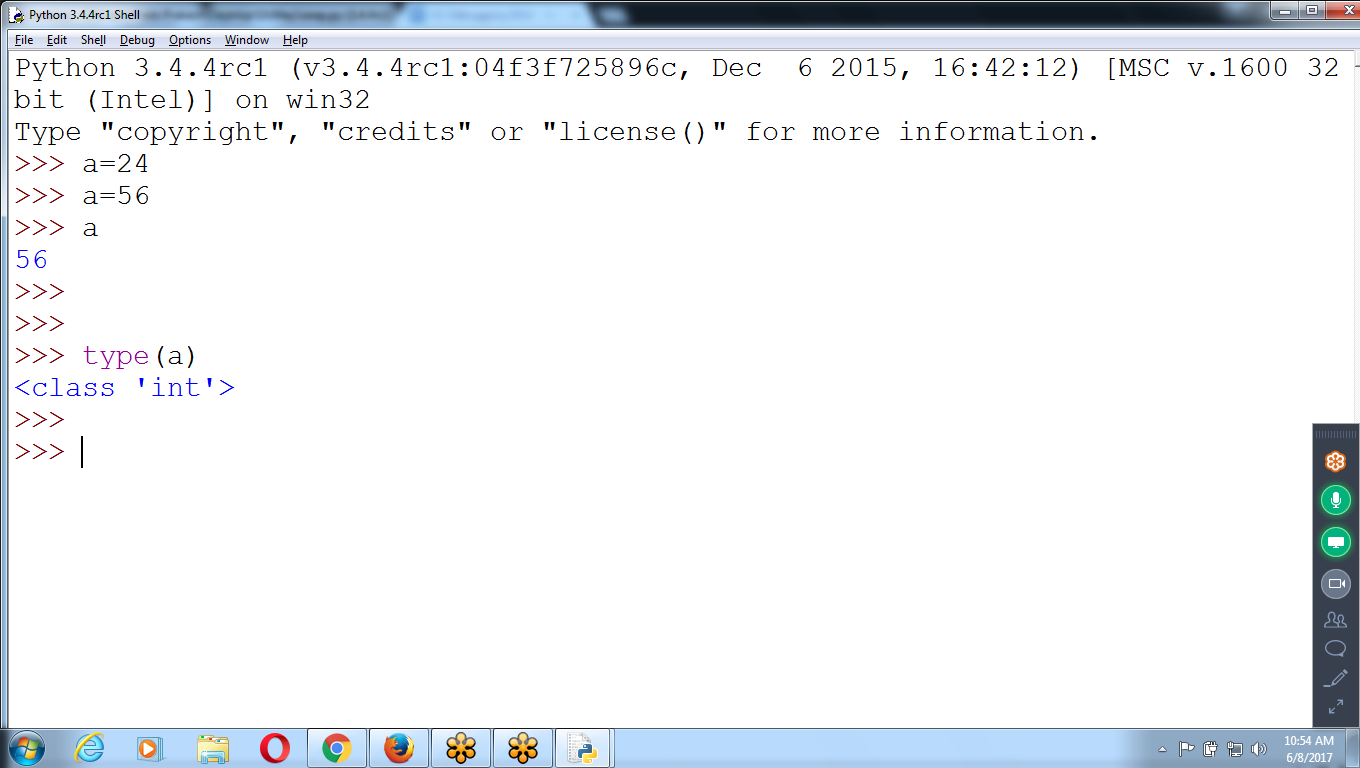
MEMORY Management of Variables

A value of type integer



>>> a = 24

>>> type(a)

<class 'int'>

>>>

A is an variable which hold address of the value : 24

>>> a = 'python progam'

>>> type(a)

<class 'str'>

>>>

A contains reference of type string

>>> a= 'raju'

>>> a = 'siva'

>>> a

'siva'

>>>

A contains starting address of the value

A 6001 to 6004

**6001**  raju

7001

7001 to 7004

siva

A is an integer

>>> a = 24

>>> a = 45

>>> a

45

>>>

>>>

St is string

>>> st = 'raju'

>>> st = 'python'

>>> st

'python'

>>>

St 8001...8004

8001 raju

**9001**

9001

python

>>> st[0]

'p'

>>> st[1]

'y'

**>>> st[2] = 'y'**

Traceback (most recent call last):

File "<pyshell#82>", line 1, in <module>

st[2] = 'y'

TypeError: 'str' object does not support item assignment

>>>

Why can’t assign new value string subscript

**STRINGS are Immutable Objects, can’t Modify Structure**

FOR List Types

>>> ls = ['raju', 'rani', **'siva']**

>>> ls

['raju', 'rani', 'siva']

>>> ls[2] = 'ravi'

>>> ls

['raju', 'rani', **'ravi']**

>>>

**List can be Modify the Structure**

**LIST are Mutable Objects**

**Tuple Structure**

>>> tp = ('raju', 'rani', 'siva')

>>> tp[2] = 'ravi'

Traceback (most recent call last):

File "<pyshell#105>", line 1, in <module>

tp[2] = 'ravi'

TypeError: 'tuple' object does not support item assignment

>>>

>>>

**Tuples are Immutable Objects**

**STrings:**

st = 'python program'

>>> st2 = 'language'

>>> st3 = st + st2

>>>

>>>

>>>

>>> st4 = **'python'** + st3

>>> st4

'pythonpython programlanguage'

>>>

**St is object which holds address of the value ‘Python program’**

st4 = 'python' + st3

>> a = 9 \*\* 999

>>> a

194207916858072401073330513240517841169895831937243168645765334645631807358586165476831829984964567897289883410682808509863485381763945405279379355788182053541434708898886353264614403164257835946591015853500491562156765579388944516423770646547300211711400609344237550775485394558425026601257627110879613741893863295847627378504481736441703291029360564416718984718052676789493826372811349572386149787861703350363229770343522164432121091627871310618608734044108407173015970850780786711471108639762810760748899301375323974504010469298672123113693793242558662498267897607159946316136440215024585534972601864730717278590674861331708227340510282977338127859756479389076075528672989549862138485404935127984793120586289288424045660573066638008624179879066798350622453419082976217706653276687992598885030141711458658381360884807741768071789239593772708382532520992894115725948613681993478965648216640862698897925988931145600683858128653568049999074868783790048889

>>> sa = str(a)

>>> type(sa)

<class 'str'>

>>> len(sa)

954

>>>

HOw many Digits exist for 9 \*\* 9999

>>> a = 9 \*\* 9999

**>>> sa = str(a)**

>>> type(a)

<class 'int'>

>>> type(sa)

<class 'str'>

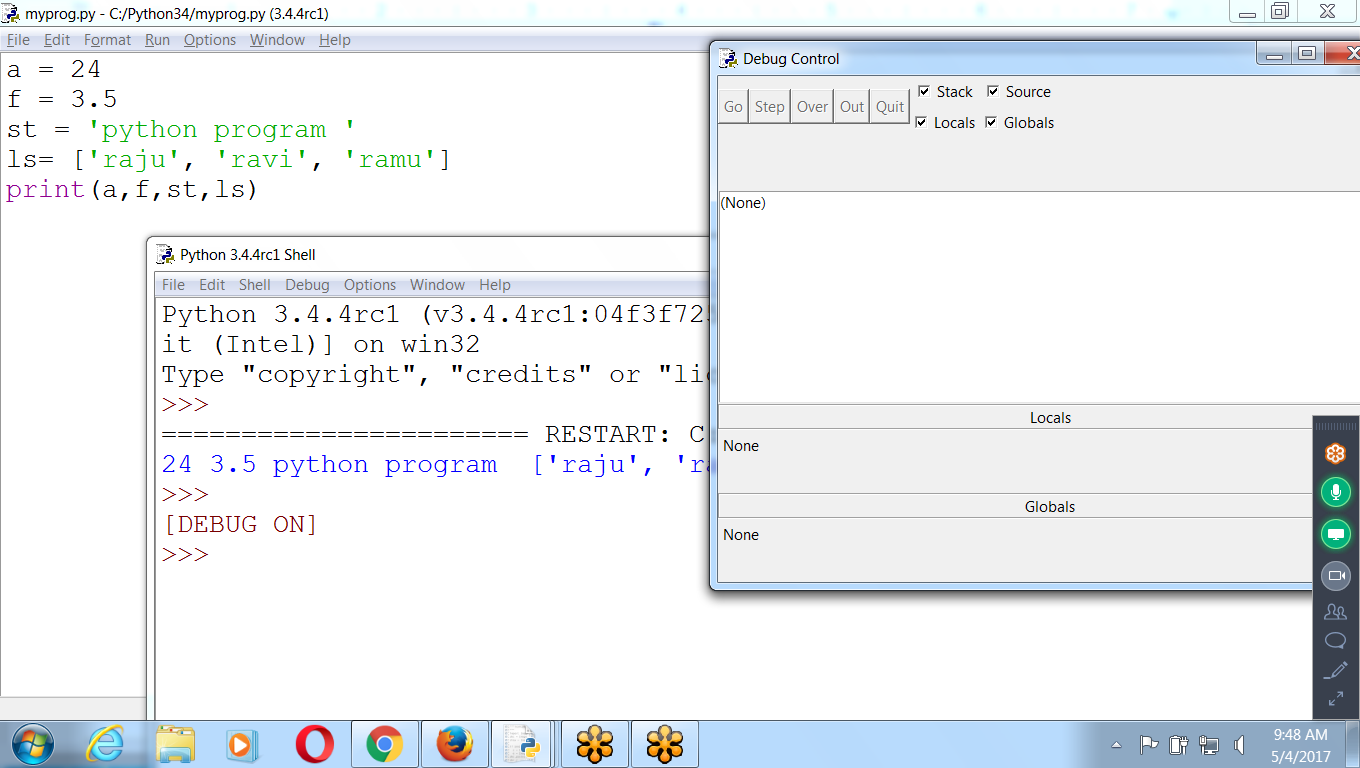
>>>

**>>> len(sa)**

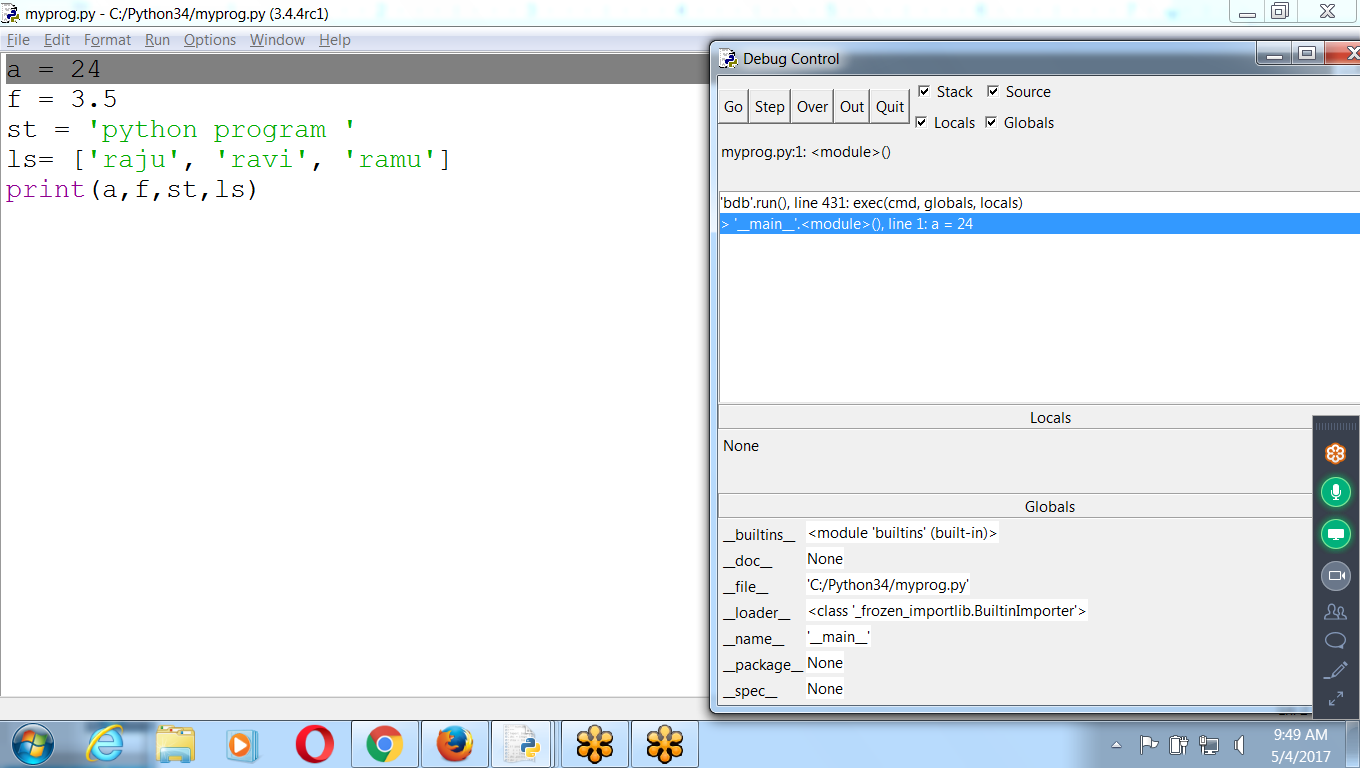
**9542**

**>>>**

3 Windows opened : Program, Shell, Debug Opened : Click Execute



Run (F5)



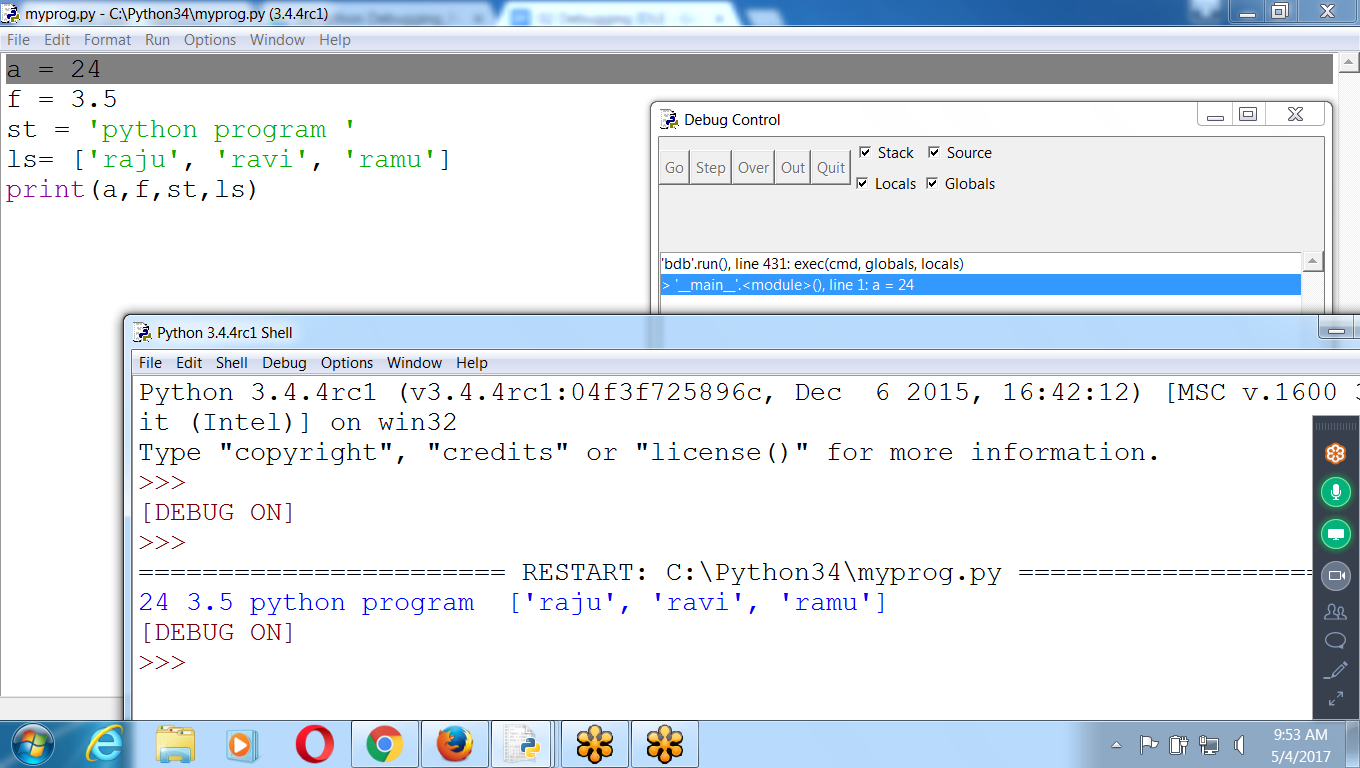
The program will stay paused until press one of the five buttons in the Debug Control window: Go, Step, Over, Out, or Quit.

## **Go**

* Clicking the Go button will cause the program to execute normally until it terminates or reaches a *breakpoint*.
* If debugging is done and want the program to continue normally, click the **Go** button.

Run - execute - CLICK “GO” in debugger

Result : Normal Execution Program

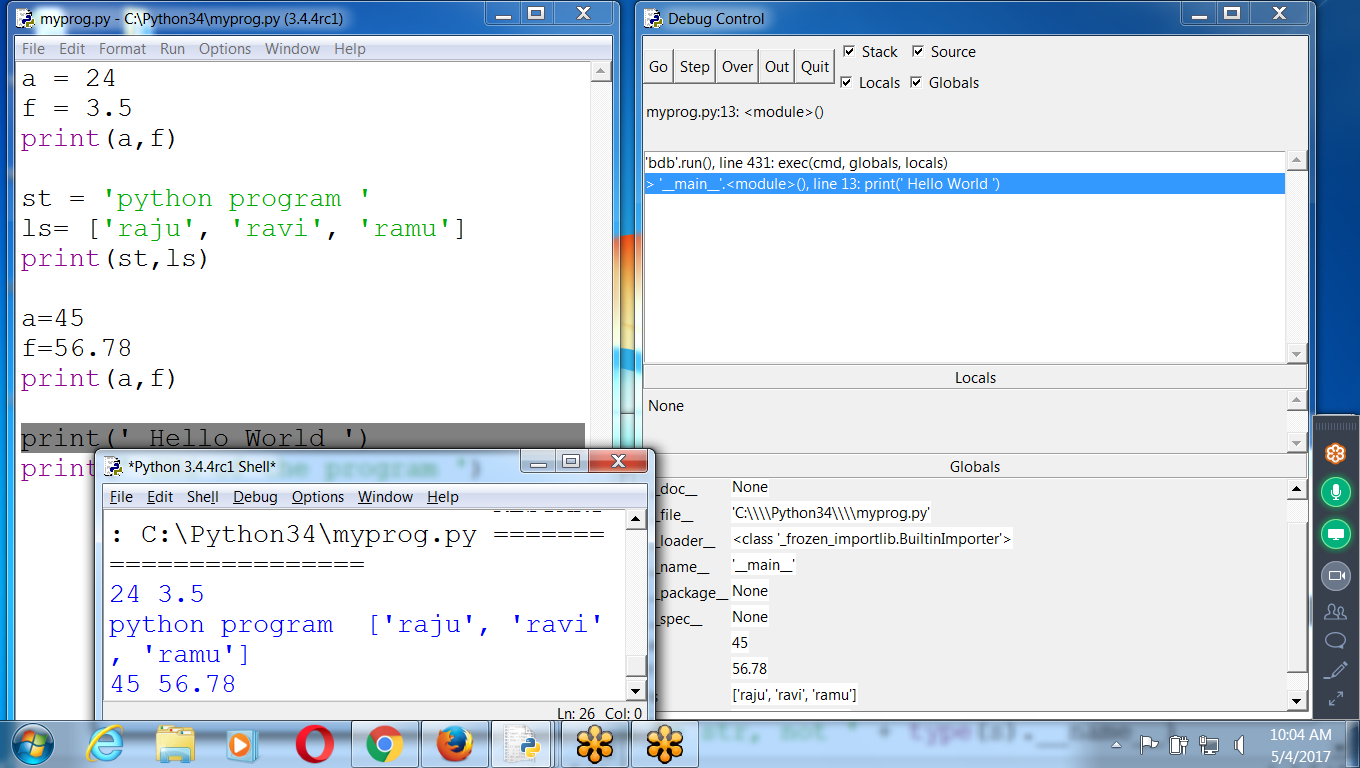


## 

## 

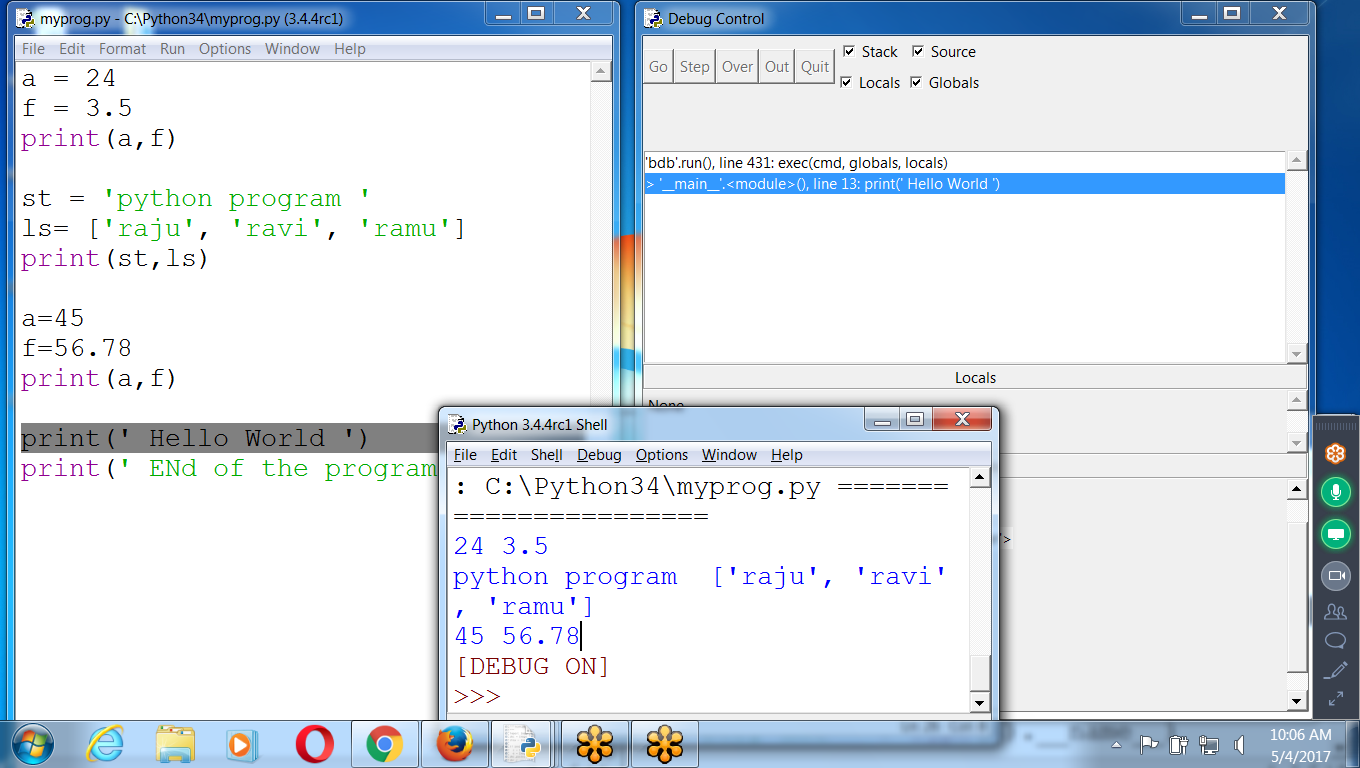
## **2) Step (Jumps into the function)**

* Clicking the Step button will cause the debugger to execute the next line of code and then pause again.
* The Debug Control window’s list of global and local variables will be updated if their values change.
* If the next line of code **is a function call,** the debugger will “step into” that function and **jump to the first line of code** of that function.



OUt: Comes out of the function

Quit: It stops current execution



## **3)Over (Don’t enter into function)**

* Clicking the Over button will execute the next line of code, similar to the Step button. However, if the next line of code is a function call, the Over button will “step over” the code in the function.
* The **function’s code will be executed** and the debugger will pause as soon as the function call returns.
* For example, if the next line of code is a **print()** call, don’t really care about code inside the built-in print() function; just want the string you pass it printed to the screen. For this reason, using the **Over button** is more common than the Step button.

## **Out ( Out of the function)**

* Clicking the Out button will cause the debugger to execute lines of code at full speed until **it returns from the current function**.
* If you have stepped into a function call with the Step button and now simply want to keep executing instructions until you get back out, click the **Out** button to “step out” of the current function call.

## **Quit ( Quit from the program)**

* To stop debugging entirely and **not to continue executing** the rest of the program, click the **Quit** button.
* The Quit button will immediately terminate the program. If you want to run your program normally again, select **Debug**▸**Debugger** again to disable the debugger.

**Testing 2 : Using Functions with Return values**

def f1(a,st):

print(' I am in f1 ')

a = 48

st = 'raju'

b = 45.4

print(' End of the f1 ')

r = a + 12

return r

a = 24

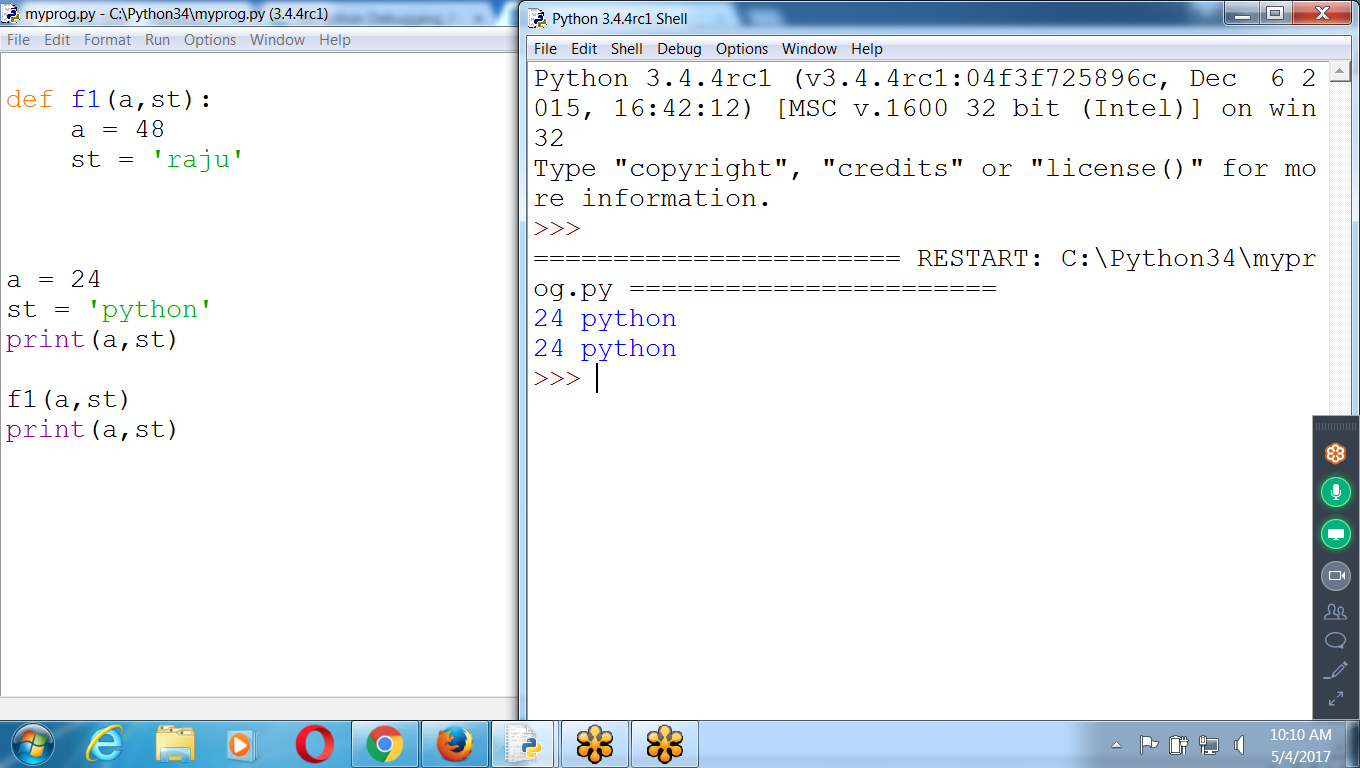
st = 'python'

print(a,st)

rt = f1(a,st)

**print(' Return value = ', rt)**

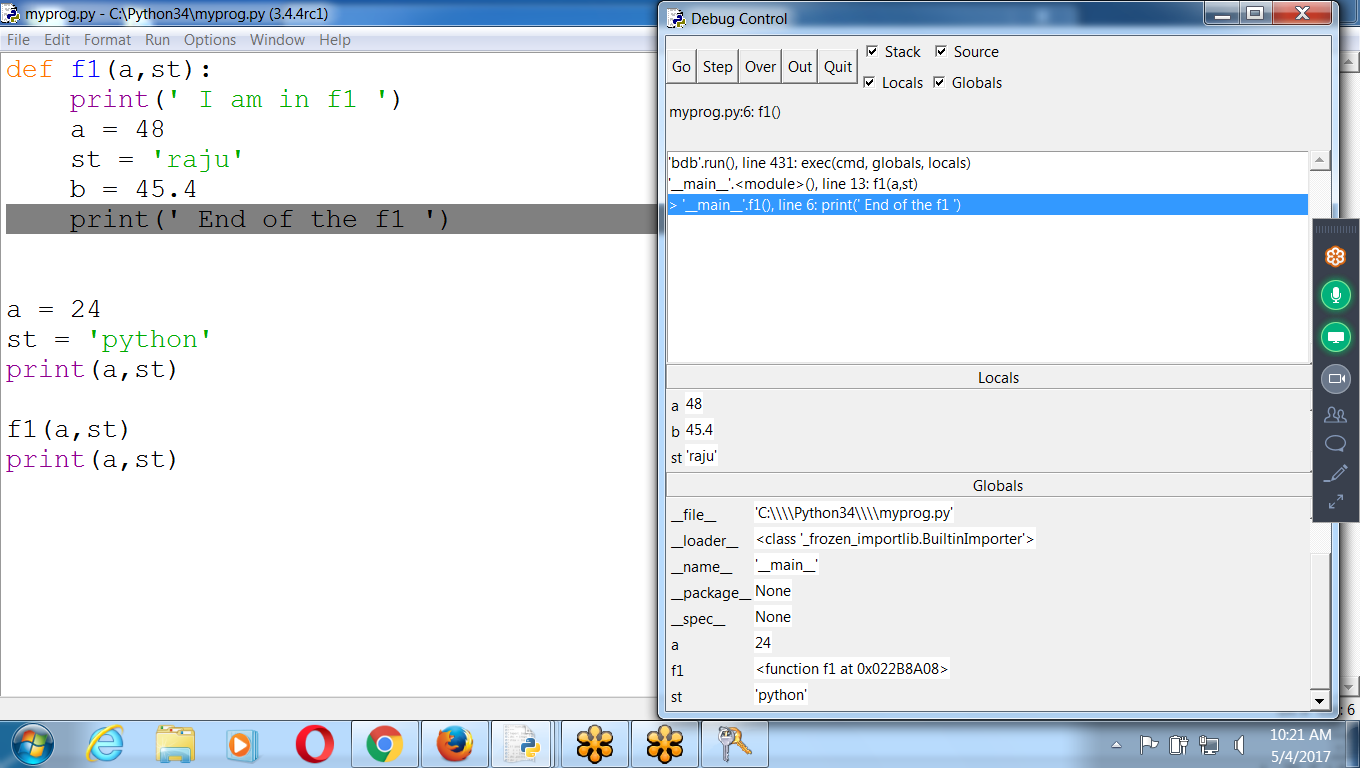
print(a,st)



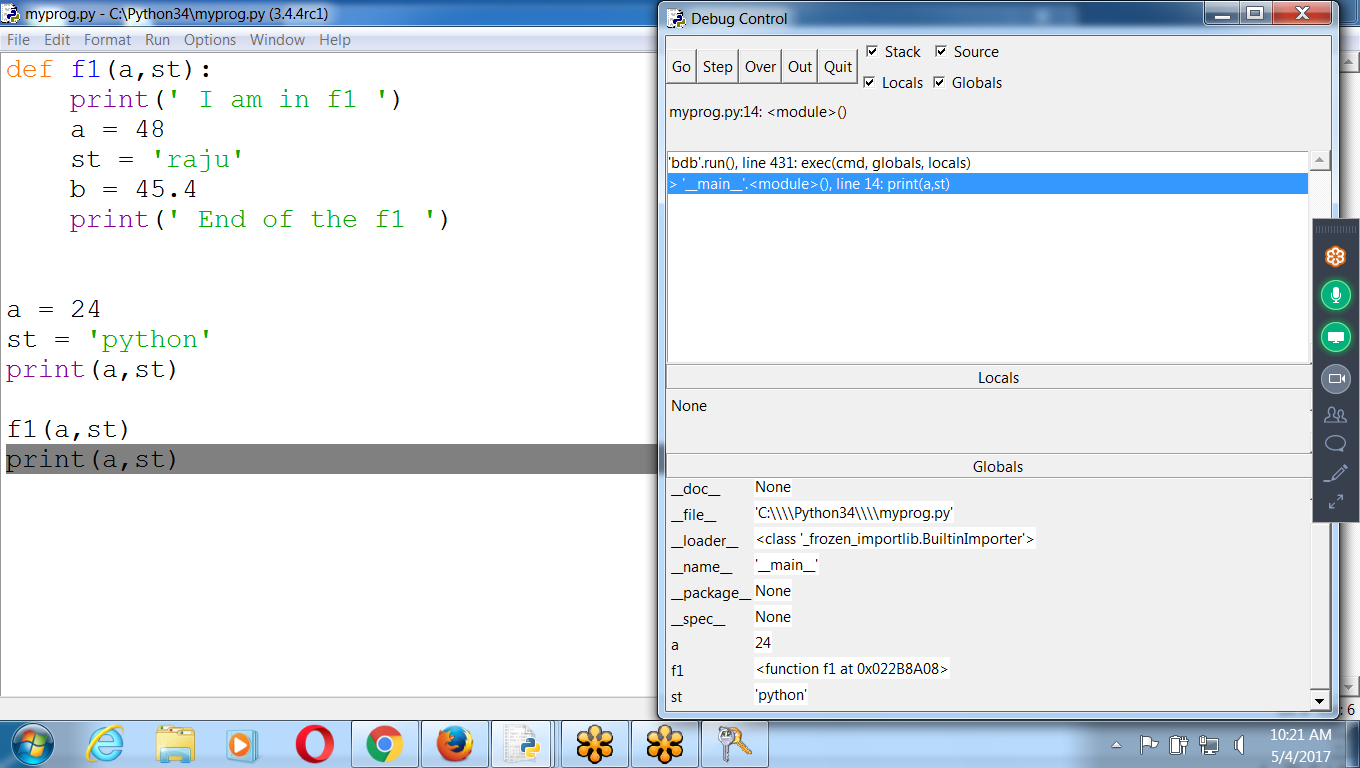
**Testing for Local and Global Variables**

Control is in F1() local variables :: a: 48, st :raju

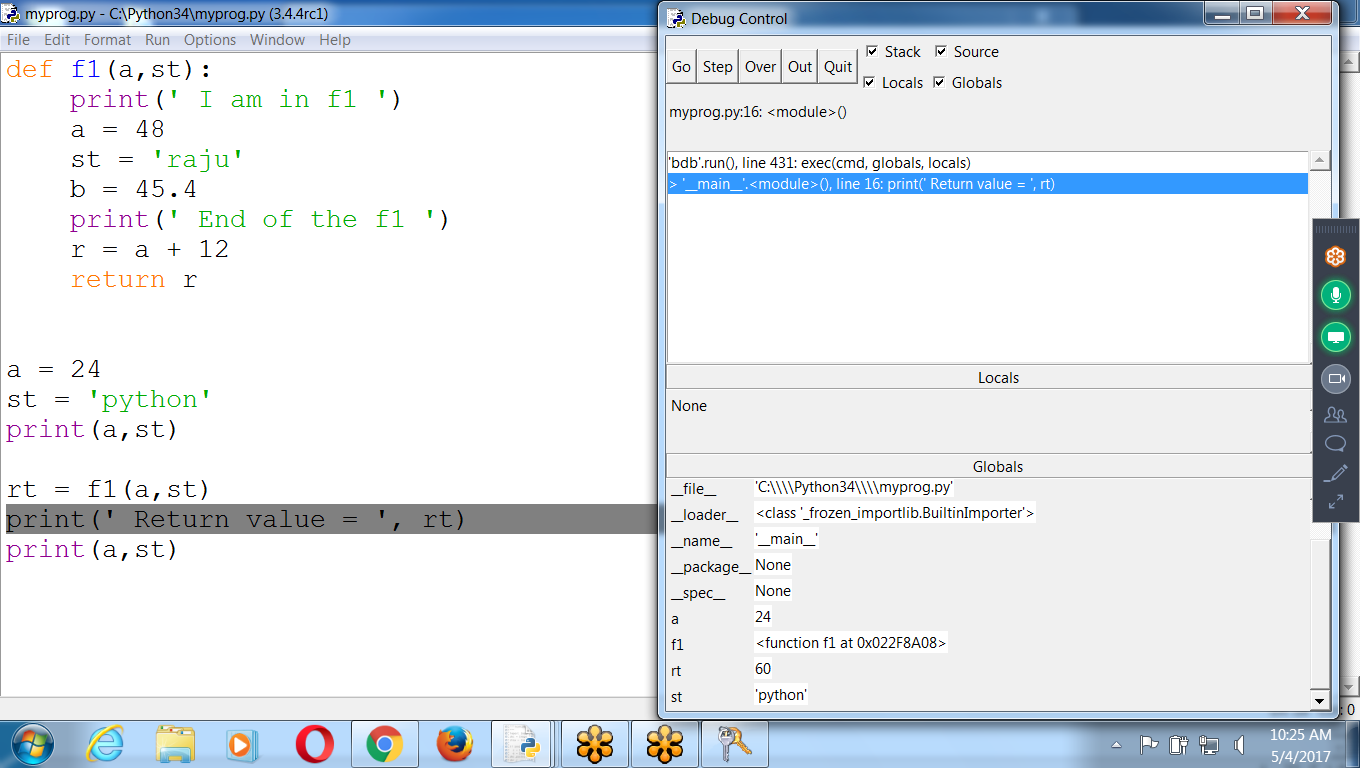
Global: a =24, st :python



Once control moves out of the function : local values are erased



3) On Returning value from a function



def f1(a,st):

print(' I am in f1 ')

a = 48

st = 'raju'

b = 45.4

print(' End of the f1 ')

r = a + 12

return r

a = 24

st = 'python'

print(a,st)

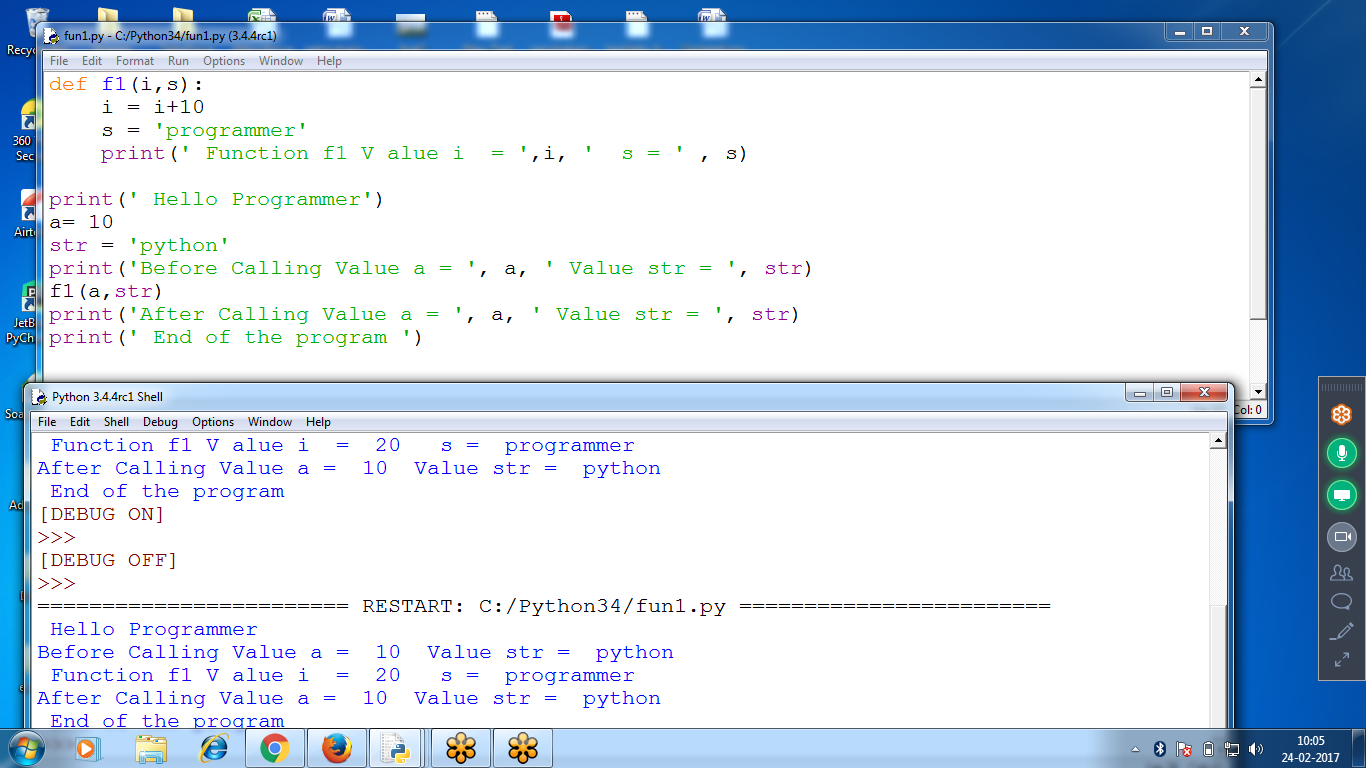
rt = f1(a,st)

print(' Return value = ', rt)

print(a,st)

DEBUGGER is OFf , Normal Execution like “RUN”

Executes All statements Directly



def f1(a,st):

print(' I am in f1 ')

a = 48

st = 'raju'

b = 45.4

print(' End of the f1 ')

a = 24

st = 'python'

print(a,st)

f1(a,st)

print(a,st)

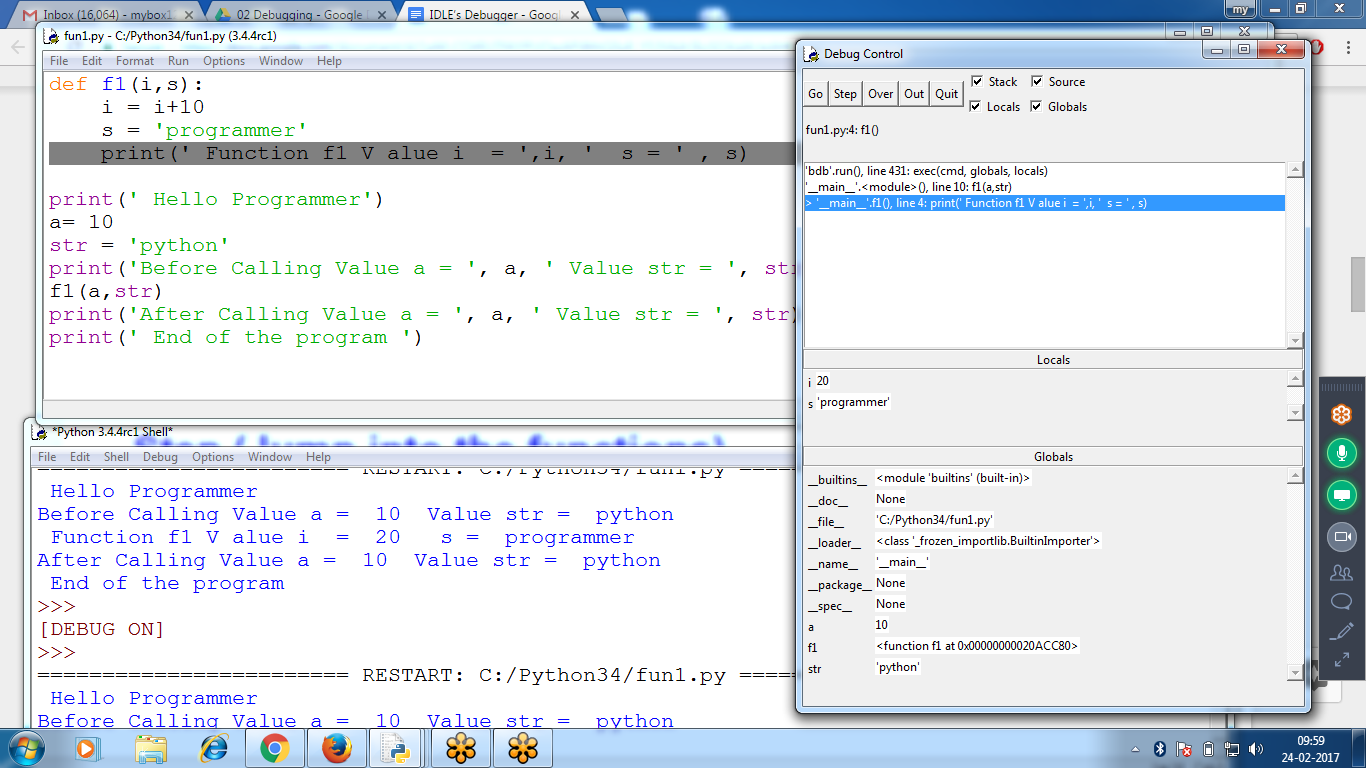
Click “GO” :: Normal Execution

Local and Global variables inside FUNCTION F1

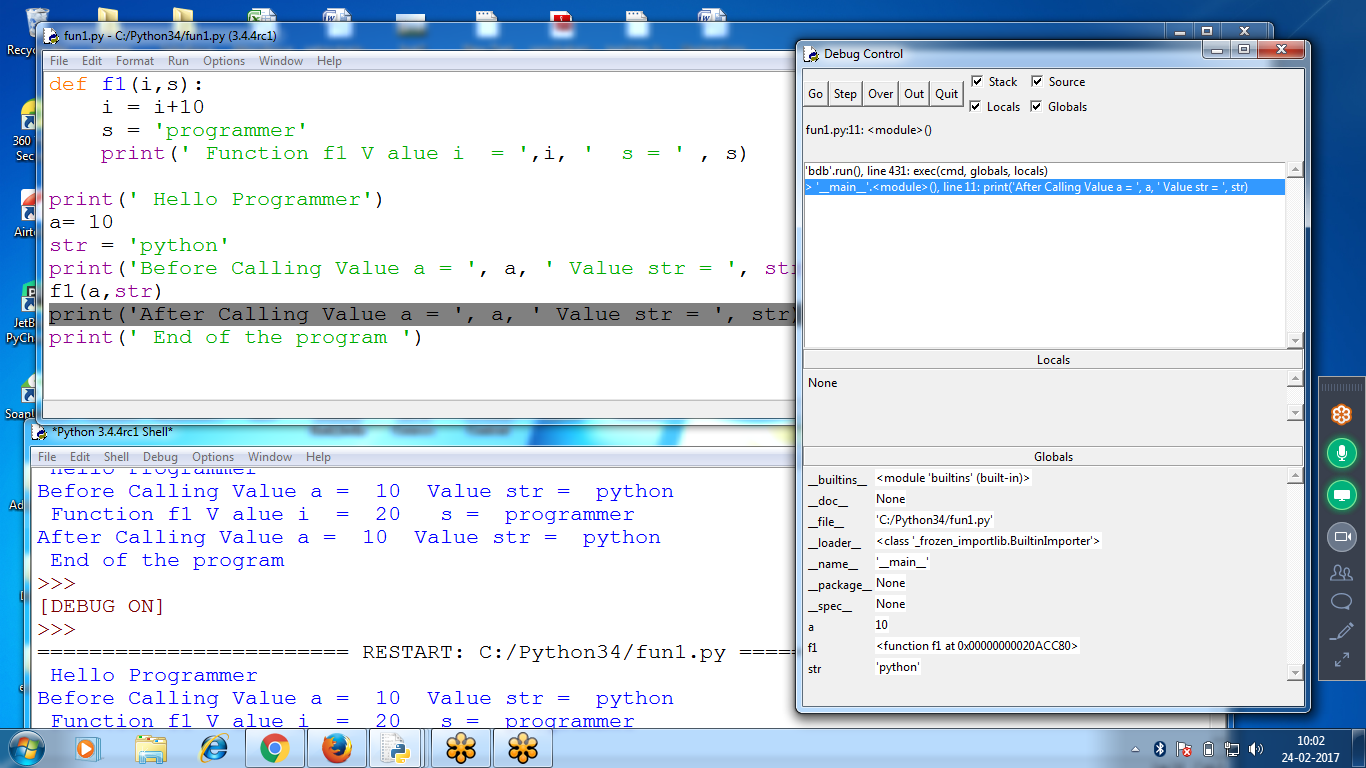
F1 Local variables :: i and s

Global Variables : a and str

Displaying Local variables i and s

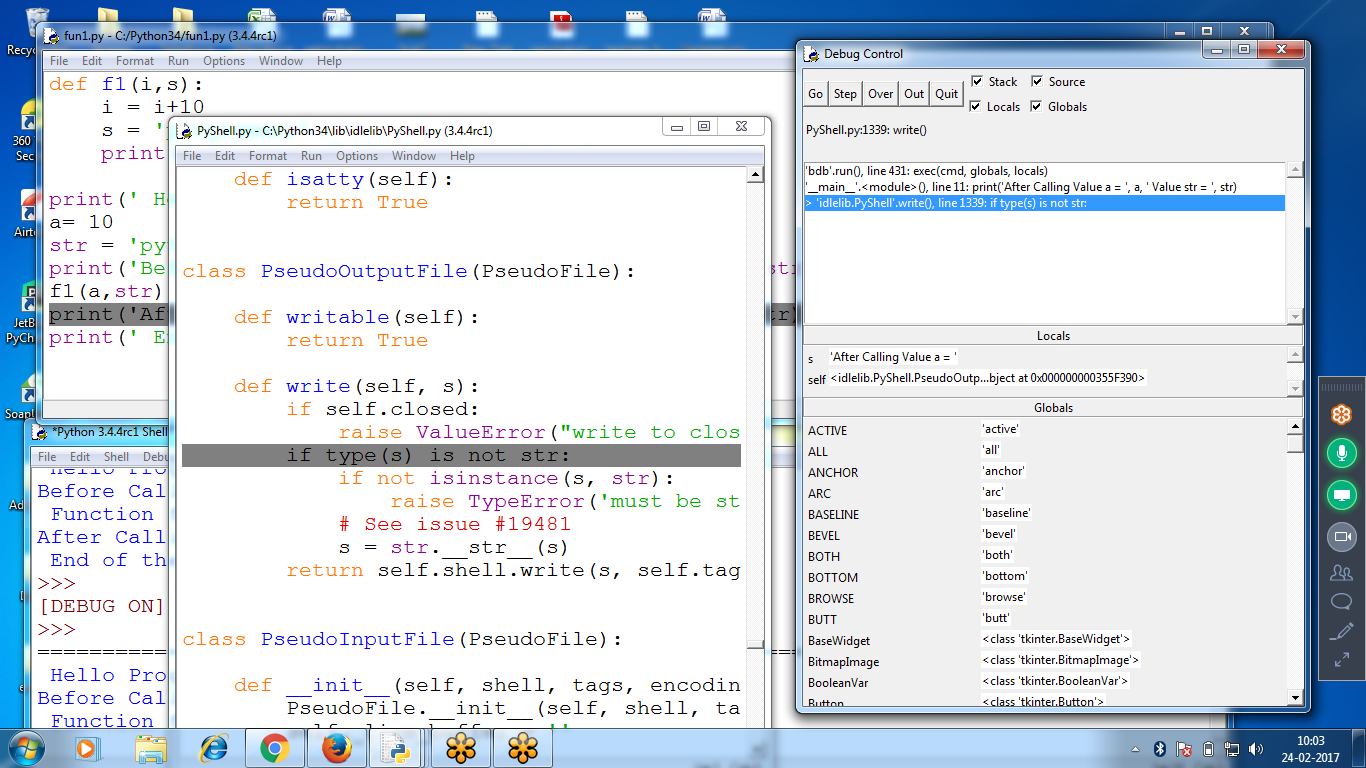


Outside Function f1: don’t find i ans s variables

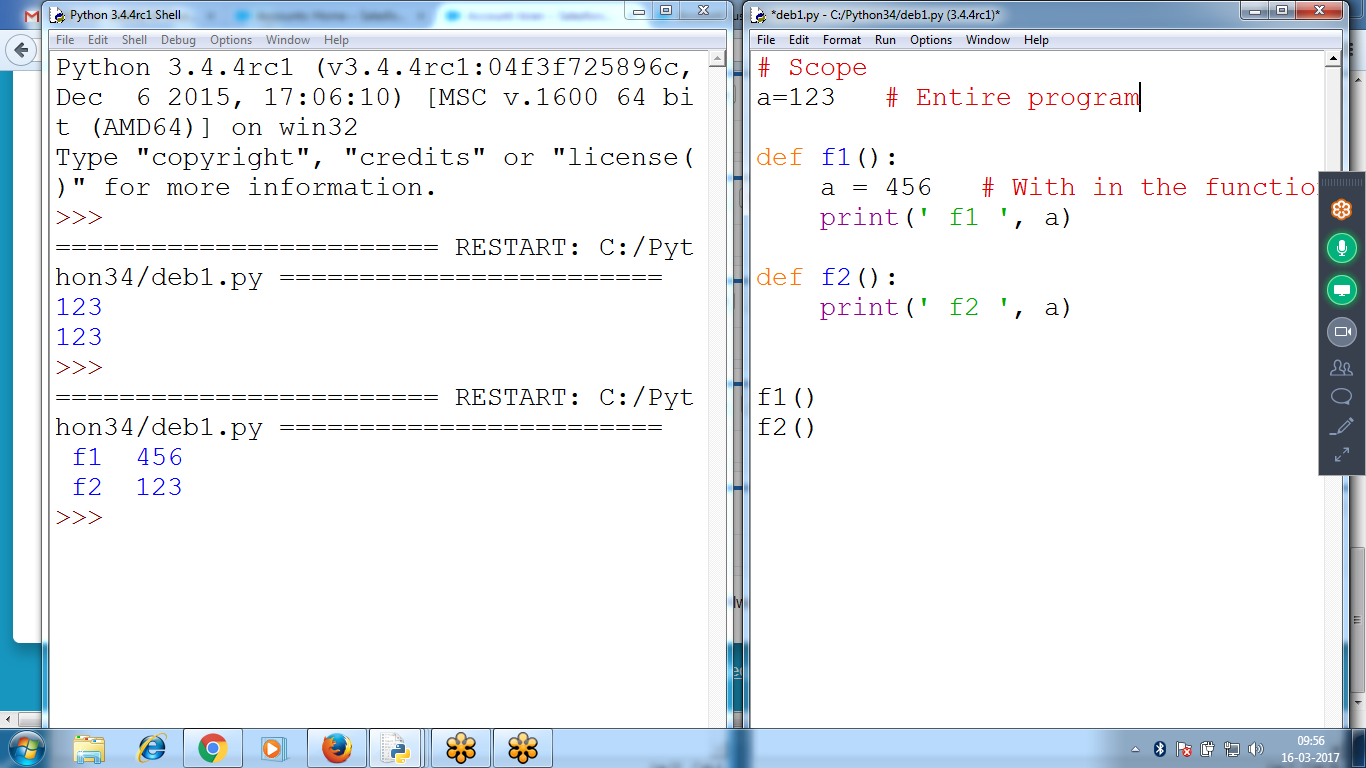


Inside Standard Function print():::

click “OUT” (to come out from the function)

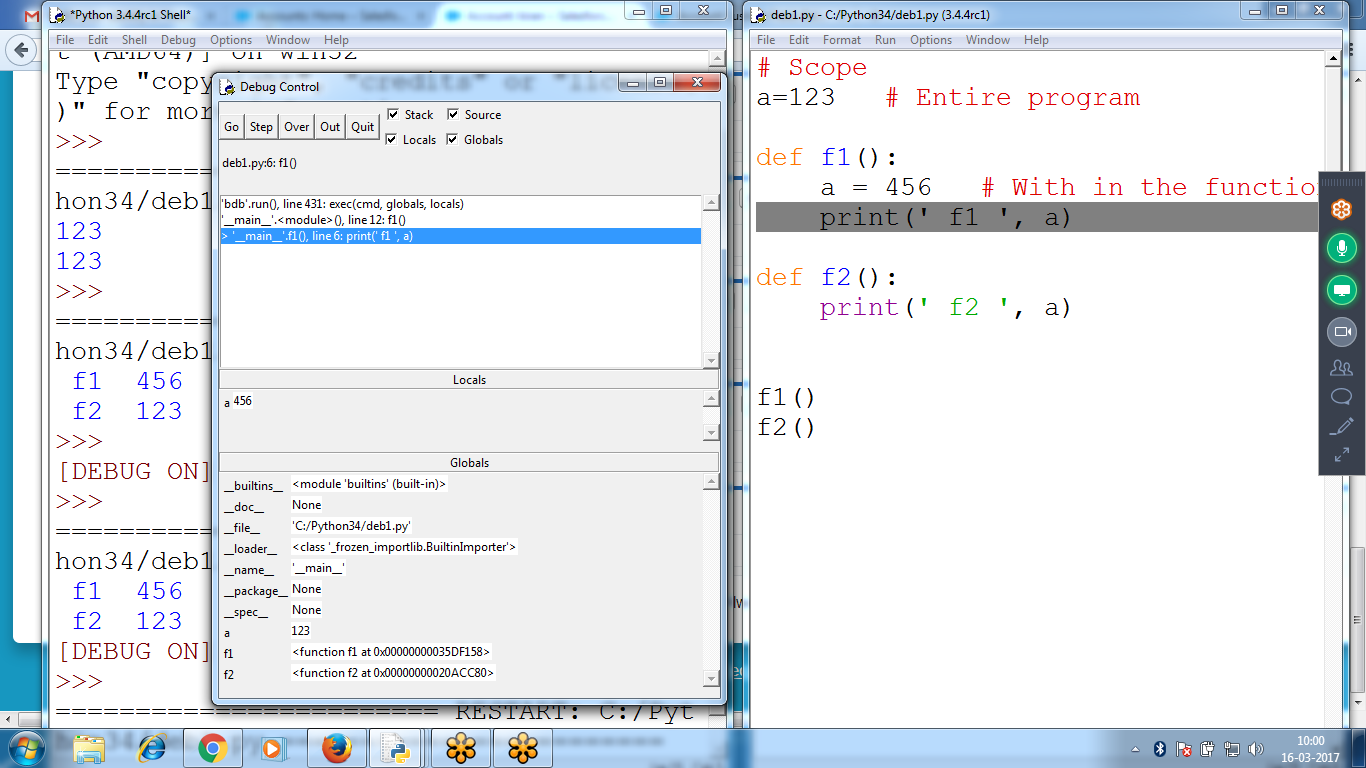


Checking For SCope ::: Global and Local

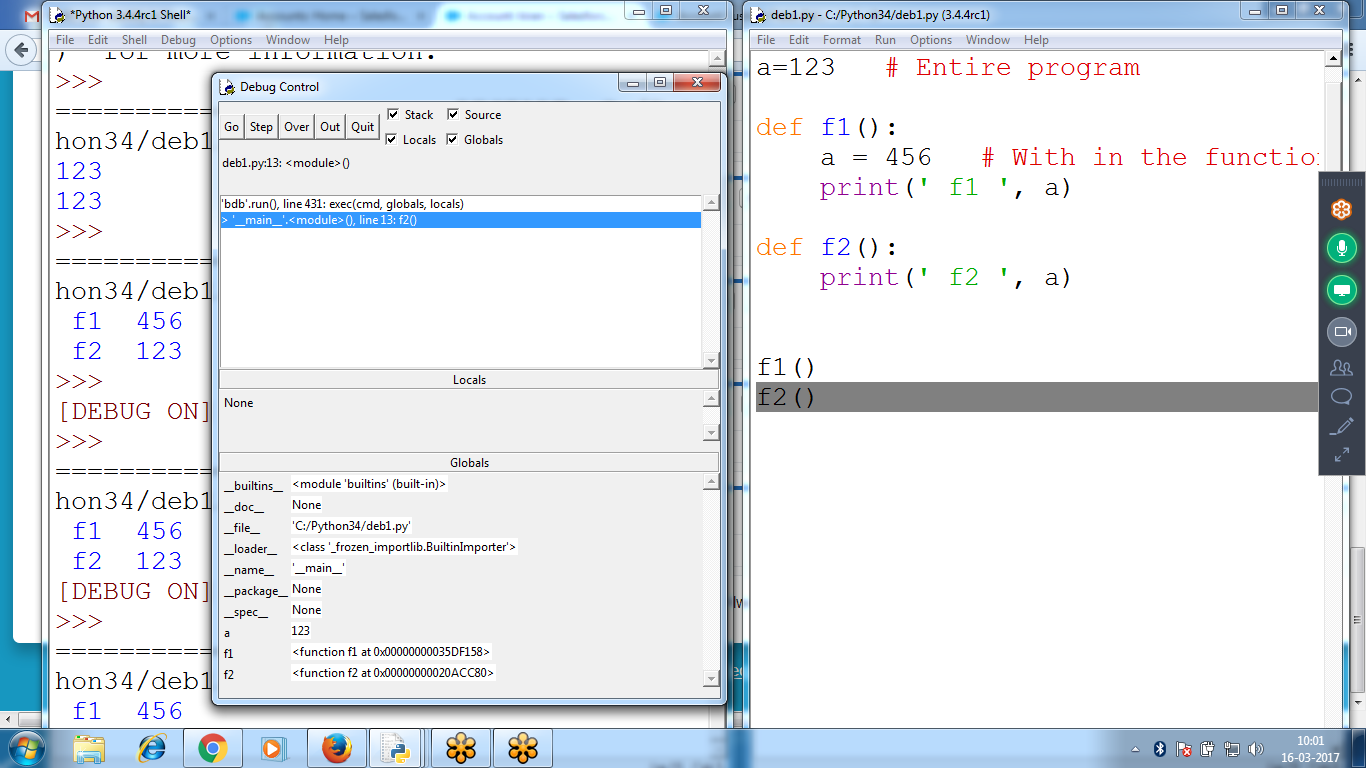


a=123 is Global

a=456 is Local



Outside the function, can access Global variables, local can not access



Program 2 :: Debugging using Functions

In print itself having fm(), fd()

On selection of “step”, enters into fm() first line code

def fm(a,b):

print('value a = ',a)

print('Value b = ',)

c = a \*b

return c

def f1():

a = int(input('Enter value for a '))

b = int(input('Enter value for b '))

**print(' Mul a, b = ', fm(a,b))**

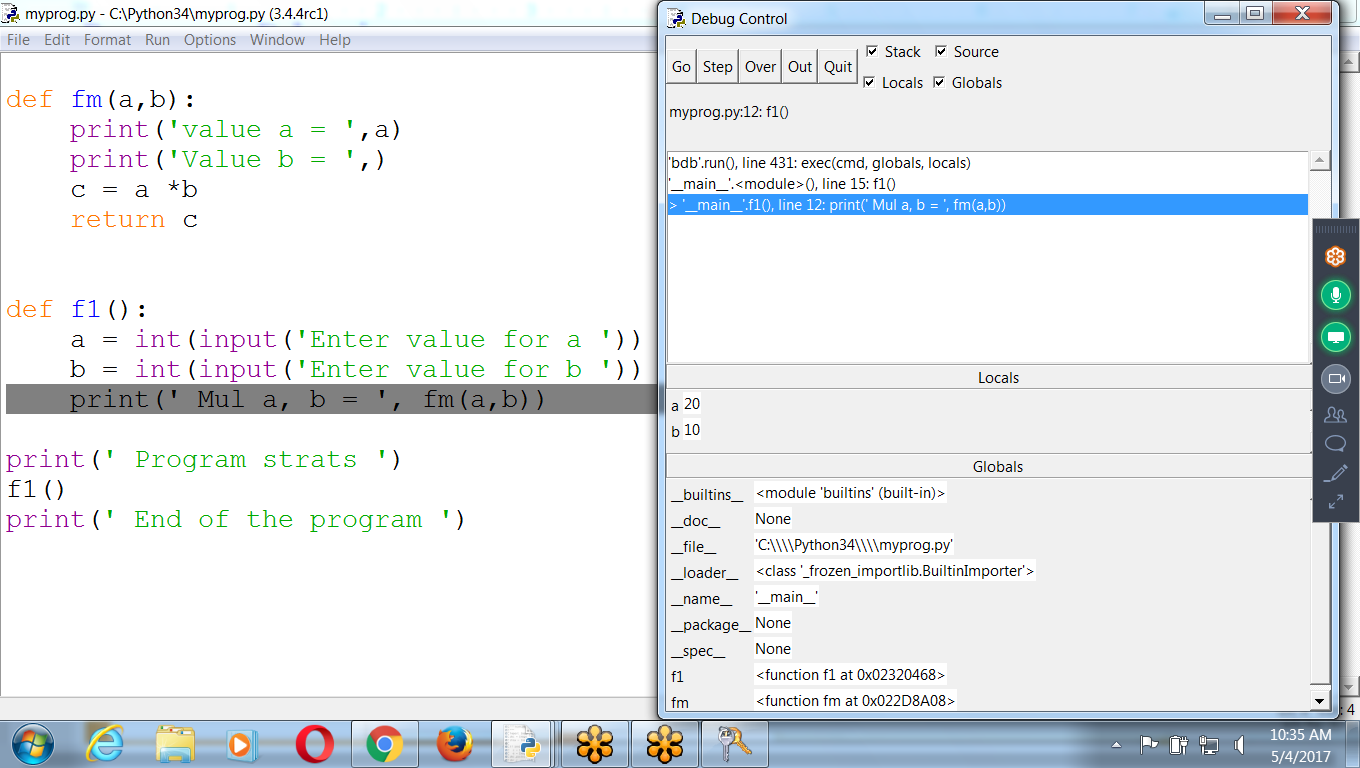
print(' Program strats ')

f1()

print(' End of the program ')

Control is at print(fm() ) and calling fm() inside print

On selection of “step” enters to fm() of first line



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

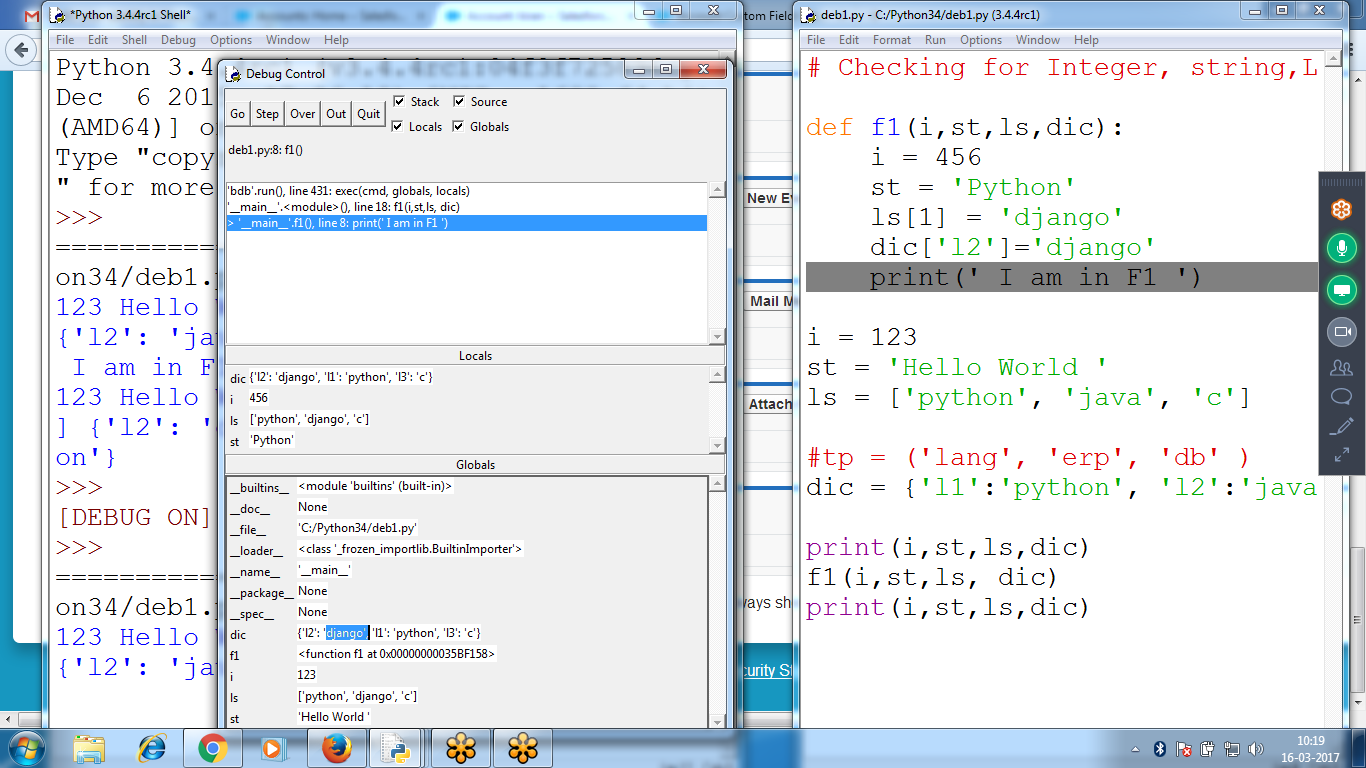
Program 3::

Process with integer, string,LIST and dictionary type values

Integer and string are IMMUTABLES, can’t MODIFY

List, Dictionary are Mutable, CAN Modify

Ls, dic are mutable, after changing effects in Local and global variables



>>> st = 'python program'

>>> st[2] = 't'

Traceback (most recent call last):

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

st[2] = 't'

TypeError: 'str' object does not support item assignment

>>> ls = ['a','b','c']

>>> ls[1] = 'babu'

>>> ls

['a', 'babu', 'c']

>>> dic = {'l1':'python', 'l2':'java'}

>>> dic['l2'] = 'c#'

>>> dic

{'l2': 'c#', 'l1': 'python'}

>>>