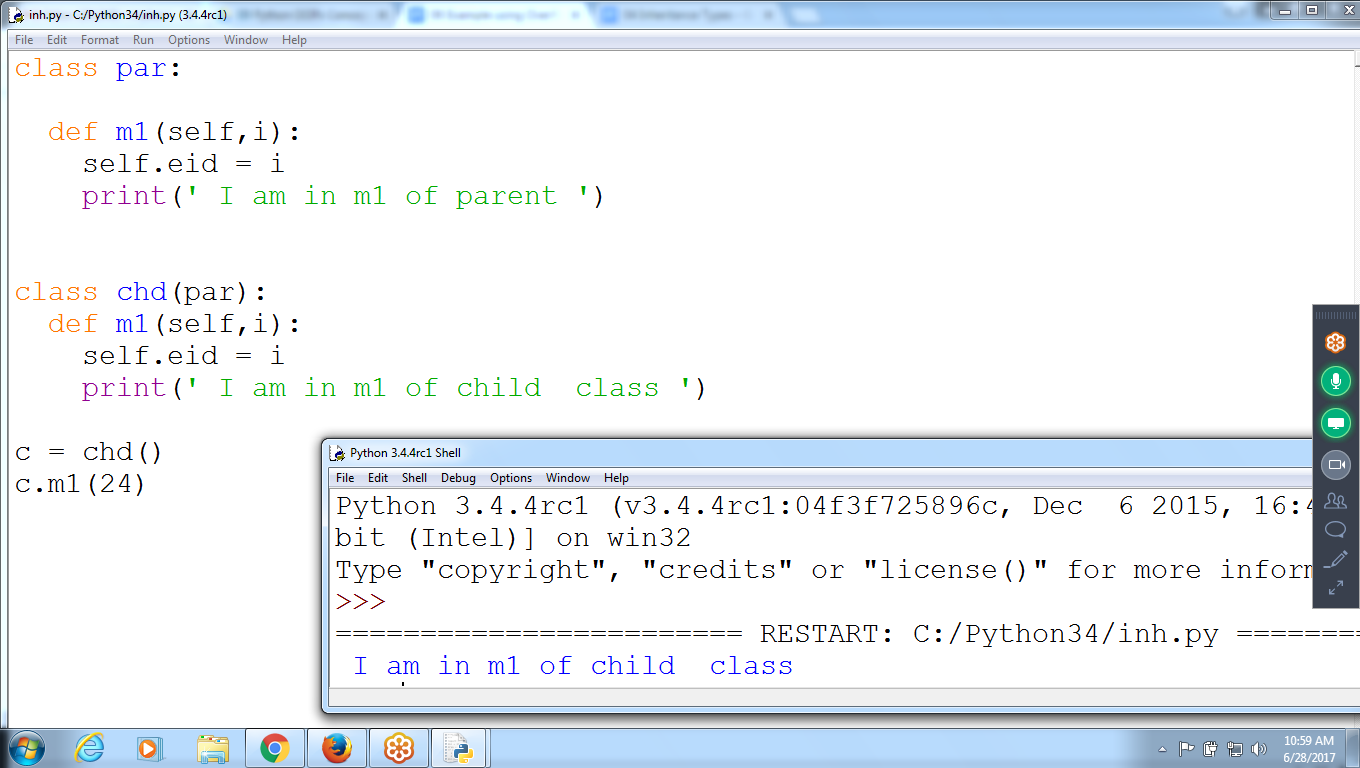
M1 is overloaded with two definitions m1(self) from child and m1(self, i) from parent



**Overriding :**

Same Method Name and Same Number of Parameters inherited from Parent

Note:: Not Calling parent Method



class par:

def m1(self,i):

self.eid = i

print(' I am in m1 of parent ')

class chd(par):

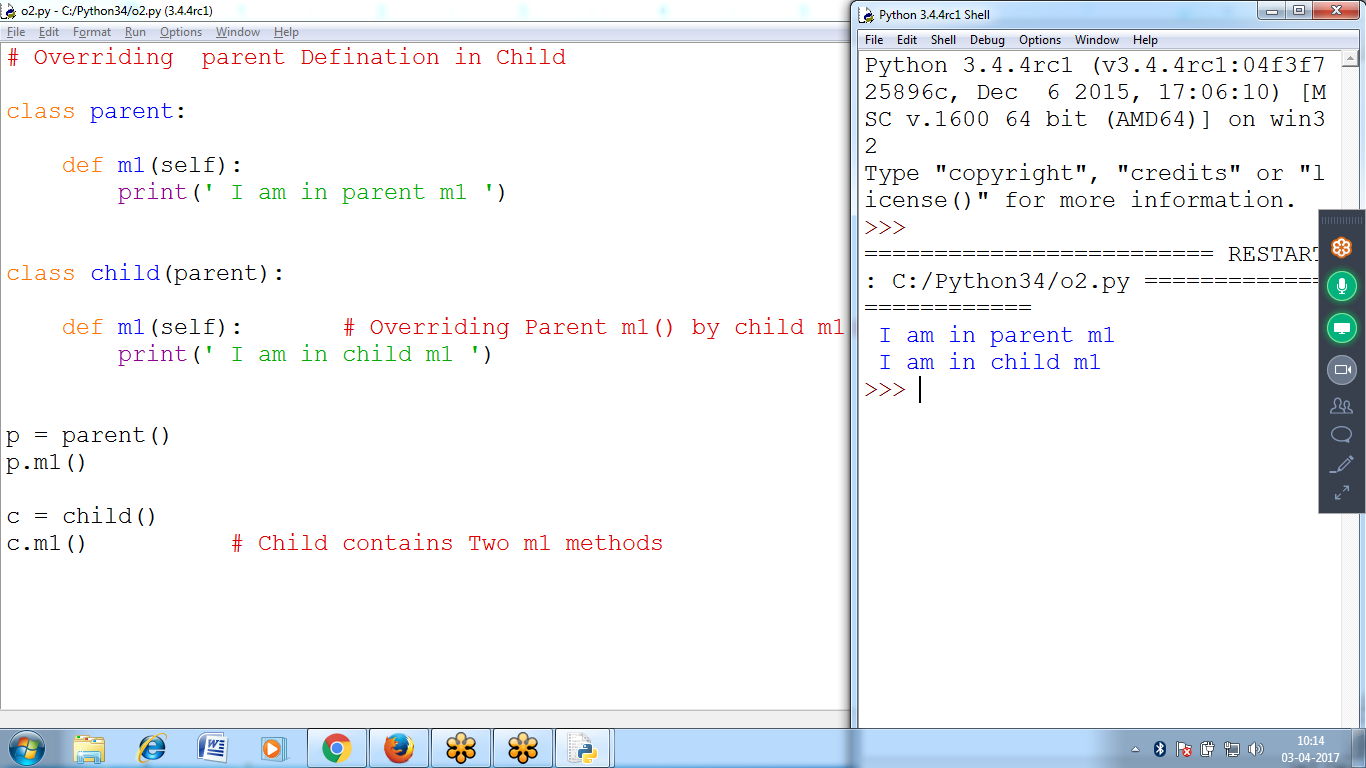
def m1(self,i):

self.eid = i

print(' I am in m1 of child class ')

c = chd()

c.m1(24)



# Overriding parent Definition in Child

class parent:

def m1(self):

print(' I am in parent m1 ')

class child(parent):

def m1(self): # Overriding Parent m1() by child m1()

print(' I am in child m1 ')

p = parent()

p.m1()

c = child()

c.m1() # Child contains Two m1 methods

Integer Addition, String Addition ( Concatenation)

>>> a =24

>>> b = 24

>>> c = a+b **# + on integer variable**

>>> c

48

>>> st1 = 'python'

>>> st2 = 'program'

>>> st1+st2 **# String Concatenation ( + on Strings )**

'Pythonprogram'

>>> ls1 = ['raju', 'rani', 'siav']

>>> ls2 = ['123', 'abc', '456']

>>> ls1 + ls2 **# + on list**

['raju', 'rani', 'siav', '123', 'abc', '456']

>>>

>>> c = a \*b

>>> c **# Integer Multiplication**

1056

>>> st3 = st1 \* st2 **# string \* string ::: Not allowed**

Traceback (most recent call last):

File "<pyshell#10>", line 1, in <module>

st3 = st1 \* st2

TypeError: can't multiply sequence by non-int of type 'str'

>>> st3 = st1 \* 2 **# String Replica string \* integer :: String replica**

>>> st3

'pythonpython'

>>>

**Special Methods**

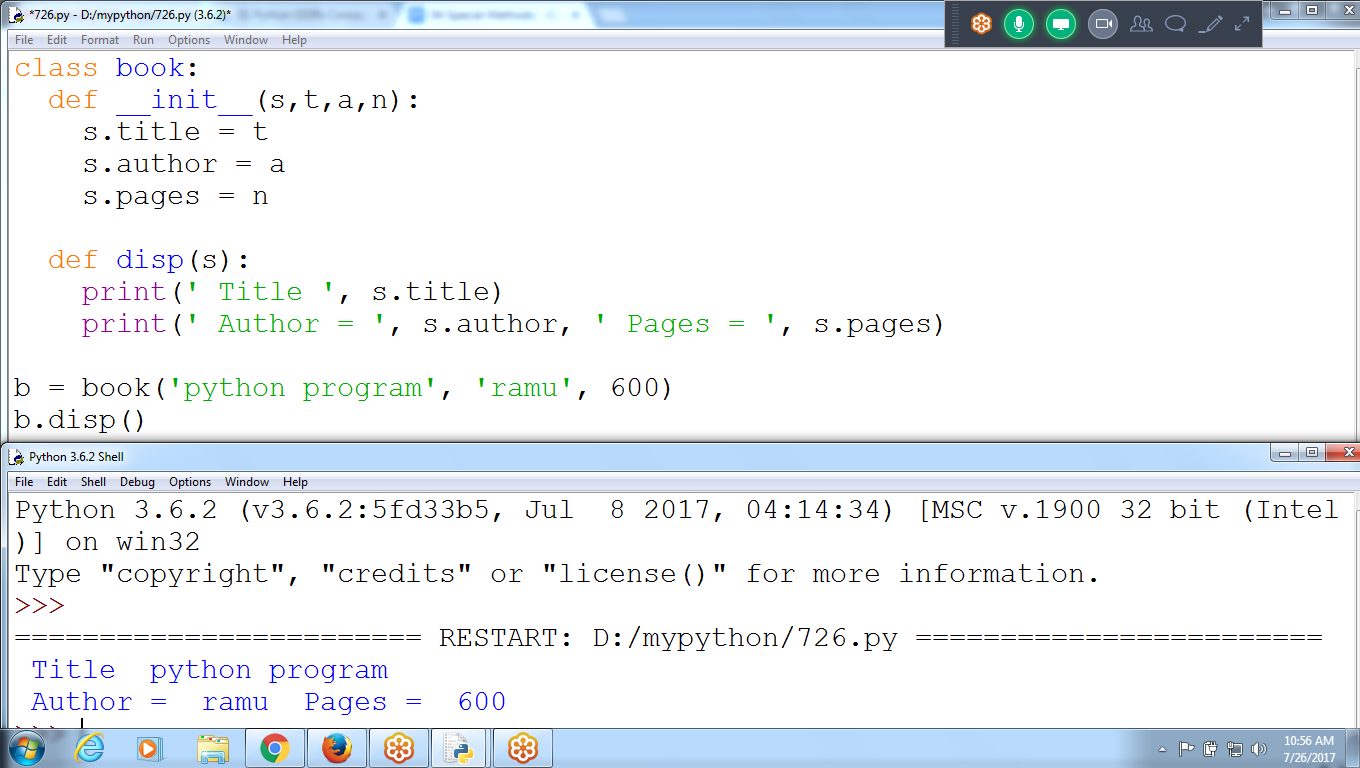
**\_\_init\_\_ vs \_\_call\_\_ vs \_\_del\_\_**

the \_\_init\_\_ method is used when the *class* is called to initialize the instance, while the \_\_call\_\_ method is called when the *instance* is called

with **call** you can redefine the same object as if it were a new, **init** is called once by **new** when object is created, so that it can be initialized

Create BOOK class and object

\_\_init\_\_() is an constructor, which fires automatically at the time of object creation



class book:

def \_\_init\_\_(s,t,a,n):

s.title = t

s.author = a

s.pages = n

def disp(s):

print(' Title ', s.title)

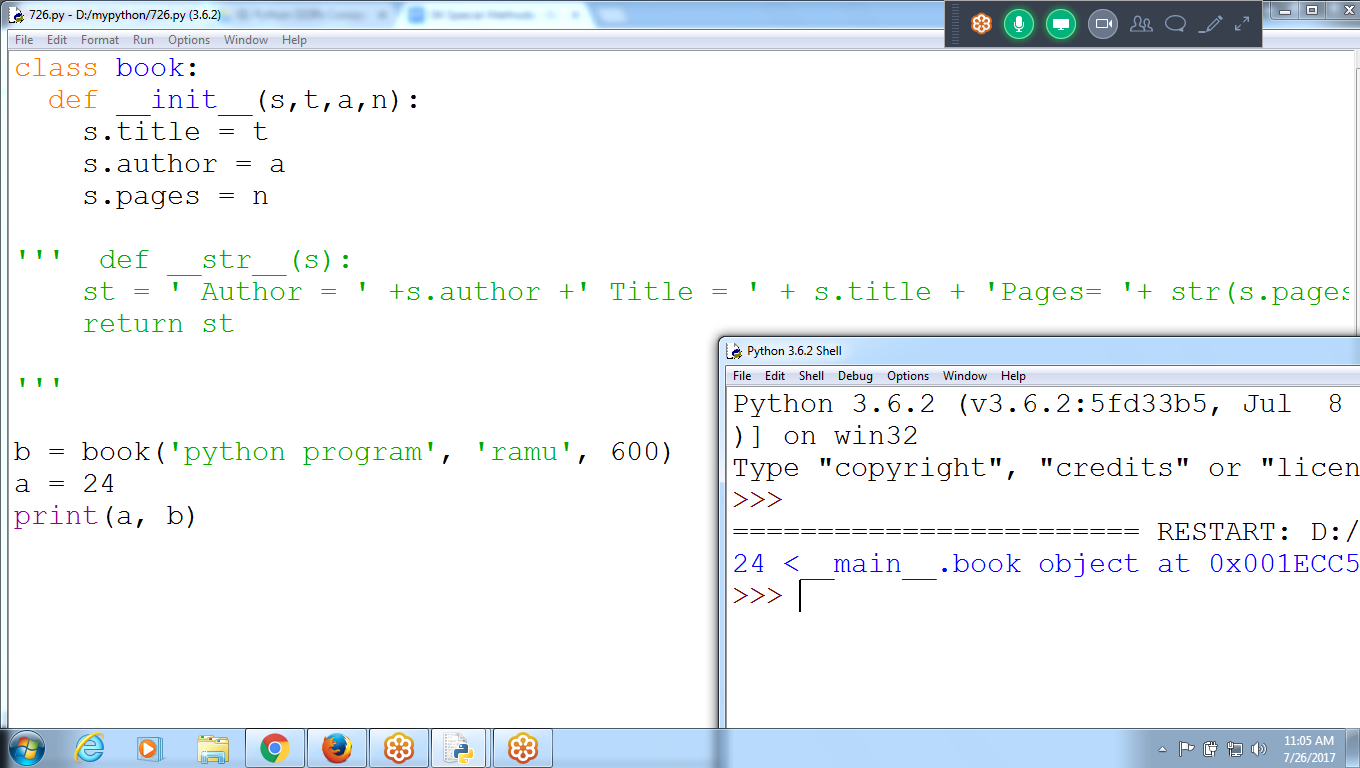
print(' Author = ', s.author, ' Pages = ', s.pages)

b = book('python program', 'ramu', 600)

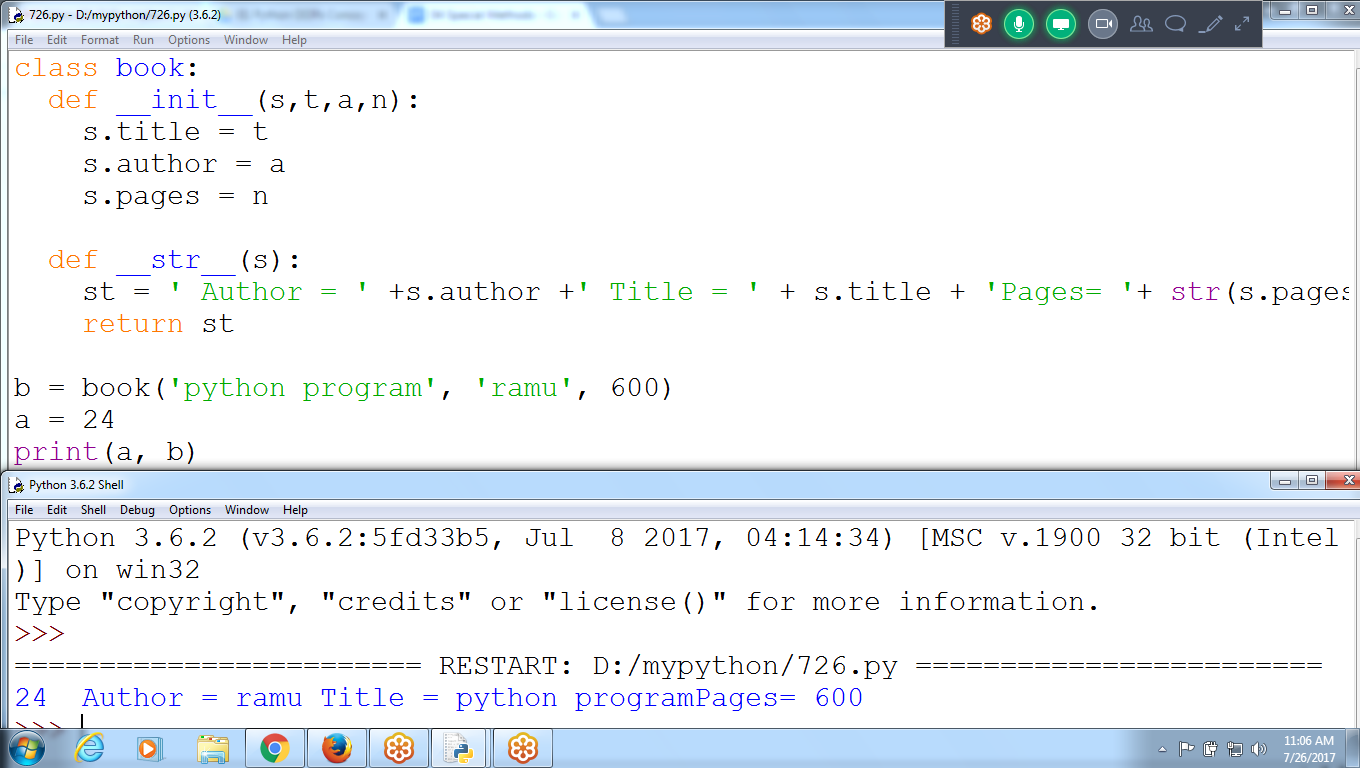
b.disp()

\_\_str\_\_() :: Replacing definition for print()

Value a prints, value b printing as Memory address

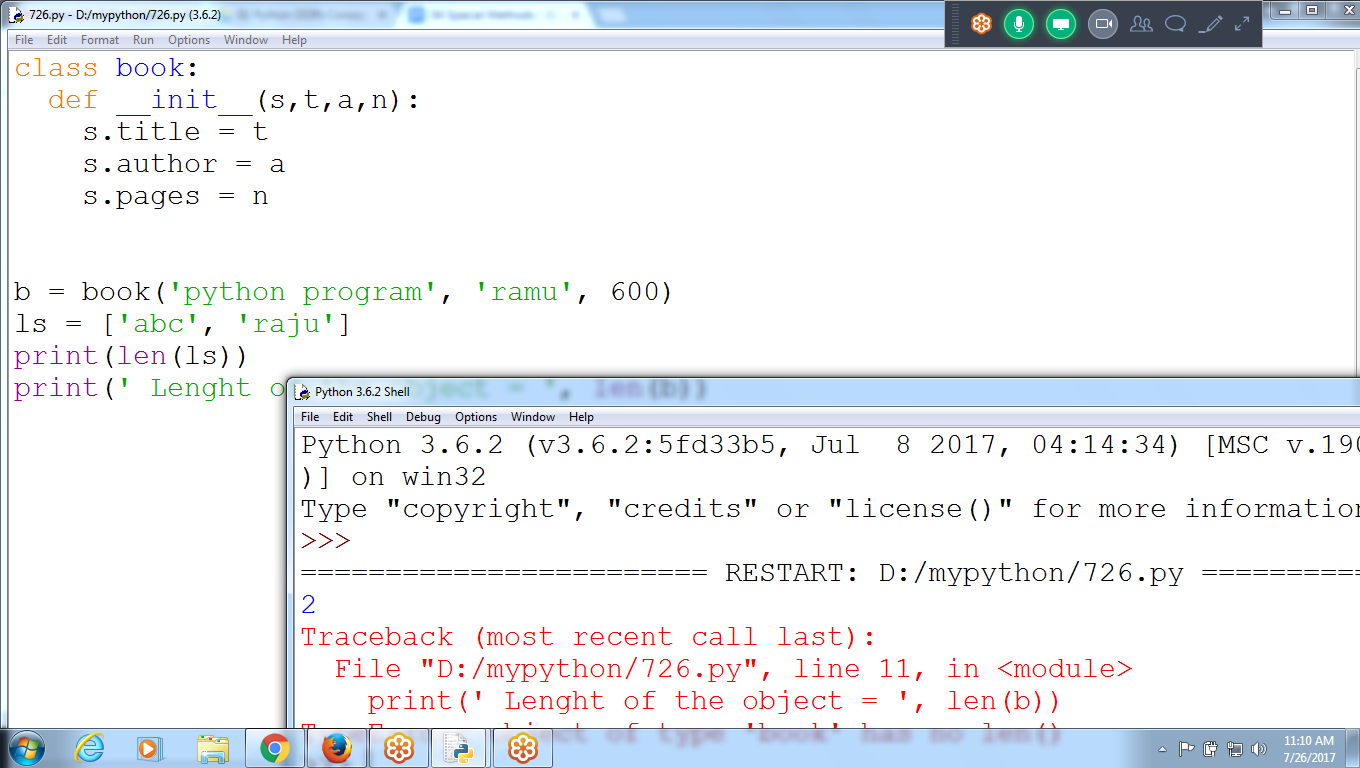


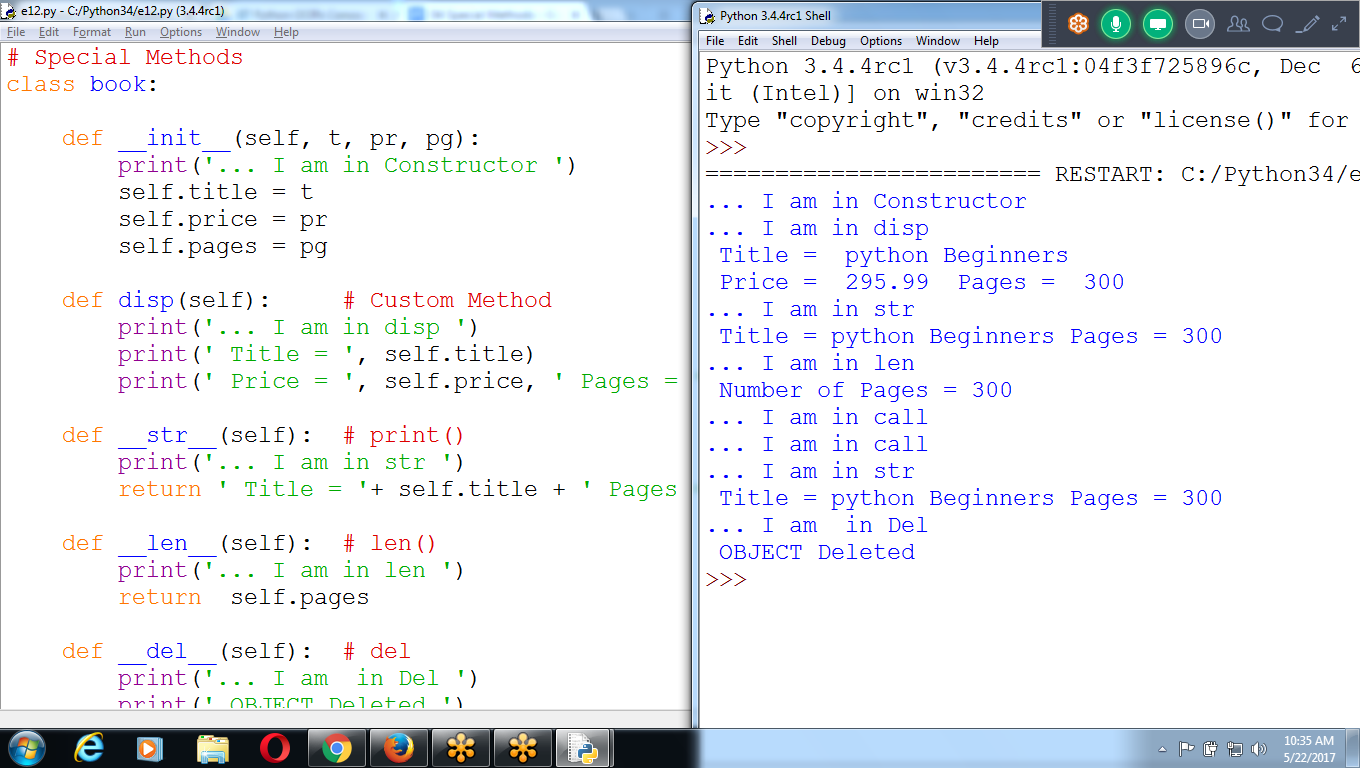
Replacing print() definition



\_\_len\_\_()

Can Print list data type values, but not for object





# Special Methods

class book:

def \_\_init\_\_(self, t, pr, pg):

print('... I am in Constructor ')

self.title = t

self.price = pr

self.pages = pg

def disp(self): # Custom Method

print('... I am in disp ')

print(' Title = ', self.title)

print(' Price = ', self.price, ' Pages = ', self.pages)

def \_\_str\_\_(self): **# print()**

print('... I am in str ')

return ' Title = '+ self.title + ' Pages = ' + str(self.pages)

def \_\_len\_\_(self): **# len()**

print('... I am in len ')

return self.pages

def \_\_del\_\_(self): **# del**

print('... I am in Del ')

print(' OBJECT Deleted ')

def \_\_call\_\_(self, t, pr, pa): **# Reassign values**

print('... I am in call ')

b = book('python Beginners', 295.99, 300)

b.disp()

print(b)

print(' Number of Pages ='**,len(b))**

b('Advanced Python', 595.98, 500)

b('django', 987.90, 100)

**print(b)**

**del b**