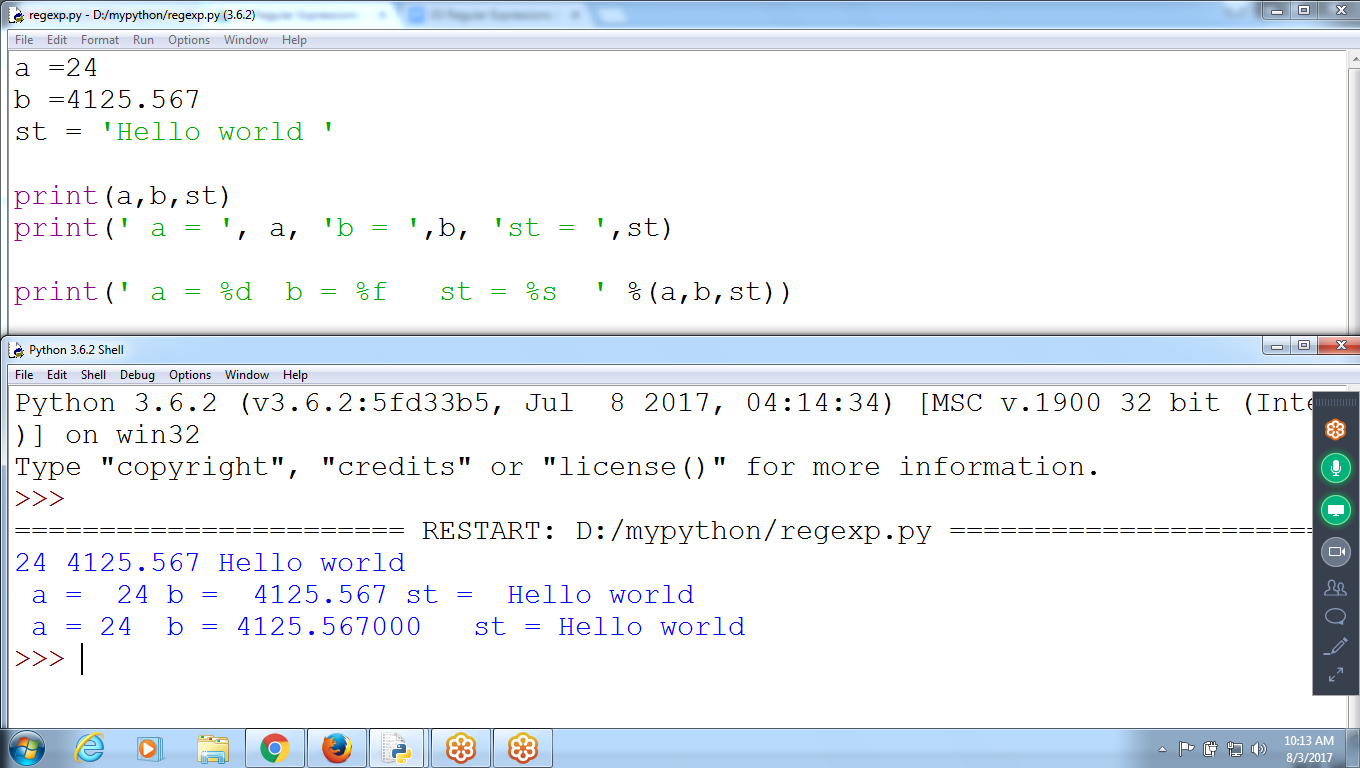
**String Memory Management**



a =24

b =4125.567

st = 'Hello world '

print(a,b,st)

print(' a = ', a, 'b = ',b, 'st = ',st)

print(' a = %d b = %f st = %s ' %(a,b,st))

>>> st

'python program'

>>> st[::1]

'python program'

>>> st[::2]

'pto rga'

>>> st

'python program '

>>> st[::-1]

' margorp nohtyp'

>>> st[::-2]

' agr otp'

>>>

>>> st= 'python program'

>>> a = 24

>>> b = 14.56

>>> print(' a = ', a, 'st = ', st, 'b = ', b)

a = 24 st = python program b = 14.56

>>> print(' a = %d st = %s b = %f ' %(a, st,b))

a = 24 st = python program b = 14.560000

>>>

>>> b

14.56

>>> print(' value b = %10.4f ' %(b))

value b = 14.5600

>>> print(' value b = %3.4f ' %(b))

value b = 14.5600

>>> print(' value b = %f ' %(b))

value b = 14.560000

>>>

**Concatenation**

>>> st1 = 'python program'

>>> st2 = 'language'

>>> st3 = st1 + st2

>>> st3

'python programlanguage'

>>>

**String Memory Management**

fs = 'Hello'

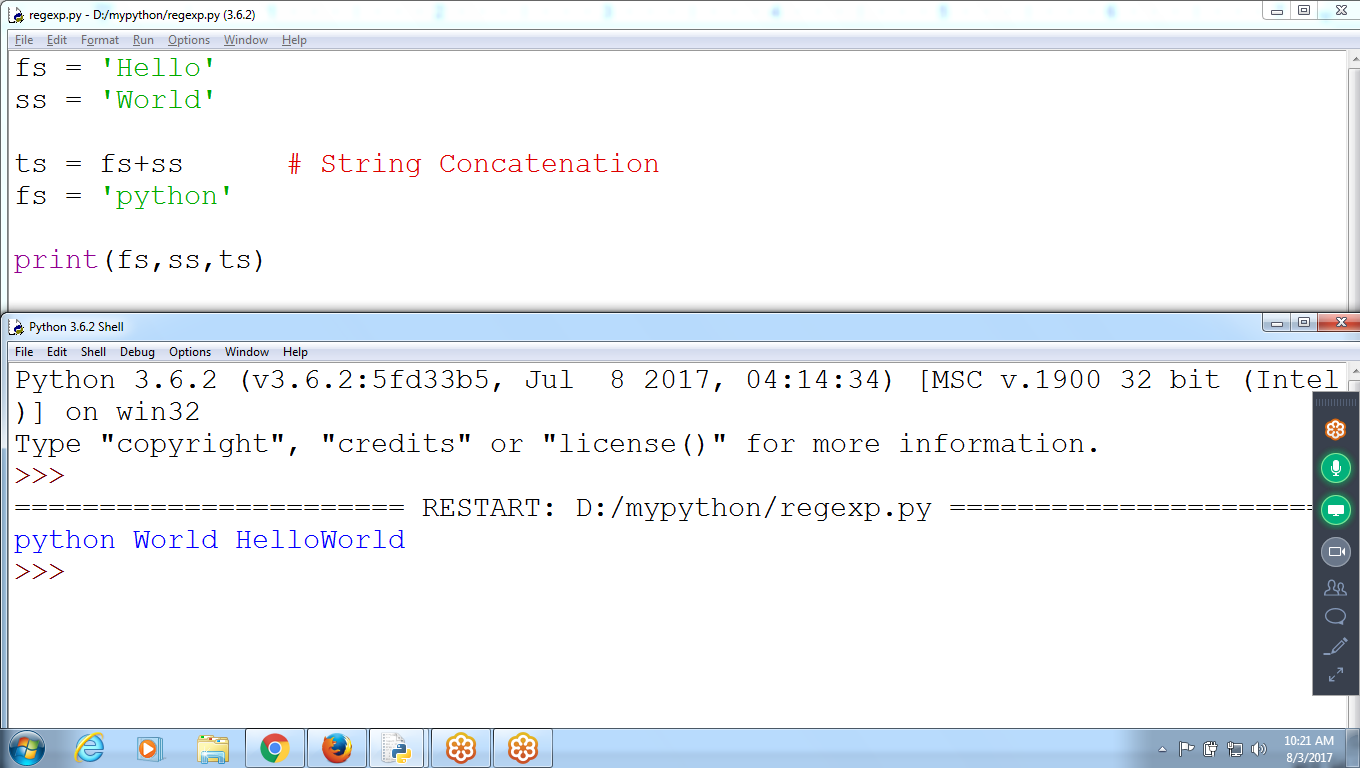
ss = 'World'

#Fs[0] = ‘z’ # indexing is not allowed to assign new value

ts = fs+ss # String Concatenation

fs = 'python' # But can assign new value

print(fs,ss,ts)



fs = 'Hello'

ss = 'World'

ts = fs+ss # String Concatenation

fs = 'python'

print(fs,ss,ts)

Fs Hello

8000 6000

**python**

**8000**

Ss world

7000 7000

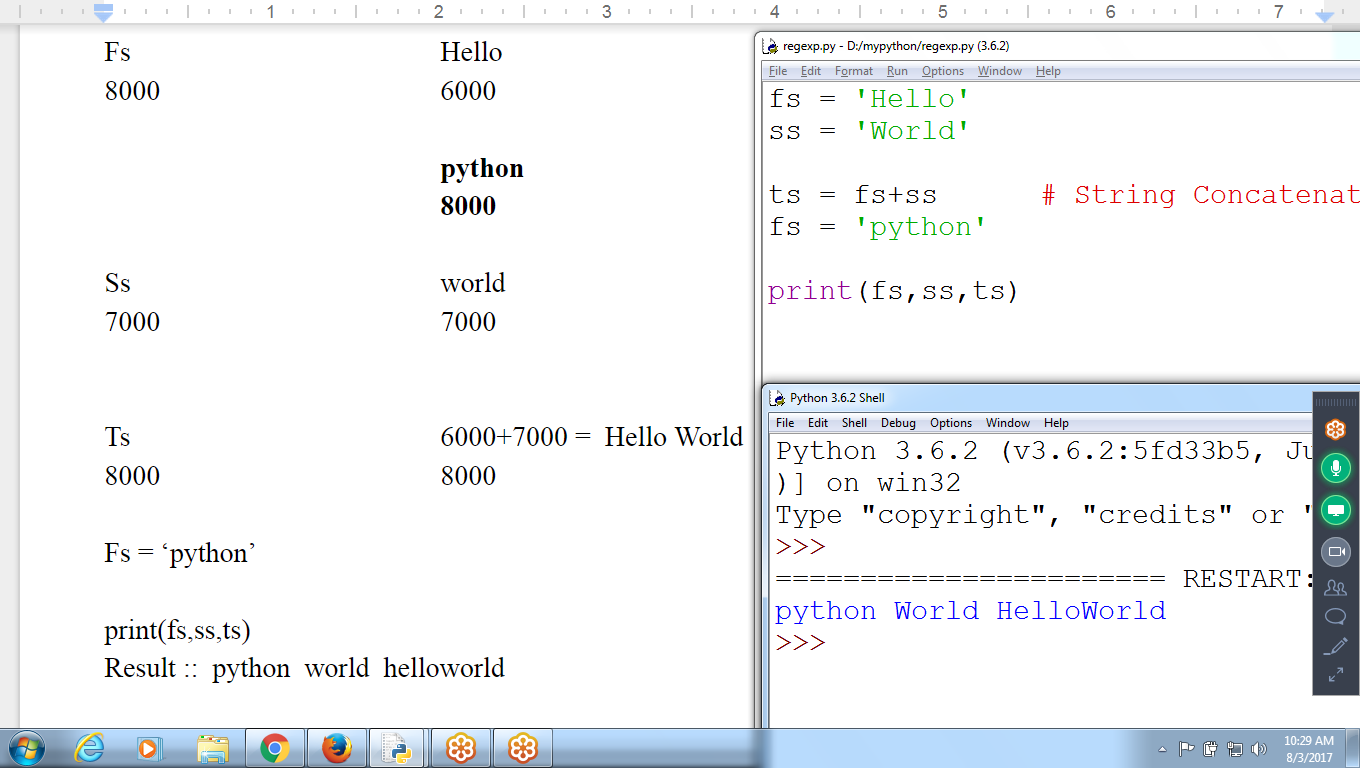
Ts 6000+7000 = Hello World

8000 8000

Fs = ‘python’

print(fs,ss,ts)

Result :: python world helloworld



**NEW Value to String**

**>>> st = 'python' # Storing at location 5001 to 5006**

**>>> st='language' # Now st stores at NEW address 6001 to 6005**

**# Reassigning NEW value to st**

**>>> st[2] = 'z' # at 6002 assigning new character ‘z’**

**Traceback (most recent call last):**

**File "<pyshell#37>", line 1, in <module>**

**st[2] = 'z'**

**TypeError: 'str' object does not support item assignment**

**>>>**

**STRINGS are IMMUTABLE Objects, can’t MODIFY structure**

**St[2] = ‘z’ # Not allowed in PYTHON Strings**

**IN Java, C# allowed**

**St = ‘python’**

**St[2] = ‘z’**

**# String Management**

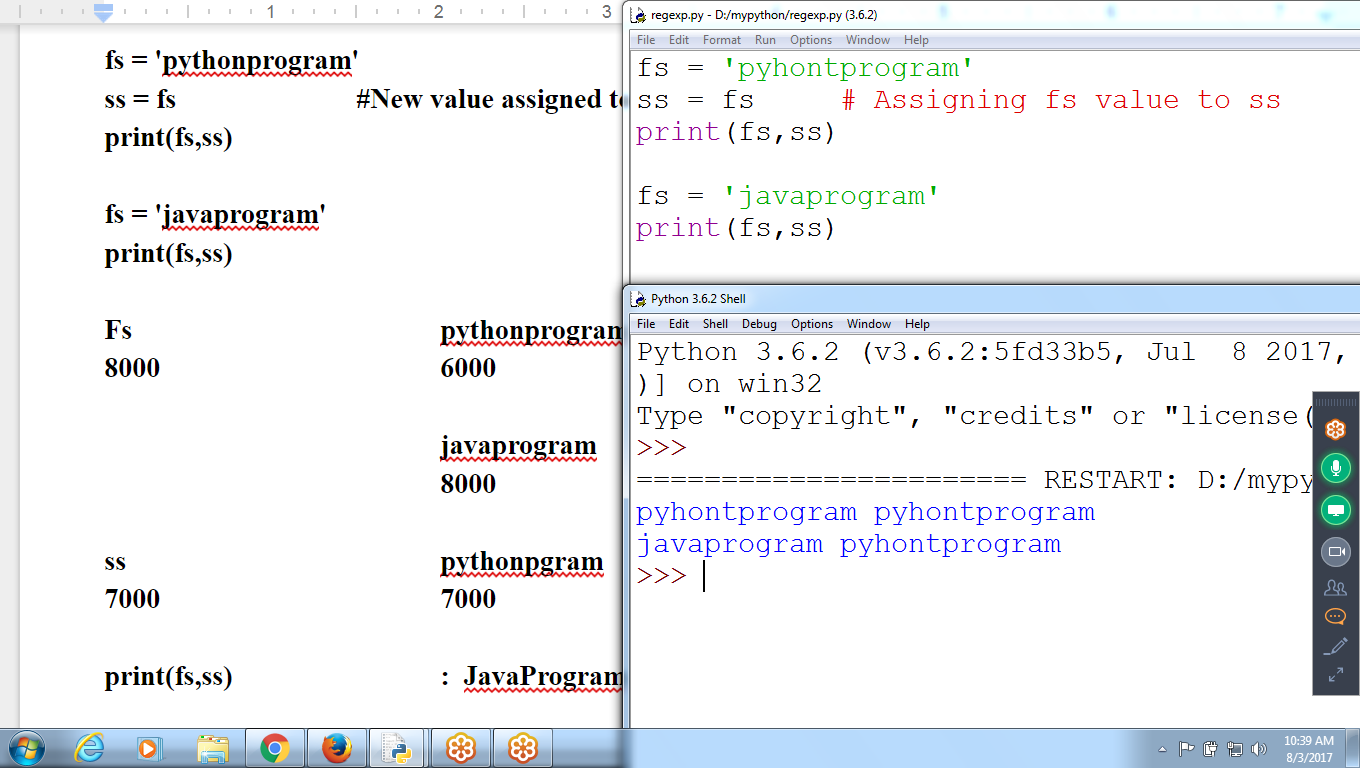
**fs = 'pythonprogram'**

**ss = fs #New value assigned to SS**

**print(fs,ss)**

**fs = 'javaprogram'**

**print(fs,ss)**

****

**Fs pythonprogram**

**8000 6000**

**javaprogram**

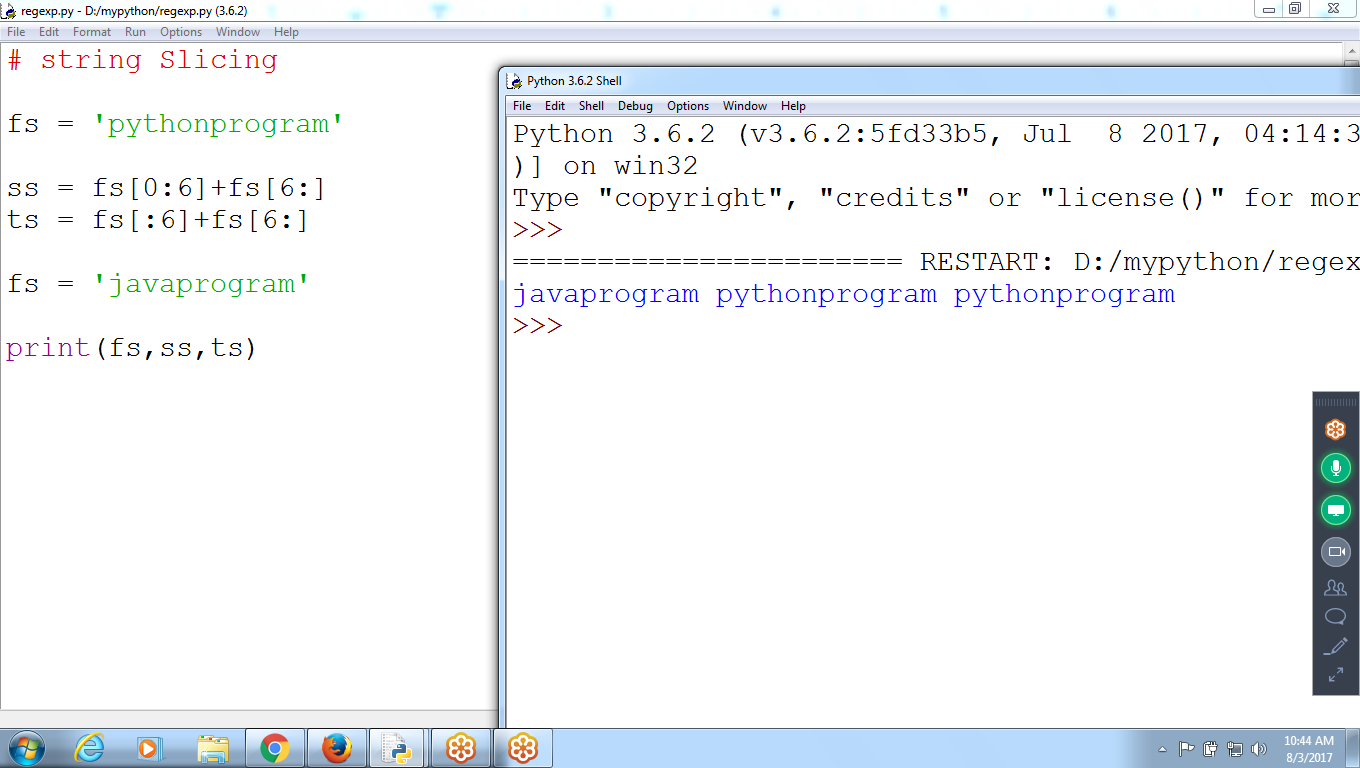
**8000**

**ss pythonpgram**

**7000 7000**

**print(fs,ss) : JavaProgram Pythonprogram**

**Using String Slicing**

****

**# string Slicing**

**fs = 'pythonprogram'**

**ss = fs[0:6]+fs[6:]**

**ts = fs[:6]+fs[6:]**

**fs = 'javaprogram'**

**print(fs,ss,ts)**

### **The format() Method for Formatting Strings**

* The format() method that is available with the string object is very versatile and powerful in formatting strings.
* Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.
* We can use positional arguments or keyword arguments to specify the order.

format strings like the old sprintf() style used in C programming language. We use the % operator

st='python'

r=1

print(st, ' Ranks ', r)

print(' %s Ranks %d ' %(st,r))

a =10

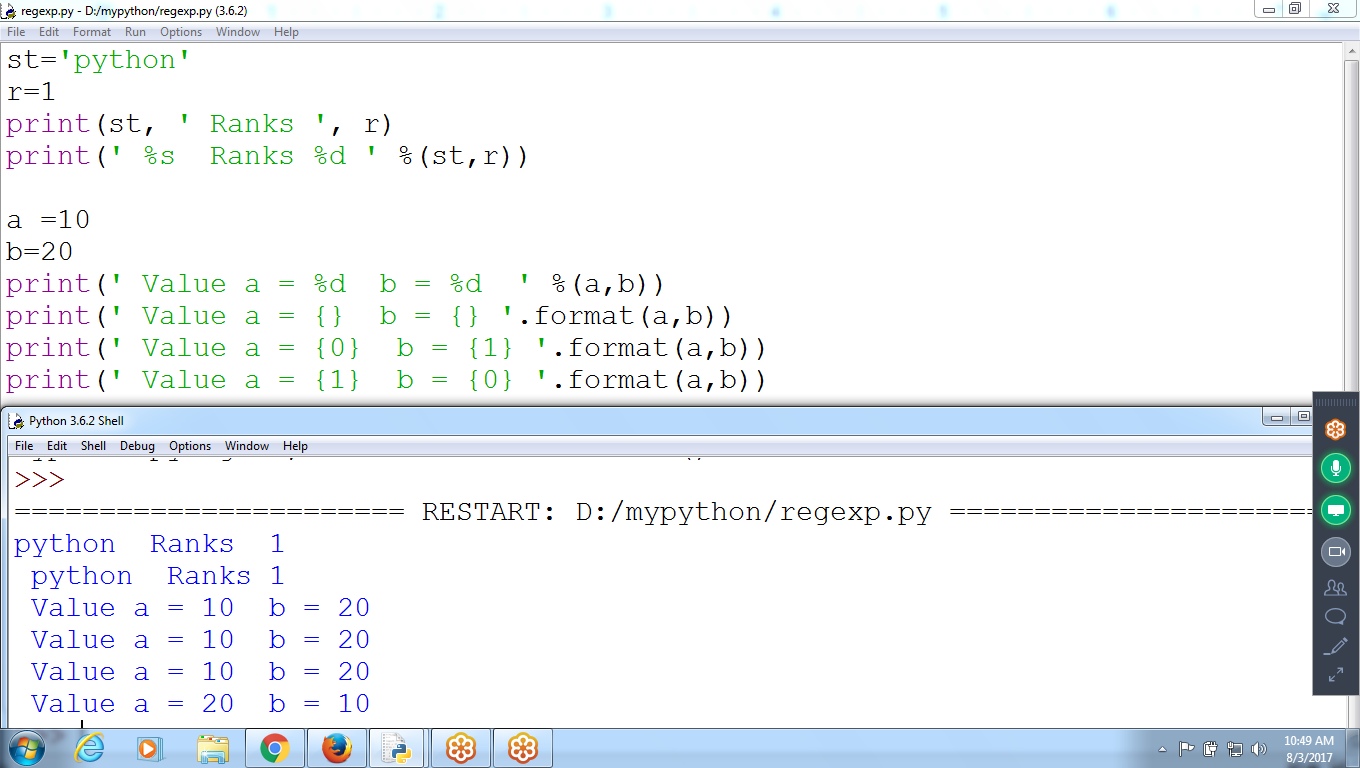
b=20

print(' Value a = %d b = %d ' %(a,b))

print(' Value a = {} b = {} '.format(a,b))

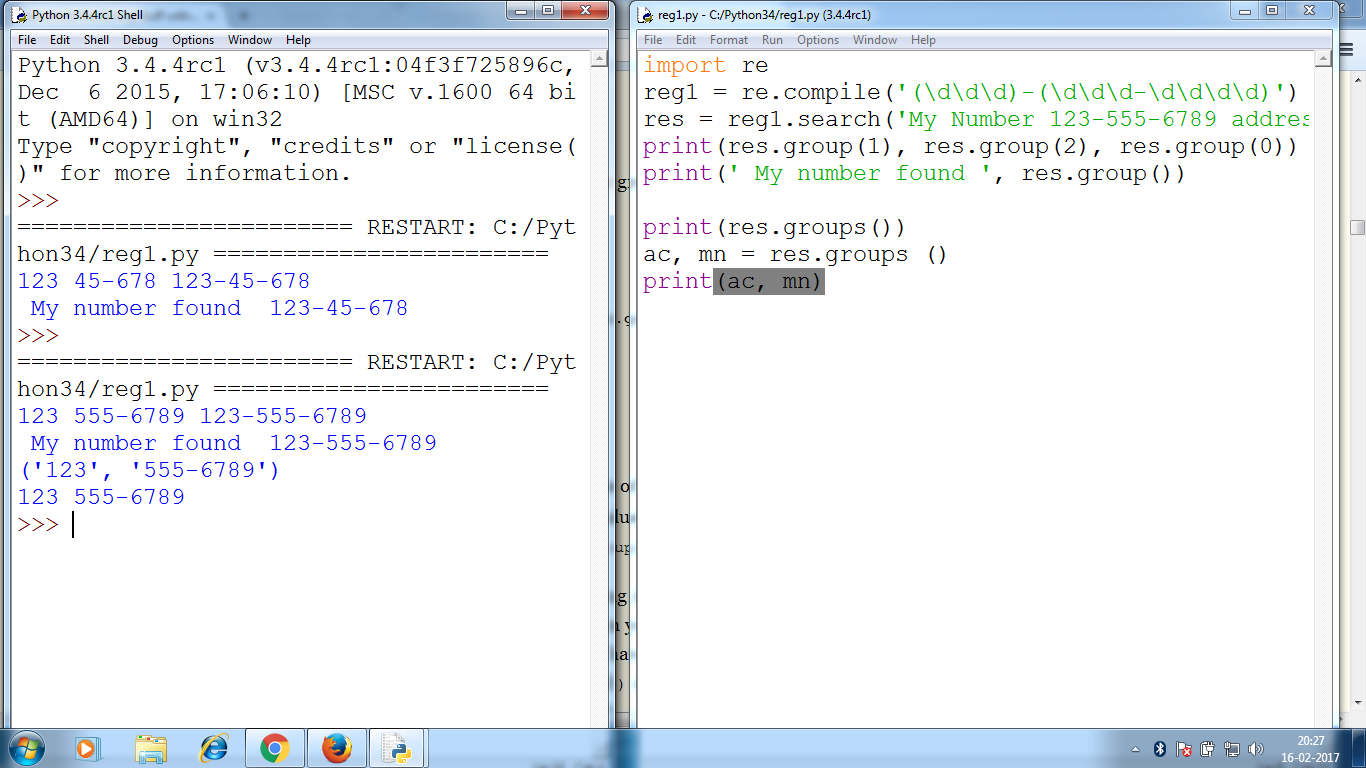
print(' Value a = {0} b = {1} '.format(a,b))

print(' Value a = {1} b = {0} '.format(a,b))



# **More Pattern Matching with Regular Expressions**

Adding parentheses will create ***groups*** in the regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Then you can use the group() match object method to grab the matching text

****

# **To find Phone Number in a string**

To find a phone number in a string. the pattern: three numbers, a hyphen, three numbers, a hyphen, and four numbers. Here’s an example: 415-555-4242.

# Write a program to find given string contains phone number or not

st = input('Enter any string ') #prem994-999-0032 anadn 970470

def fphone(r):

if(len(r) != 12):

return False

for i in range(0,3):

if(not r[i].isdecimal()):

return False

if(r[3] != '-'):

return False

for i in range(4,7):

if(not r[i].isdecimal()):

return False

if(r[7] != '-'):

return False

for i in range(8,12):

if(not r[i].isdecimal()):

return False

return True

for i in range(len(st)):

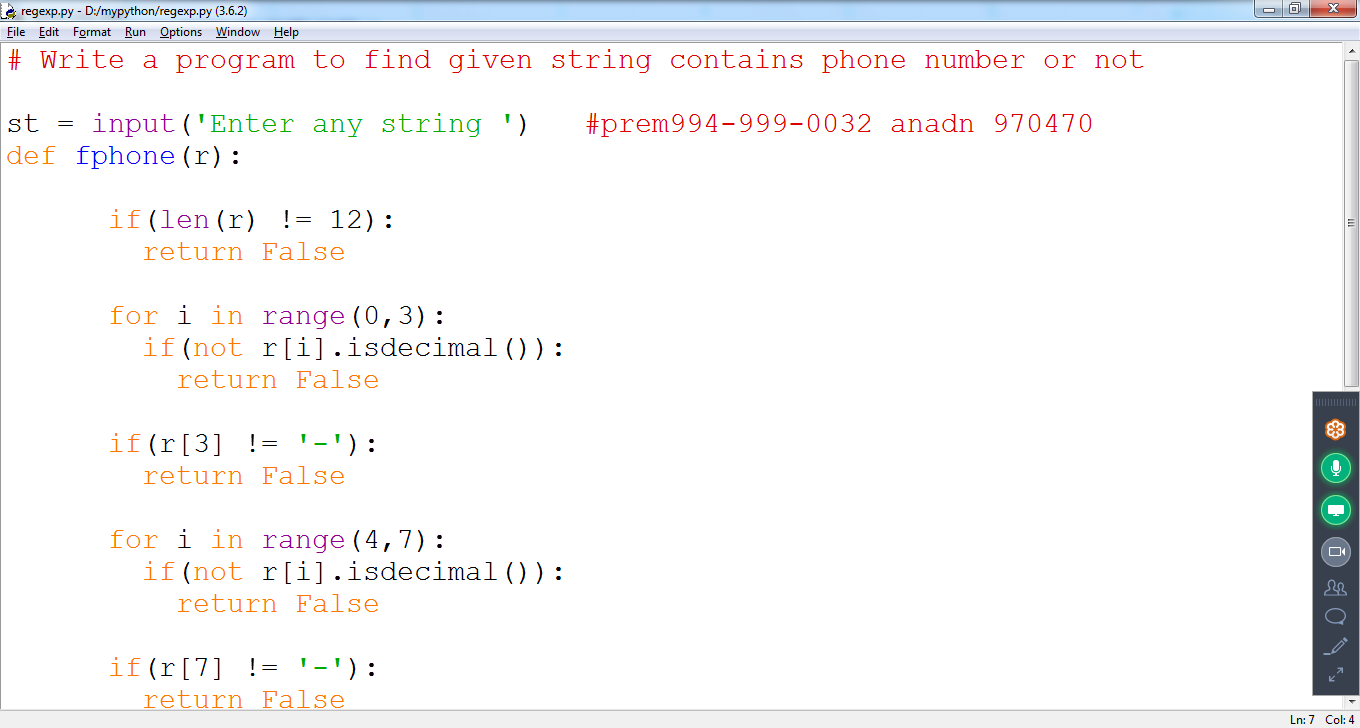
res = st[i:i+12]

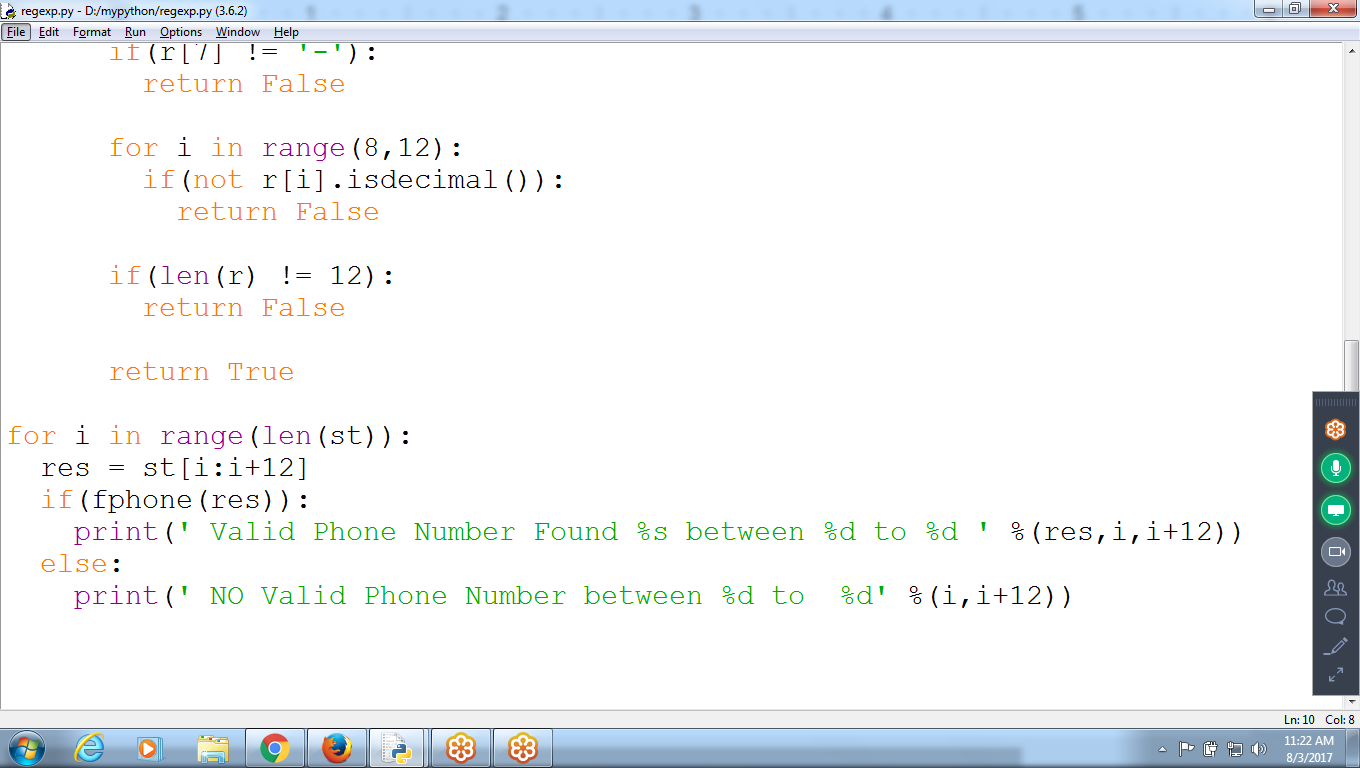
if(fphone(res)):

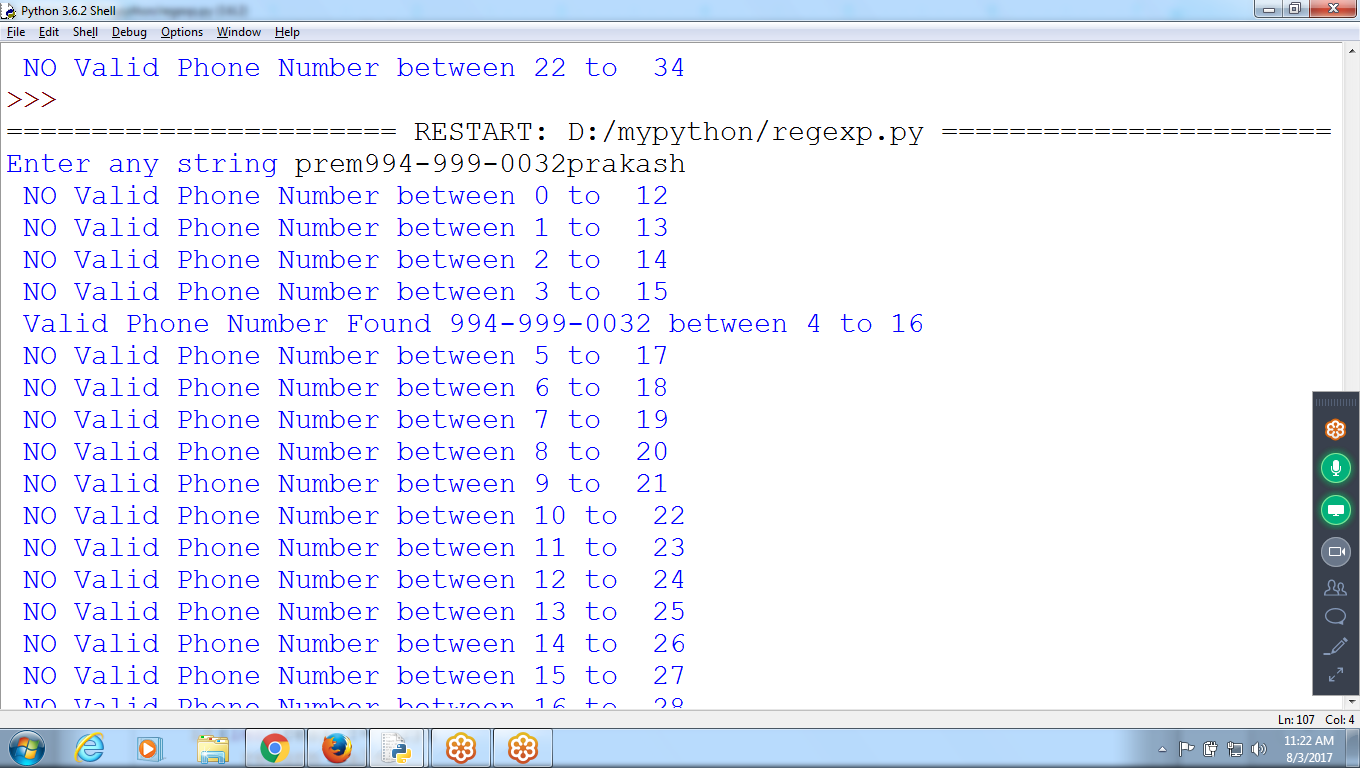
print(' Valid Phone Number Found %s between %d to %d ' %(res,i,i+12))

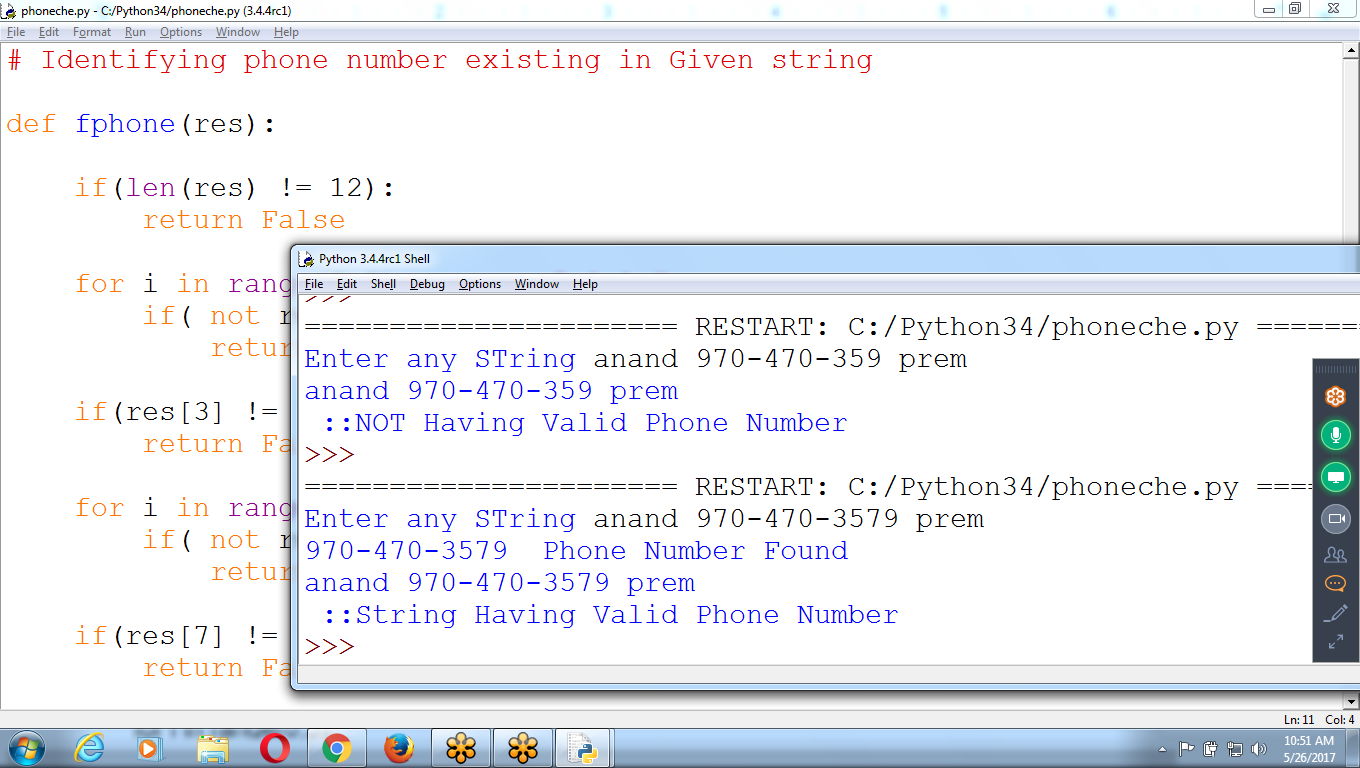
else:

print(' NO Valid Phone Number between %d to %d' %(i,i+12))



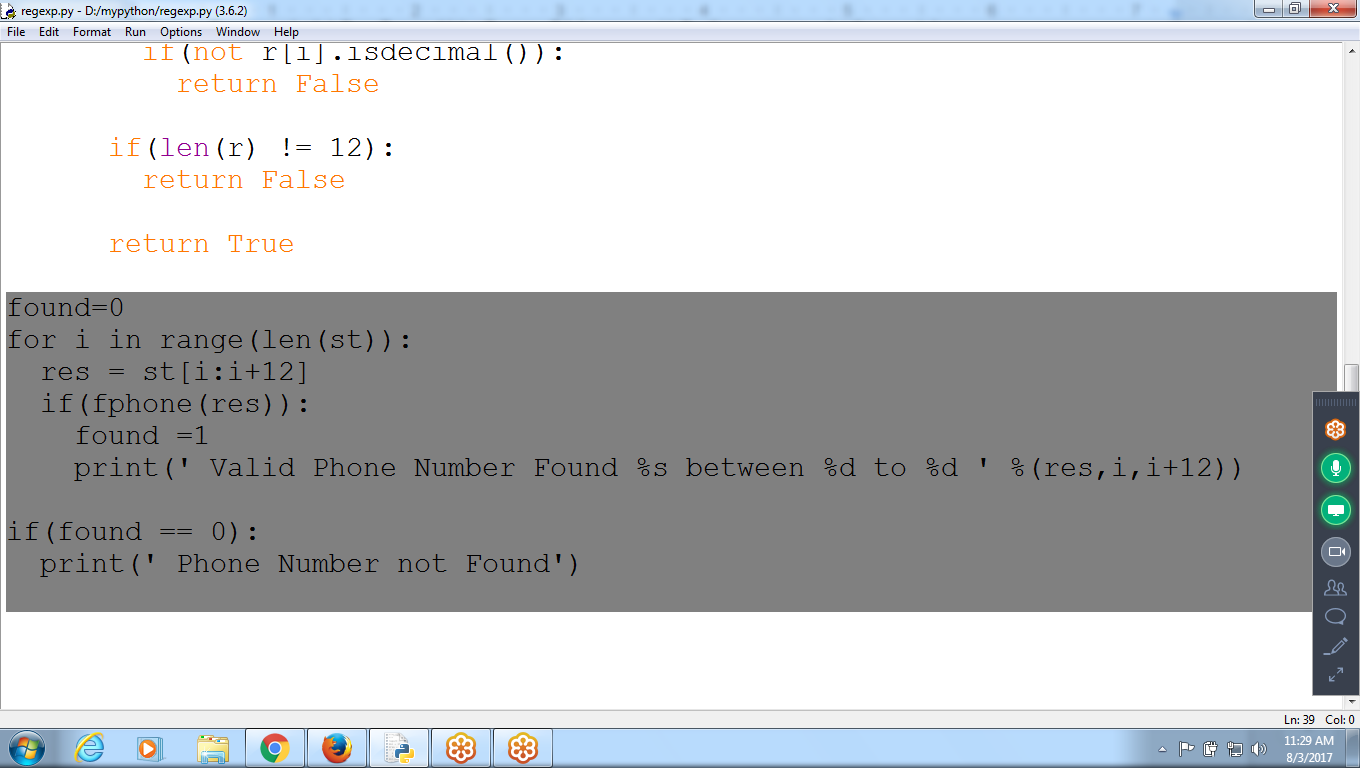




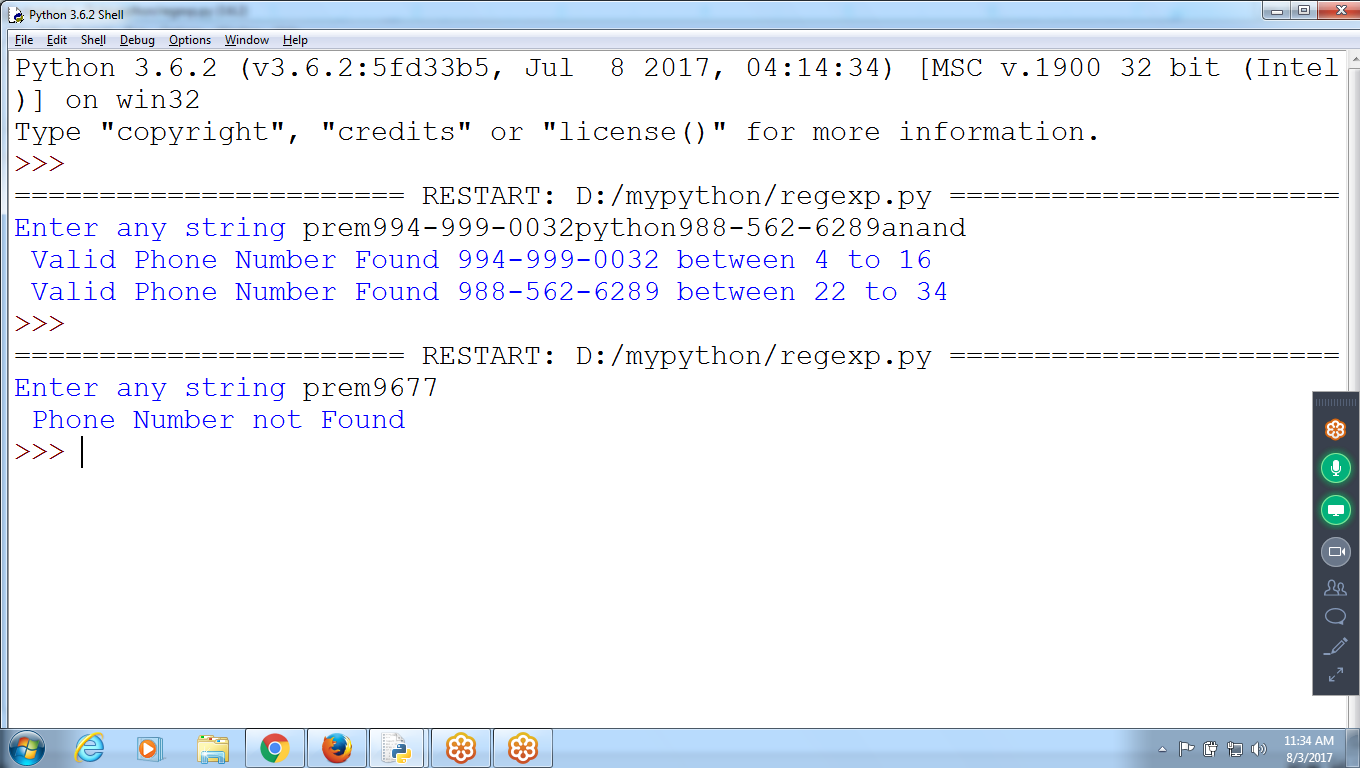


**Second Option::: should not display Phone number not found**

**Change The Code**

****

**Result::**

****

# **Regular Expressions**

a phone number formatted like 415.555.4242 or (415) 555-4242? if the phone number had an extension, like 415-555-4242 x99? The phone() function **would fail** to validate them

* Regular expressions, called *regexes* for short, are descriptions for a pattern of text.
* For example, **\d** in a regex stands for a digit character—that is, any **single numeral 0 to 9**

**\d\d\d-\d\d\d-\d\d \d\d (or) \d{3}-\d{3}-\d{4}**

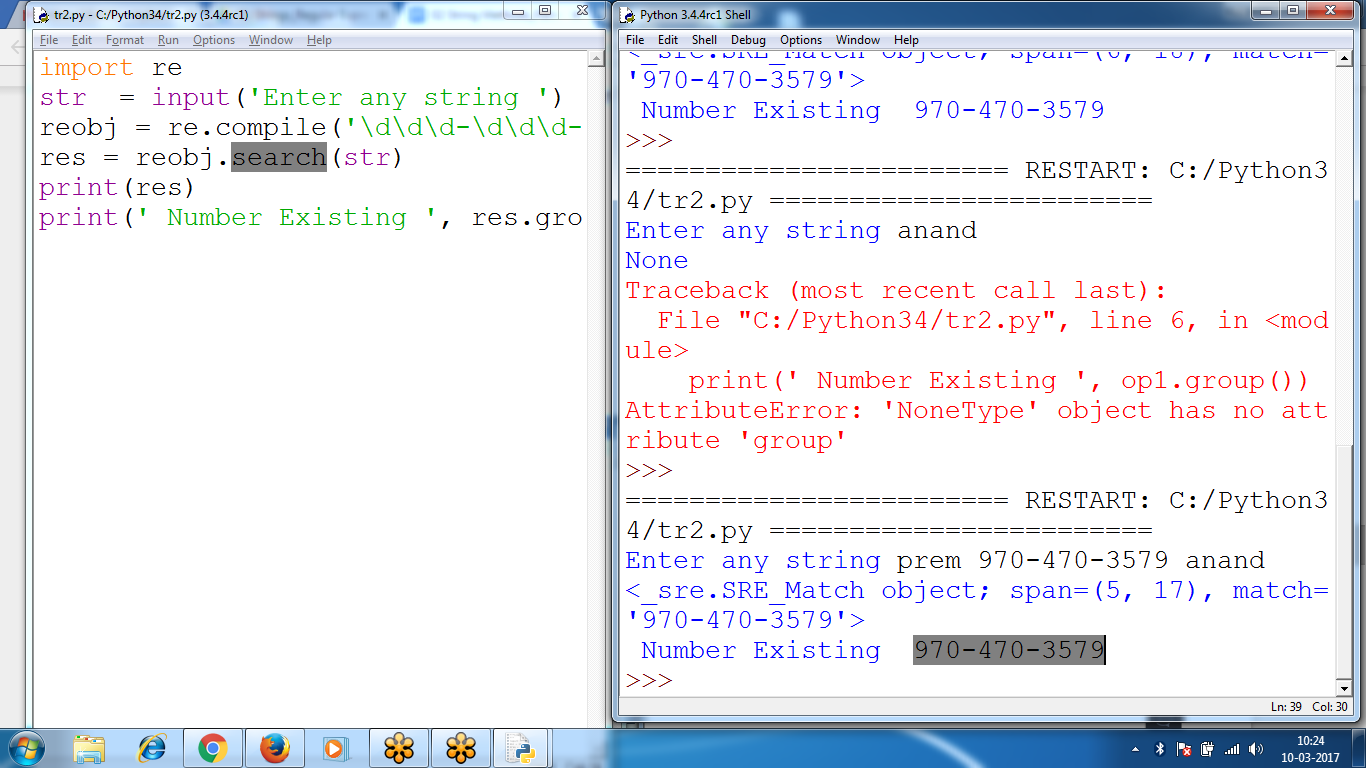
## **Creating Regex Objects**

* All the regex functions in Python are in the **re** module
* Passing a string value representing regular expression to **re.compile()** returns a Regex pattern object
* Search for Phone number in given string

>>> **phoneNumRegex = re.compile('\d\d\d-\d\d\d-\d\d\d\d')**

Now the phoneNumRegex variable contains a Regex object.

Typing r'\d\d\d-\d\d\d-\d\d\d\d' is much easier than typing '\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d'



import re

str = input('Enter any string ')

reobj = re.compile('\d\d\d-\d\d\d-\d\d\d\d')

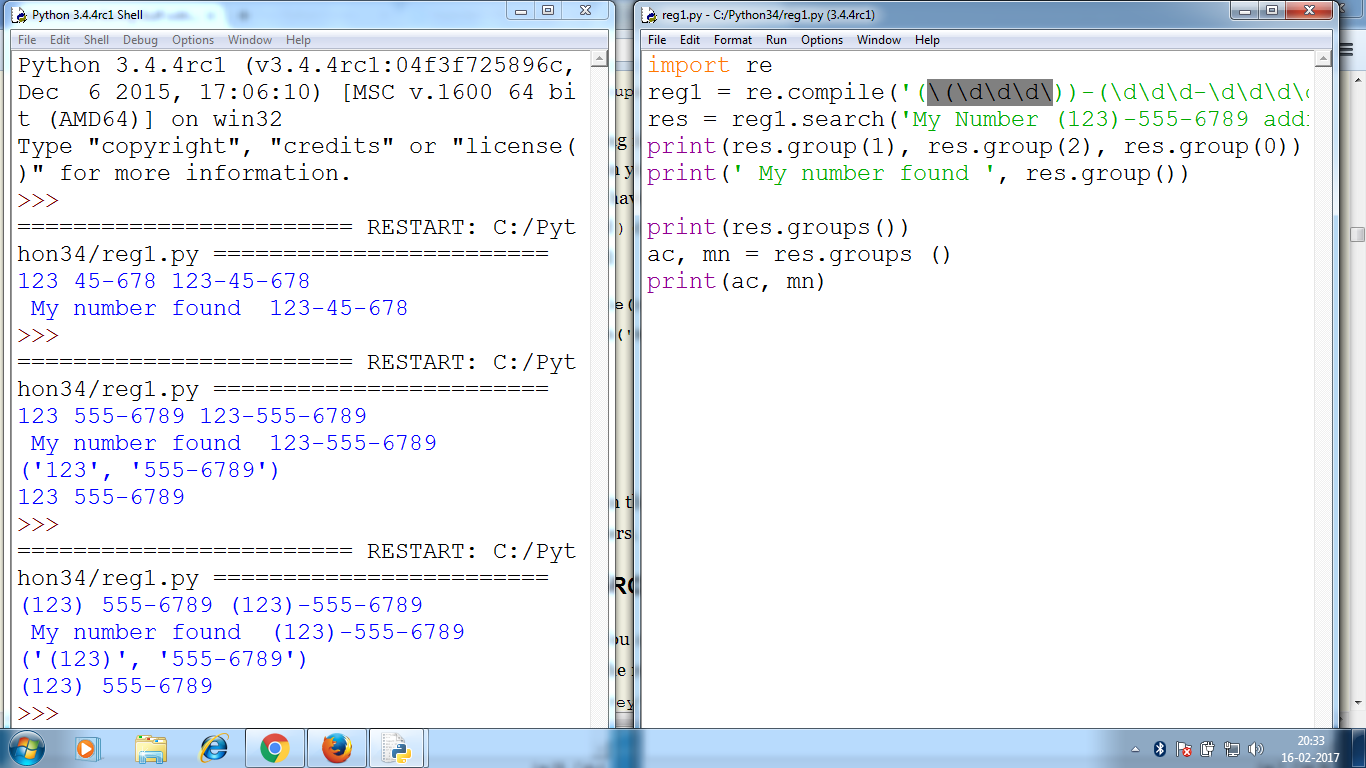
res = reobj.search(str)

print(res)

print(' Number Existing ', res.group())

**if you need to match a parenthesis in text?**

The \( and \) escape characters in the raw string passed to re.compile()

****

default\_order = "{}, {} and {}".format('Prem','Anand','Chaitu')

print('\n--- Default Order ---')

print(default\_order)

# order using positional argument

positional\_order = "{1}, {0} and {2}".format('Prem','Anand','Chaitu')

print('\n--- Positional Order ---')

print(positional\_order)

# order using keyword argument

keyword\_order = "{a}, {c} and {p}".format(p='Prem', a='Anand', c='Chaitu')

print('\n--- Keyword Order ---')

print(keyword\_order)

--- Default Order ---

Prem, Anand and Chaitu

--- Positional Order ---

Anand, Prem and Chaitu

--- Keyword Order ---

Anand, Chaitu and Prem

