class Employee:

empCount=0

def \_\_init\_\_(self,name,salary):

self.name=name

self.salary=salary

Employee.empCount+=1

def displayCount(self):

print("total employee %d" % Employee.empCount)

def displayEmployee(self):

print("Name;",self.name,",salary:",self.salary)

emp1=Employee("nikhil",9999)

emp1.displayEmployee()

print("is salary an attribute:",hasattr(emp1,'salary'))

print(getattr(emp1,'salary'))

setattr(emp1,'salary',7000)

print("modified salary",getattr(emp1,'salary'))

print(hasattr(emp1,'desg'))

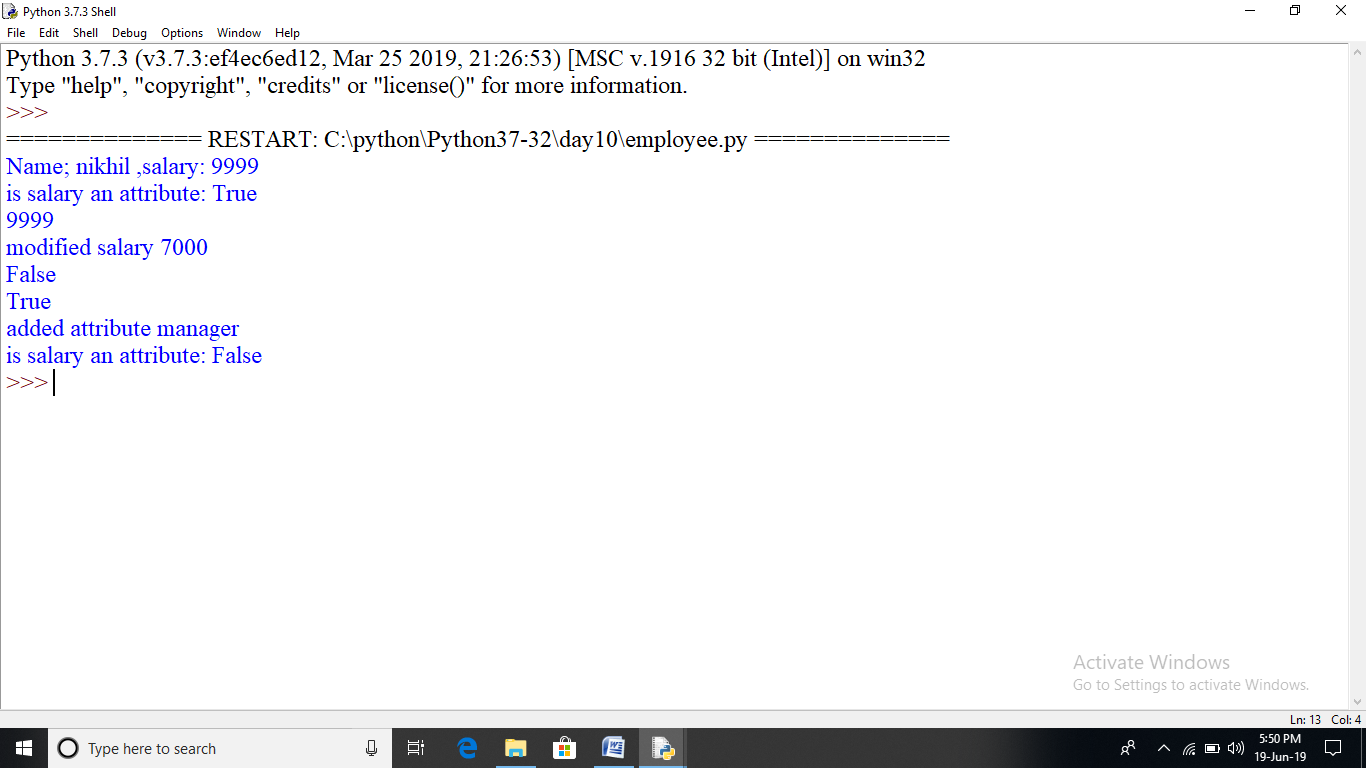
setattr(emp1,'desg','manager')

print(hasattr(emp1,'desg'))

print("added attribute",getattr(emp1,'desg'))

delattr(emp1,'salary')

print("is salary an attribute:",hasattr(emp1,'salary'))



class Employee:

empCount=0

def \_\_init\_\_(self,name,salary):

self.name=name

self.salary=salary

Employee.empCount+=1

def displayCount(self):

print("total employee %d" % Employee.empCount)

def displayEmployee(self):

print("Name;",self.name,",salary:",self.salary)

emp1=Employee("nikhil",9999)

emp1.displayEmployee()

print("is salary an attribute:",hasattr(emp1,'salary'))

print(getattr(emp1,'salary'))

setattr(emp1,'salary',7000)

print("modified salary",getattr(emp1,'salary'))

print(hasattr(emp1,'desg'))

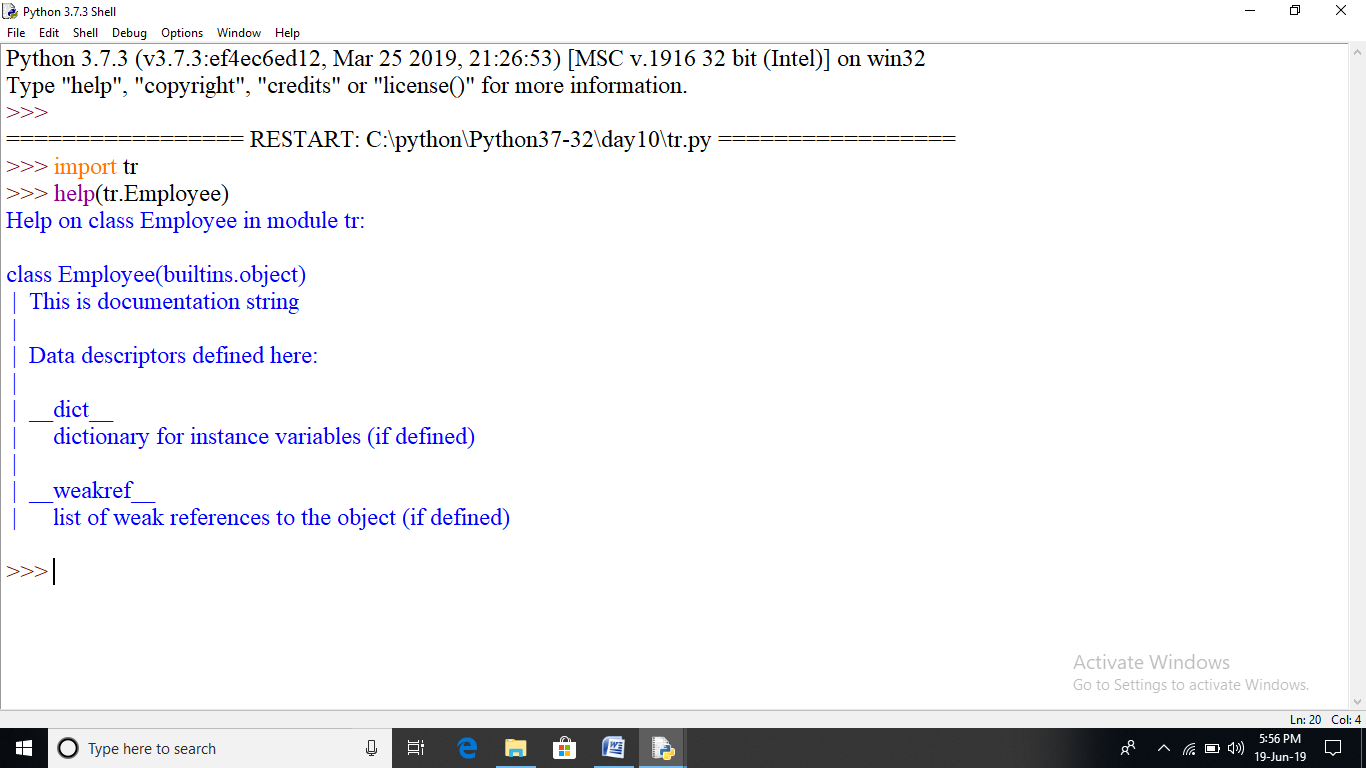
setattr(emp1,'desg','manager')

print(hasattr(emp1,'desg'))

print("added attribute",getattr(emp1,'desg'))

delattr(emp1,'salary')

print("is salary an attribute:",hasattr(emp1,'salary'))



##destructor

class Point:

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

self.x=x

self.y=y

def \_\_del\_\_(self):

class\_name=self.\_\_class\_\_.\_\_name\_\_

print(class\_name,"destroyed")

pt1=Point()

pt2=pt1

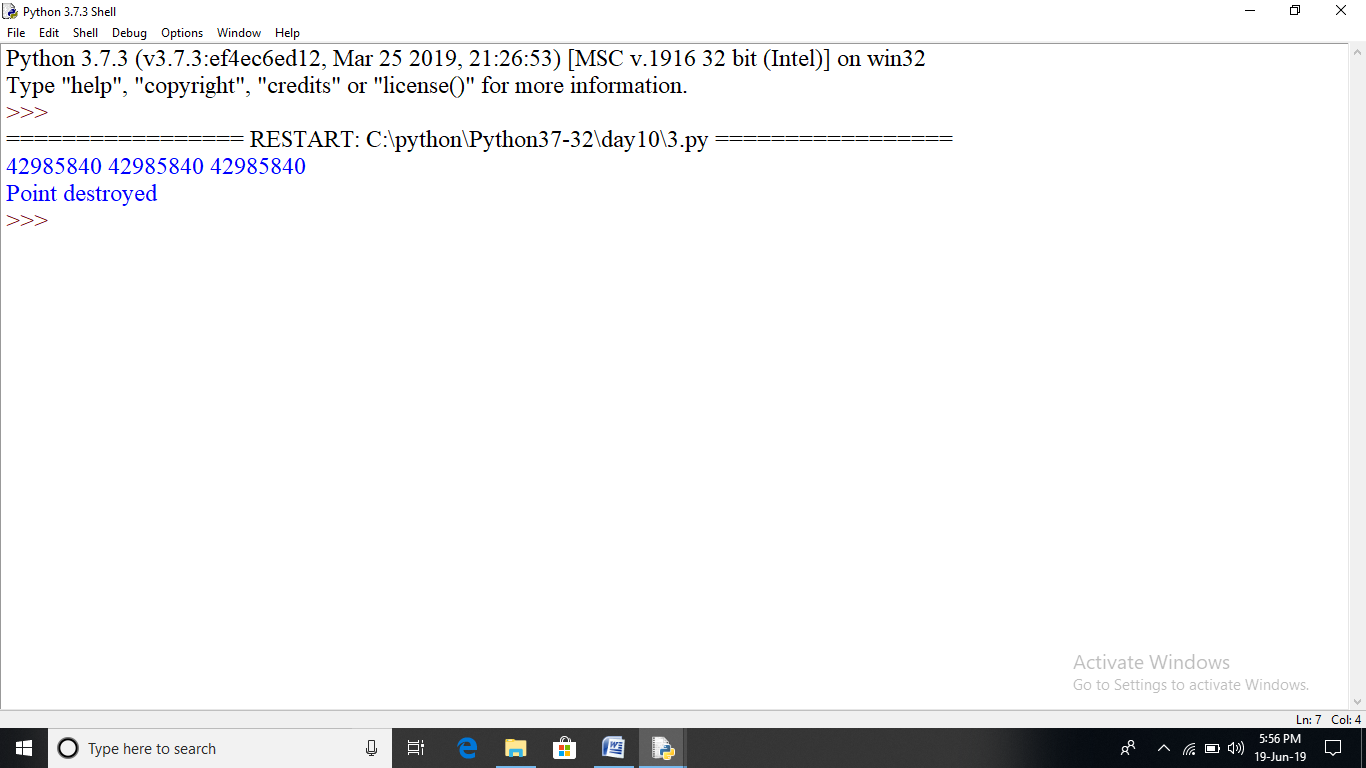
pt3=pt1

print(id(pt1),id(pt2),id(pt3))

del pt1

del pt2

del pt3



def printinfo(arg1,\*vartuple):

print("output is:",arg1)

print("content of variabe length tuple is:")

for var\_temp in vartuple:

print(var\_temp,end=' ')

print()

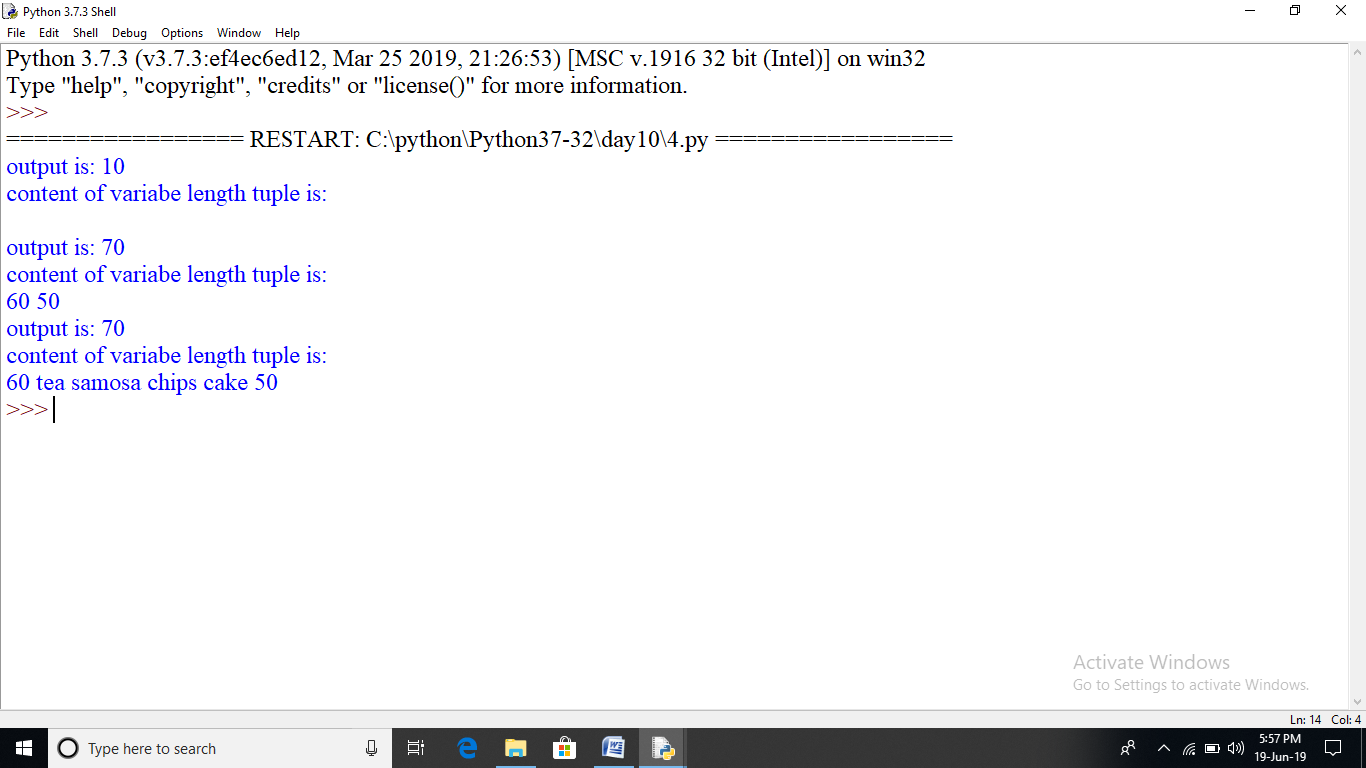
return

if \_\_name\_\_=='\_\_main\_\_':

printinfo(10)

printinfo(70,60,50)

printinfo(70,60,'tea','samosa','chips','cake',50)



def test\_fun(stream,course,fee):

print("args1",stream)

print("args2",course)

print("args3",fee)

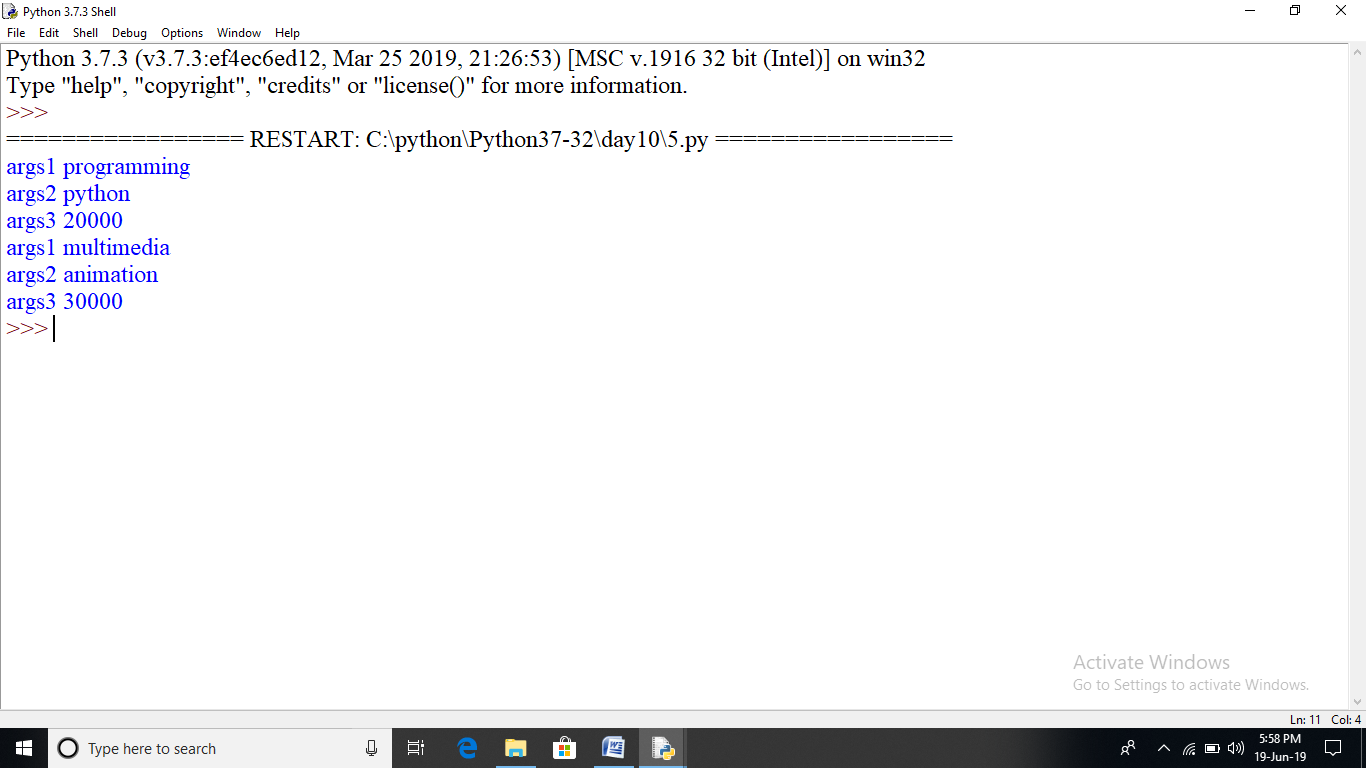
return

tup1=('programming','python',20000)

test\_fun(\*tup1)

dict1={"fee":30000,"course":"animation","stream":"multimedia"}

test\_fun(\*\*dict1)



def printEmpSkill(name,\*skillset):

print('{0} is skilled in {1} language '.format(name,skillset))

print('{0} is skilled in {1} language '.format(name,len(skillset)))

return None

printEmpSkill('alex','python','html')

printEmpSkill('parvesh','java','c','c++')

printEmpSkill('ajay')



total=100

def add(arg1,arg2):

global total

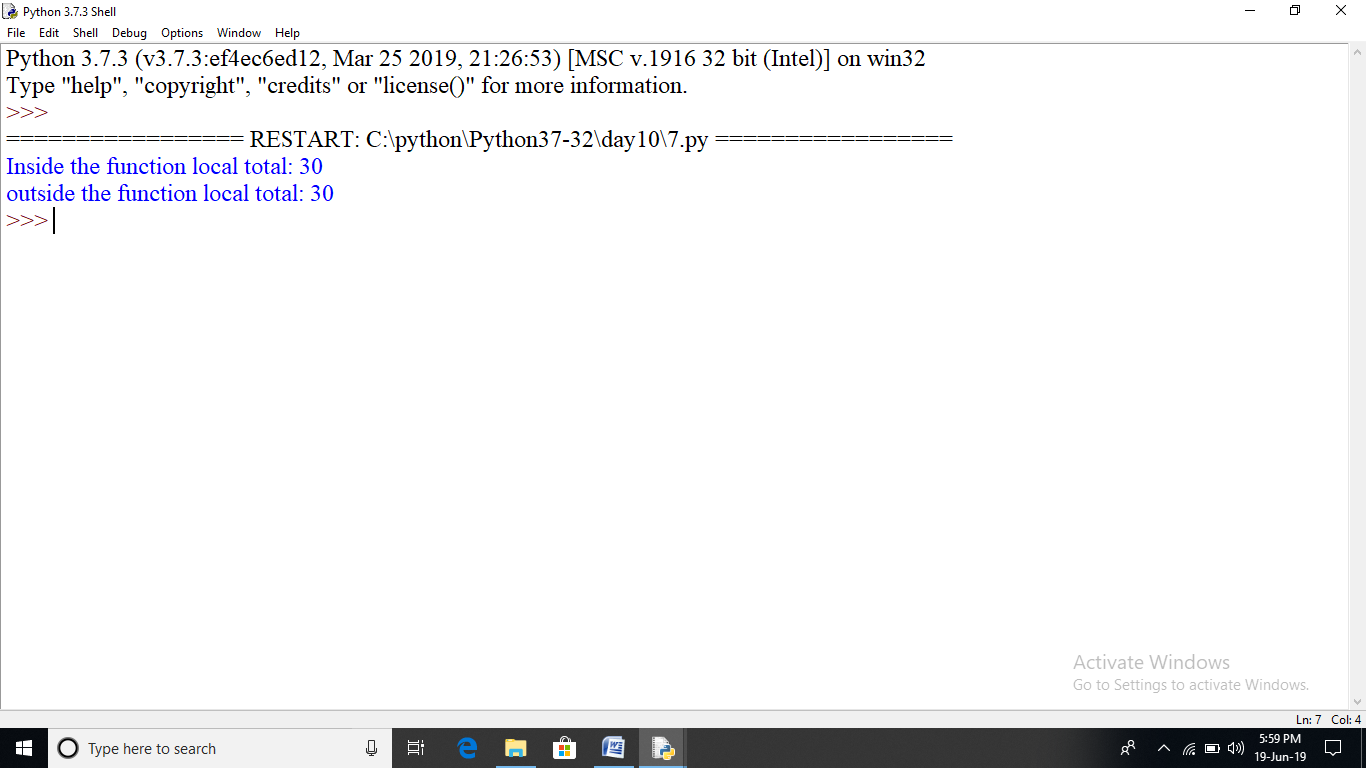
total=arg1+arg2

print("Inside the function local total:",total)

return total

a=add(10,20)

print("outside the function local total:",total)



#order of using \*args \*\*kwargs and formal args

def test\_var\_args(f\_arg,\*argv,\*\*kwargs):

print("first normal arg:",f\_arg)

if argv is not None:

for arg in argv:

print("another arg through \*argv:",arg)

if kwargs is not None:

for (key,value) in kwargs.items():

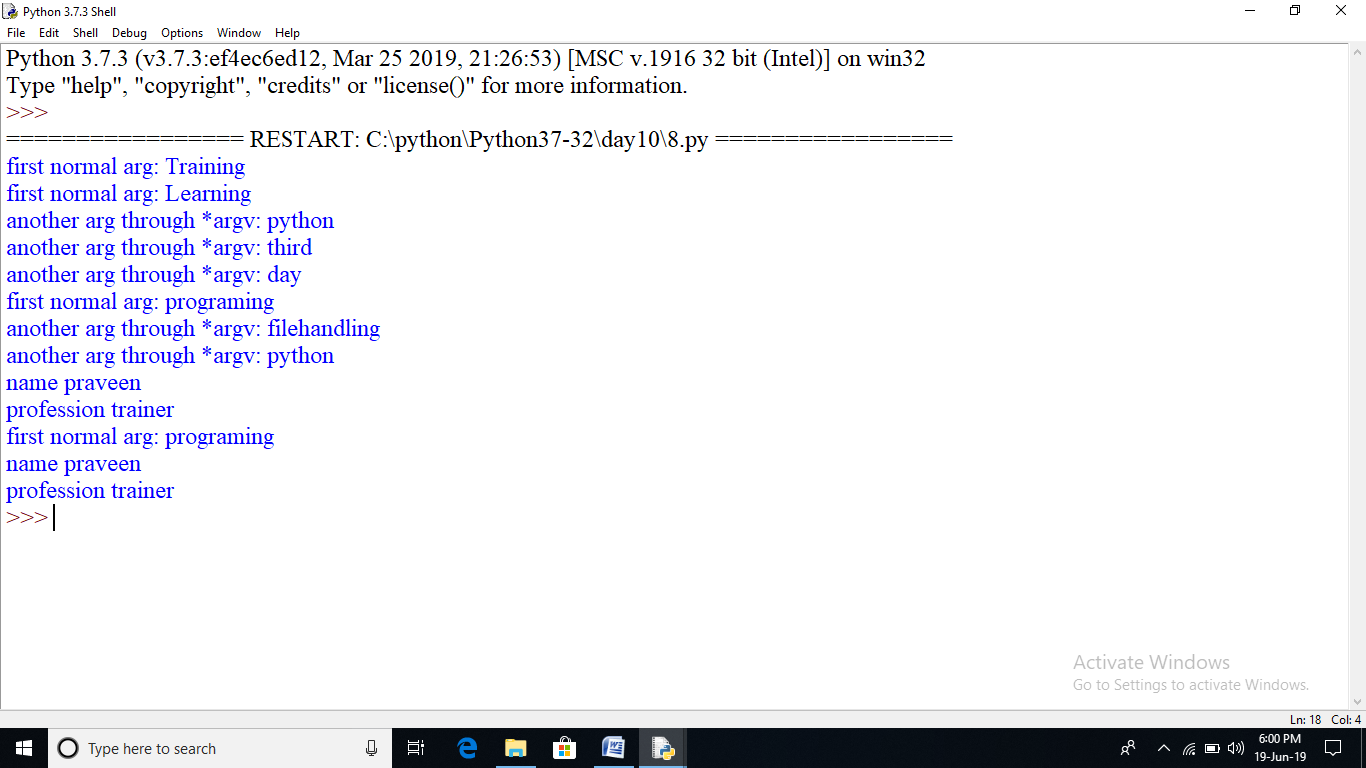
print(key,value)

test\_var\_args('Training')

test\_var\_args('Learning','python','third','day')

test\_var\_args('programing','filehandling','python',name="praveen",profession="trainer")

test\_var\_args('programing',name="praveen",profession="trainer")



x=5;y=7

def changeme(mylist):

"This function demonstrates local and global variables"

p=89

global x,y

x=y+2

mylist=[1,2,3,4]

print("values inside the function:",mylist)

print("local variables are:",locals())

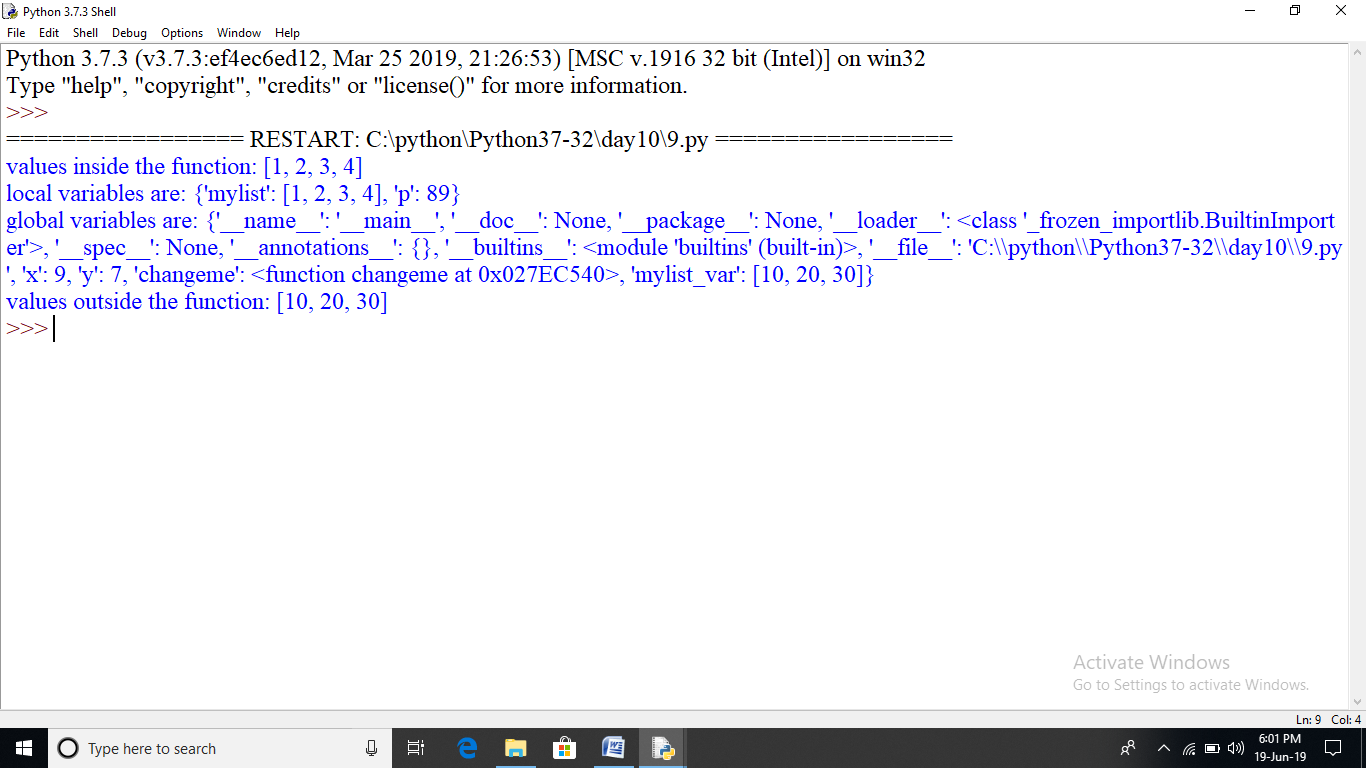
print("global variables are:",globals())

return

mylist\_var=[10,20,30]

changeme(mylist\_var)

print("values outside the function:",mylist\_var)



#factorial with recursion

#the mathematical definition of factorial is:

#n!=n\*(n-1)! if n>0 and f(0)=1

def factorial(n):

if n==0:

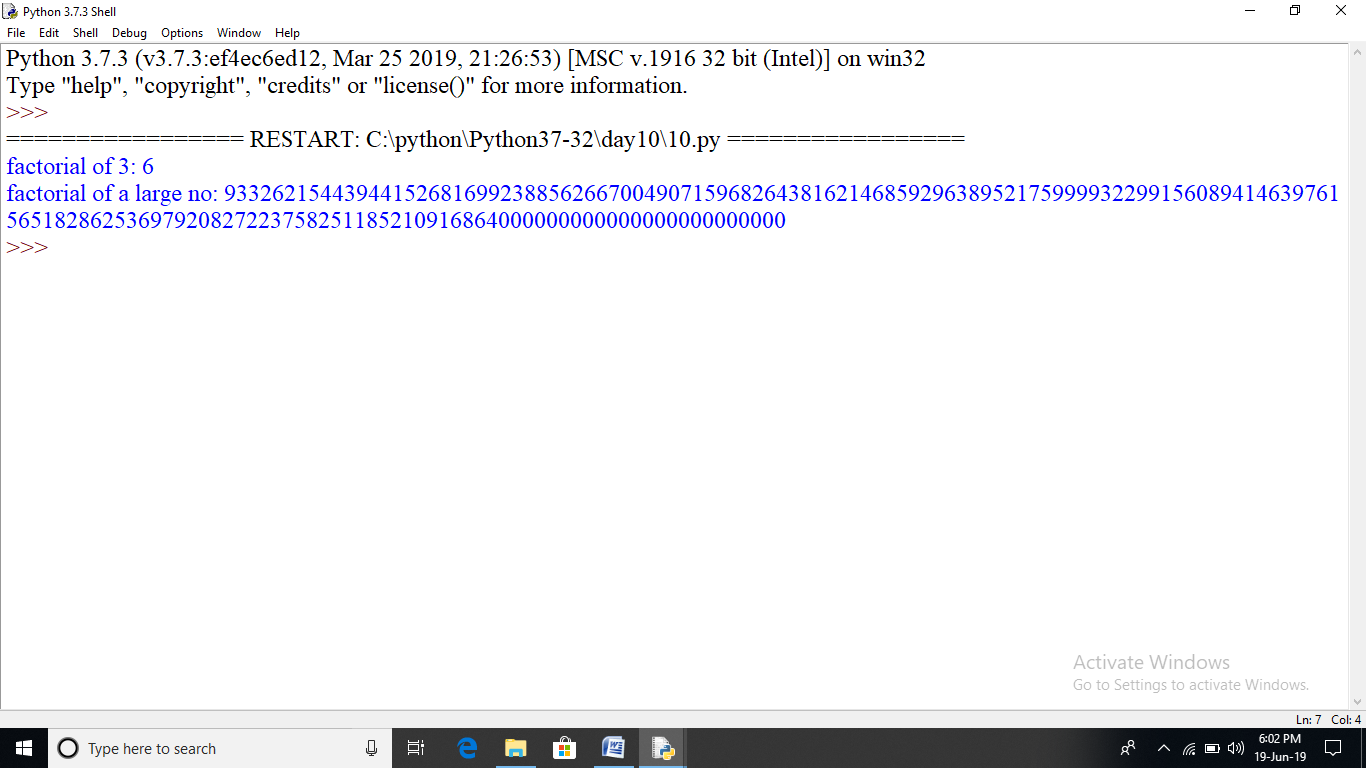
return 1

else:

return n\*factorial(n-1)

print("factorial of 3:",factorial(3))

print('factorial of a large no:',factorial(100))



#you can resolve recursion depth by modifing the number of recursion calls

import sys

sys.setrecursionlimit(5000)

def factorial(n):

if n==0:

return 1

else:

return n\*factorial(n-1)

print(factorial(3000))

