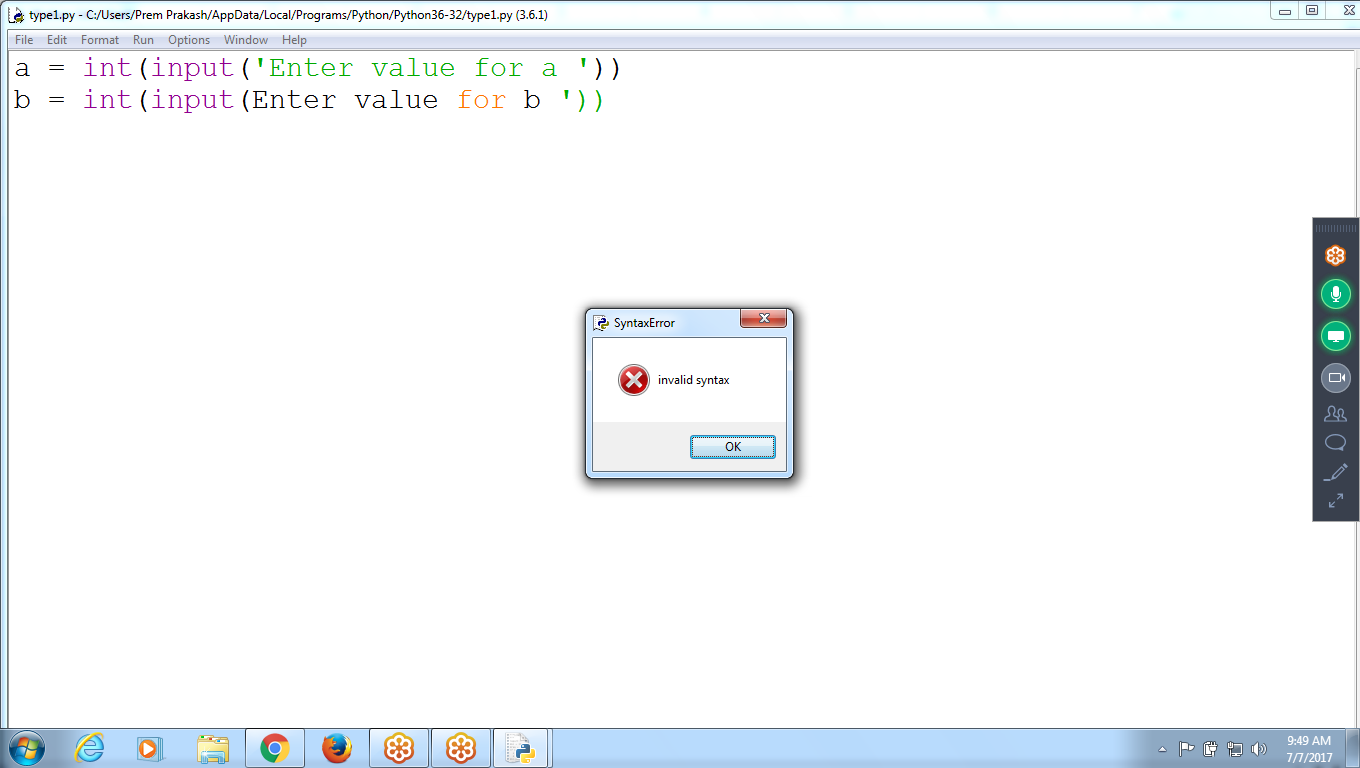
# **Types of Errors and Exception Handling**

# **Syntax Error** : As Programming Rules , if fails throws errors

# \* Encounters before Execution

Syntax Error :: Before execution of the program ::: Single QUotation missing Error



# **Logical Errors :** After Execution, watching output

Compare **Program Result with Expected Result,** if both are different then it is Logical Error

a=10

b=20

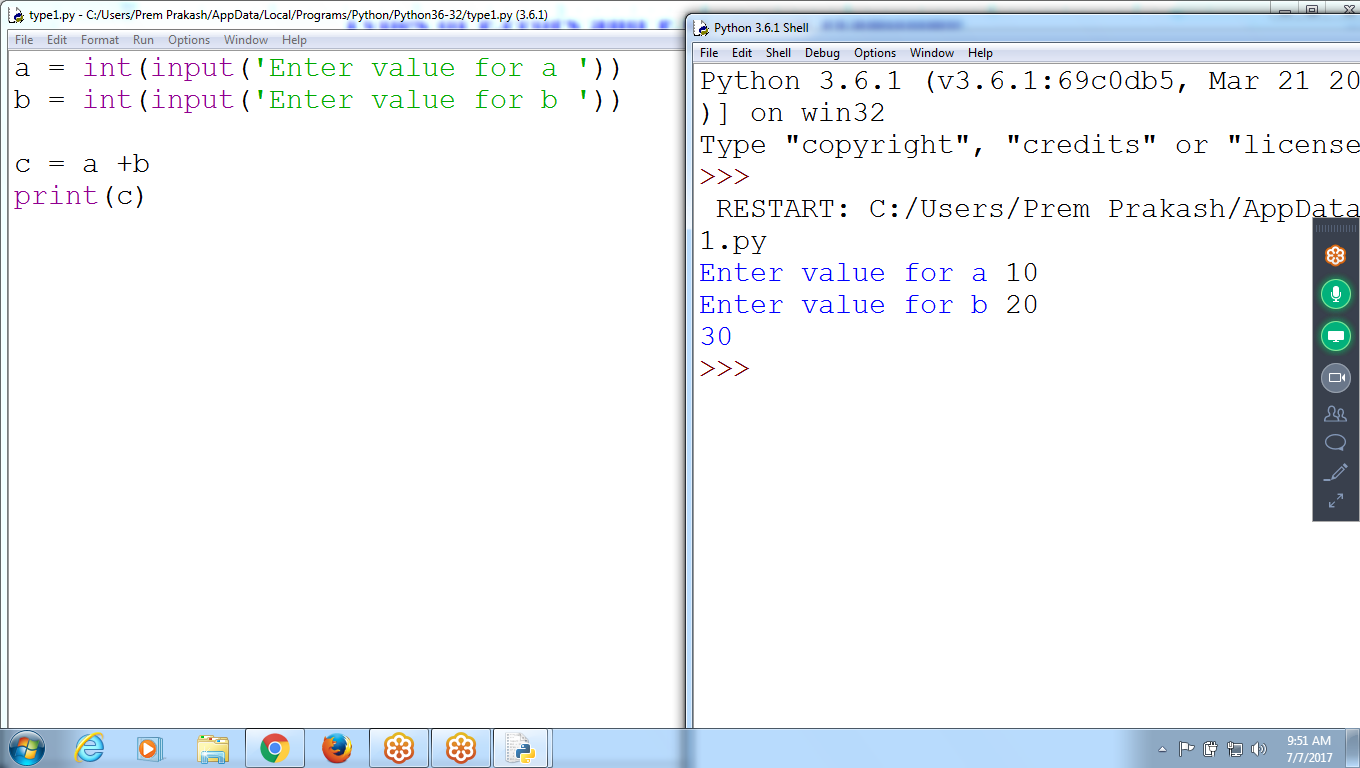
A + b

200

TO get 200 value :: a =10, b=20 Result :: 30

Expecting value :: 200, but result : 30 :: ERROR Type : Logical Error

**Logical Error Encounters after EXecution of the program**

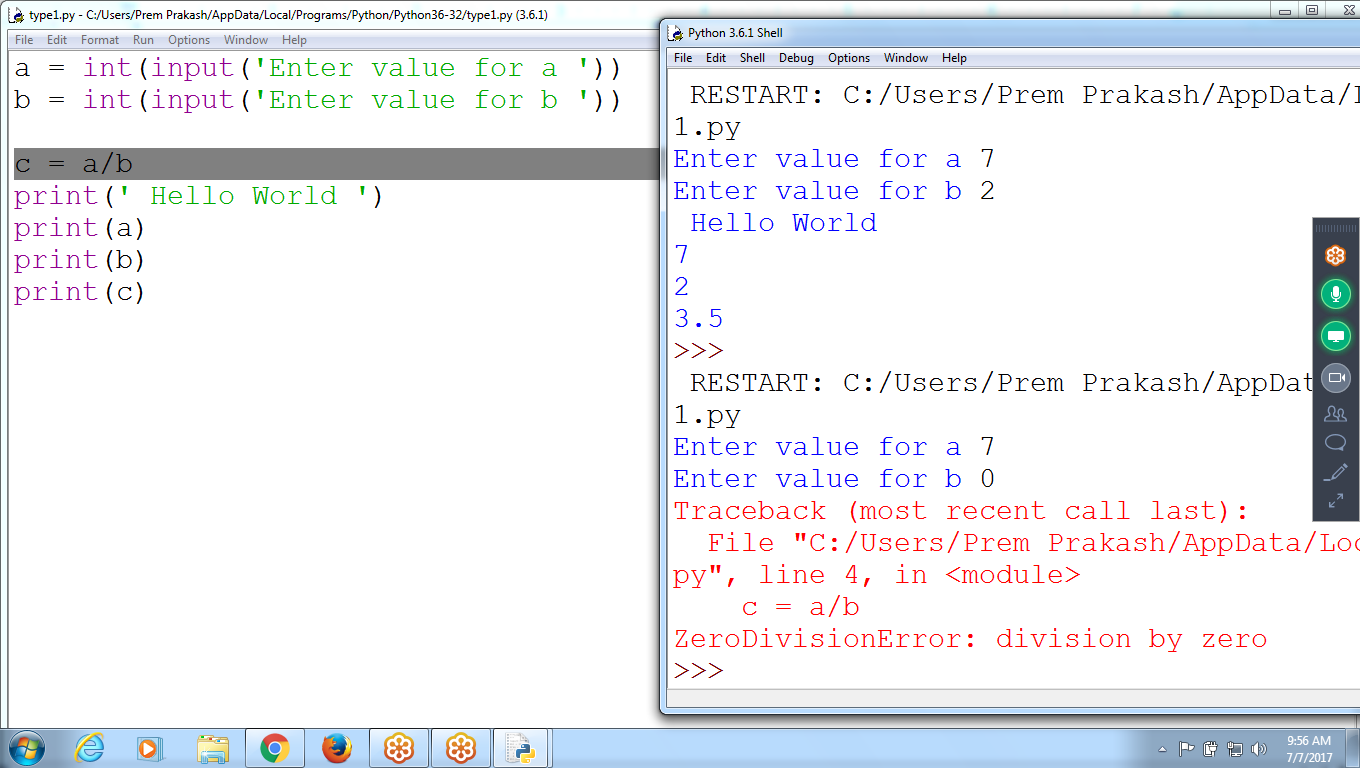


# **Execution Error: At the time of Execution RUNTIME Errors**

# During **Execution of the Program**, Error Encounters it is called **RUNTIME Error**

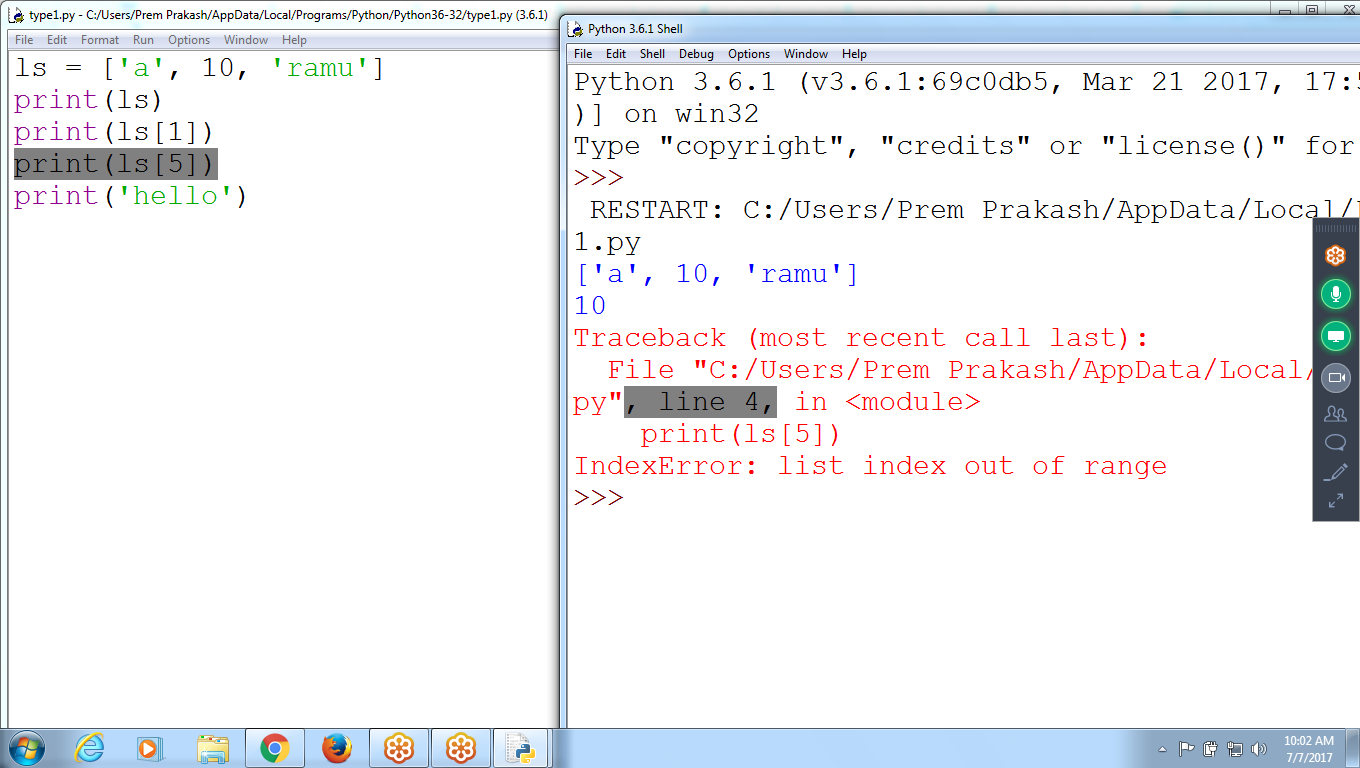
# if any error encountered during execution, program terminates ABNORMALLY, this situation is called **Exception**

Ex: ZERODIVISIONError



**Index Error:**

**Processor can’t Retrieve value at ls[5], terminates Abnormally**



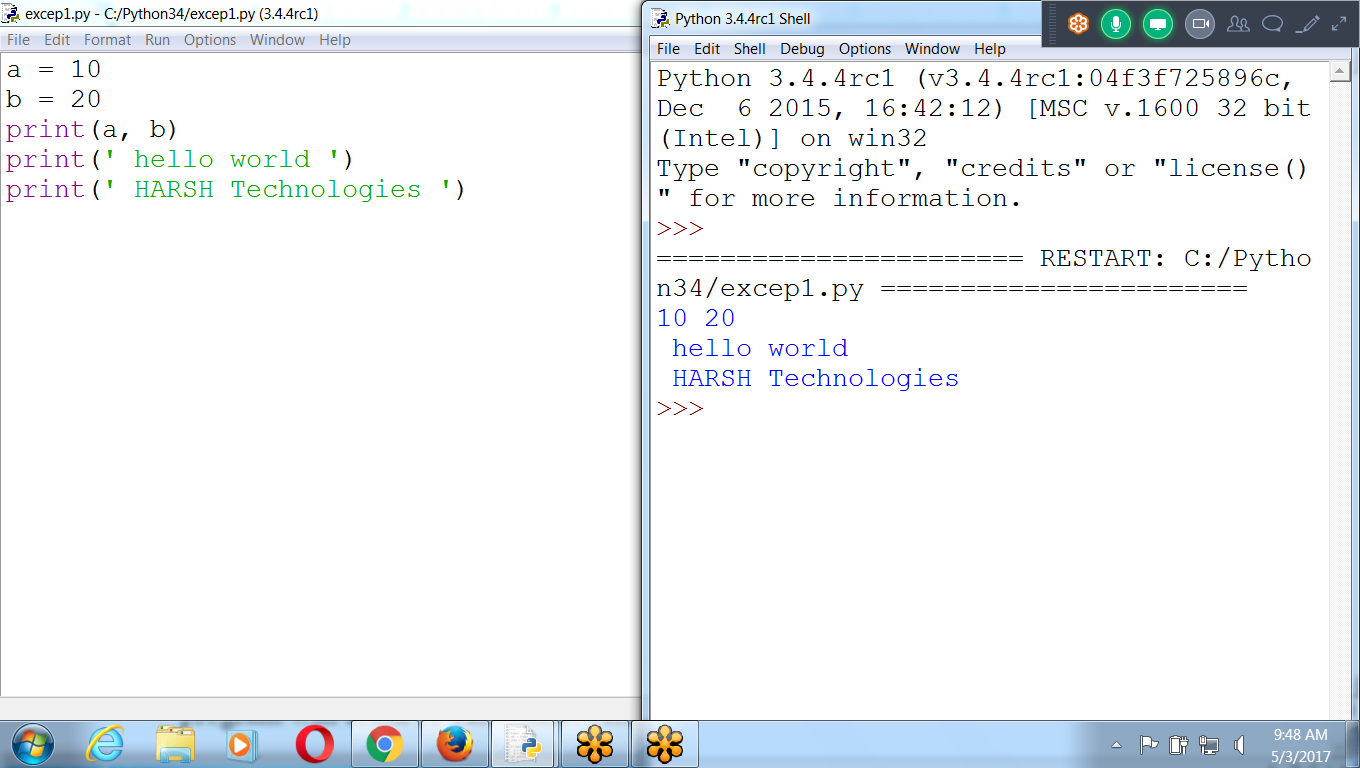
Normal Termination :: To Execute Program, ALL statements must execute Compulsory ::

Abnormal Termination: Execution Terminates abnormally, due to **RUNTIME Error**

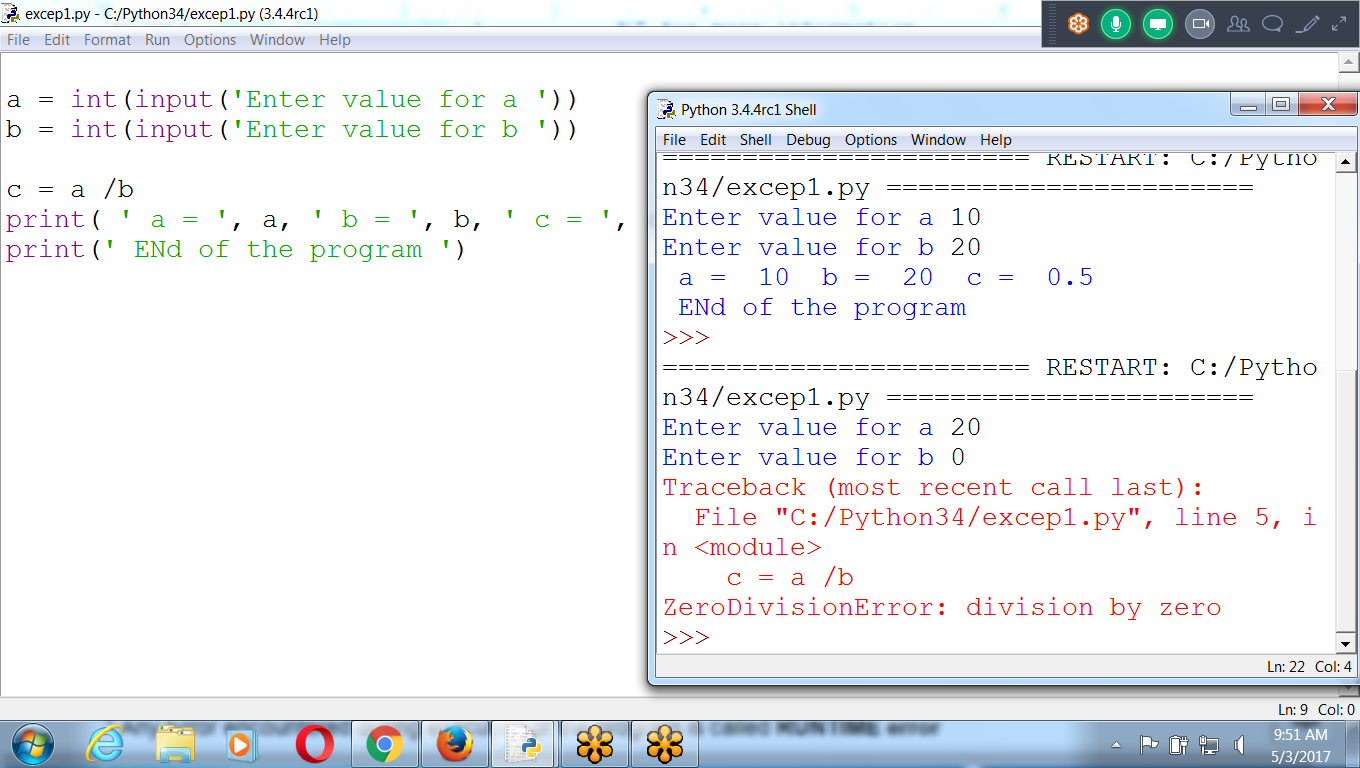
If Abnormal Termination:

1. Provide solution to Processor and continue remaining statements to execute
2. Skip the statement for execution and continue remaining statements to execute

Normal Termination : **ALL Statements are executed Successfully**



Abnormal Termination : Control moves out of the program abnormally



\* Any Error encountered during execution of the program is called **RUNTIME error**

\* IF Runtime error encounters, program terminates **Abnormally** this situation is called **EXCEPTION**

\* if there is Exception, handle them using **Try-Except** block to terminate program **NORMALLy**

# **Exception Handling**

* Getting an **Runtime Error** or *exception* in Python program means the entire program will crash or Terminates Abnormally. Instead, the program to detect errors, handle them, and continue to run.

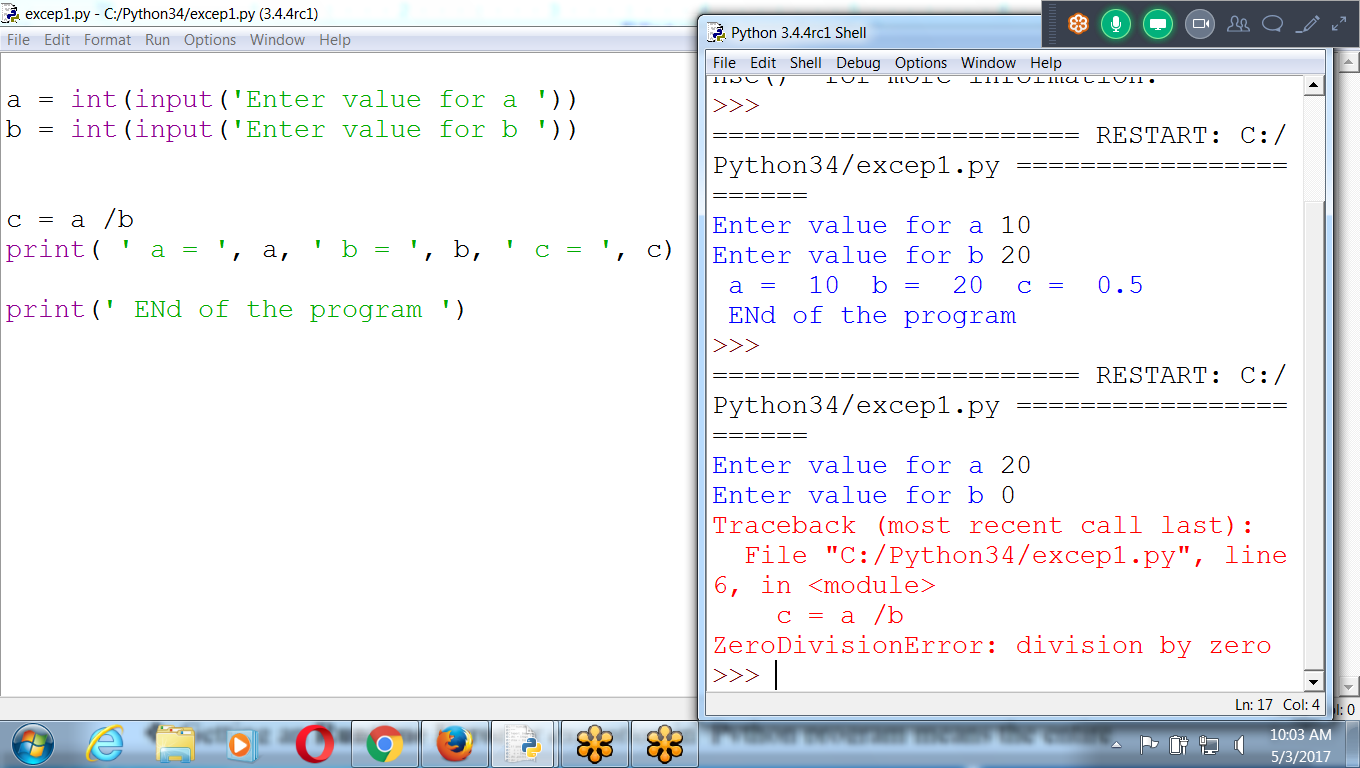
# **Try Block contains statements which may arise Runtime** Error,

# **If No Runtime Error, control executes statements after Except block**

# 

# How to Handle Runtime Errors

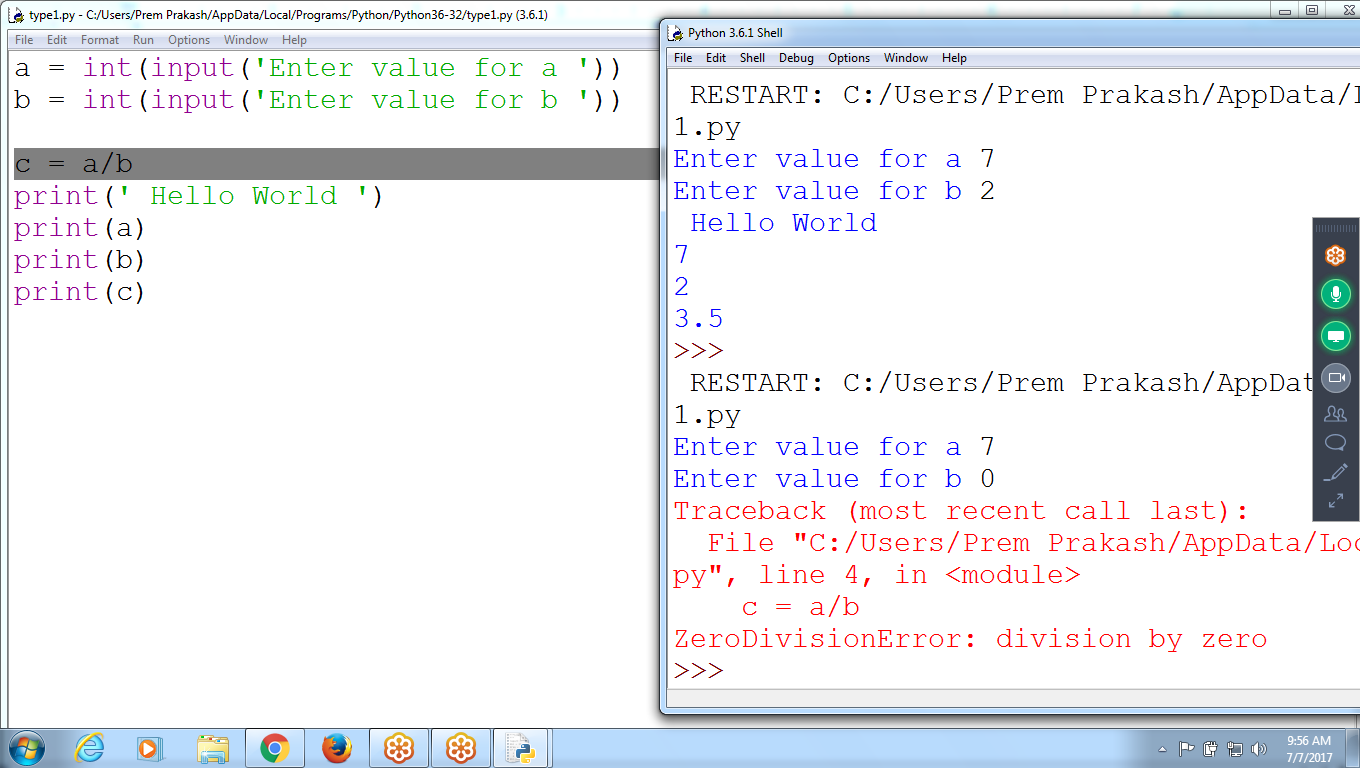
Runtime Error ::



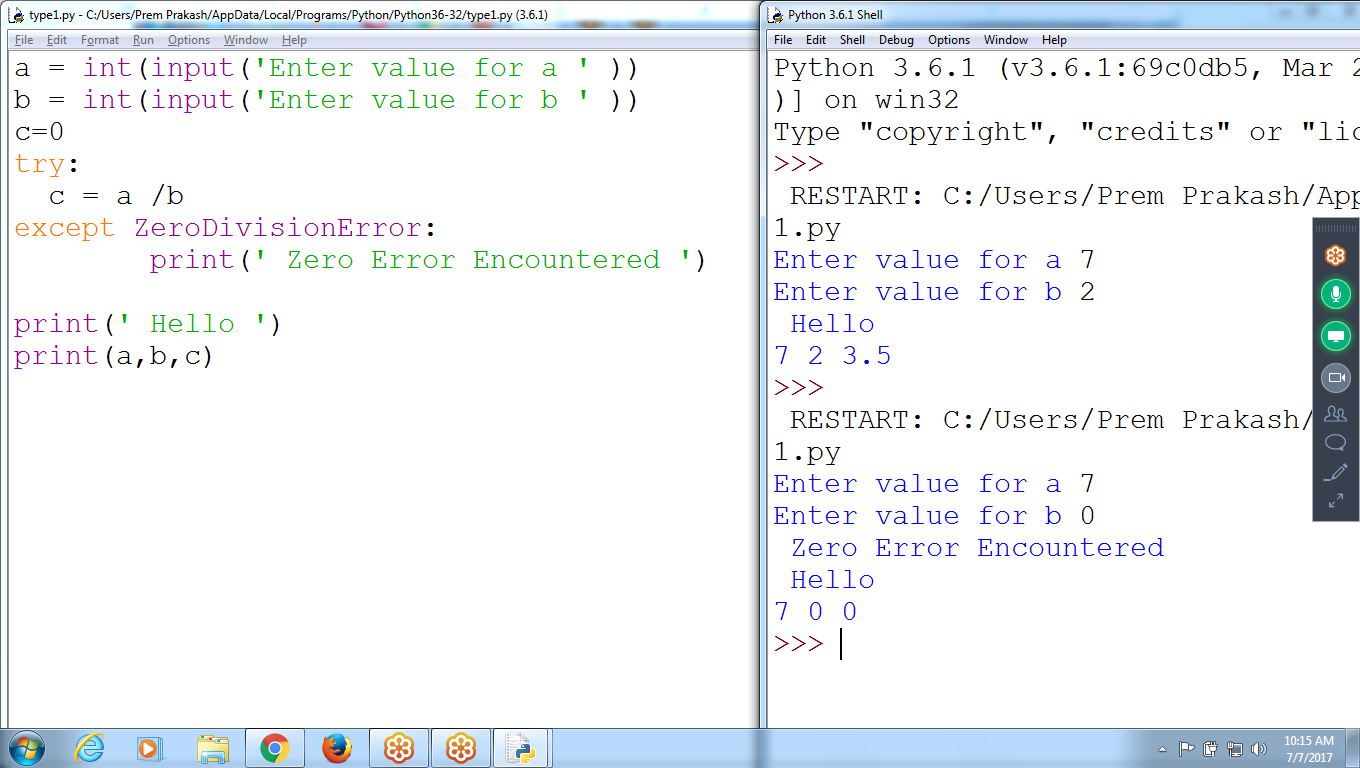
If there is Exception, handle them using **Try-except** block to terminate program **NORMALLy**

**Normal Termination : Use Try- except block**

Example 1: Index Error



Solution



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

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

**c=0**

**try:**

**c = a /b**

**except ZeroDivisionError:**

**print(' Zero Error Encountered ')**

**print(' Hello ')**

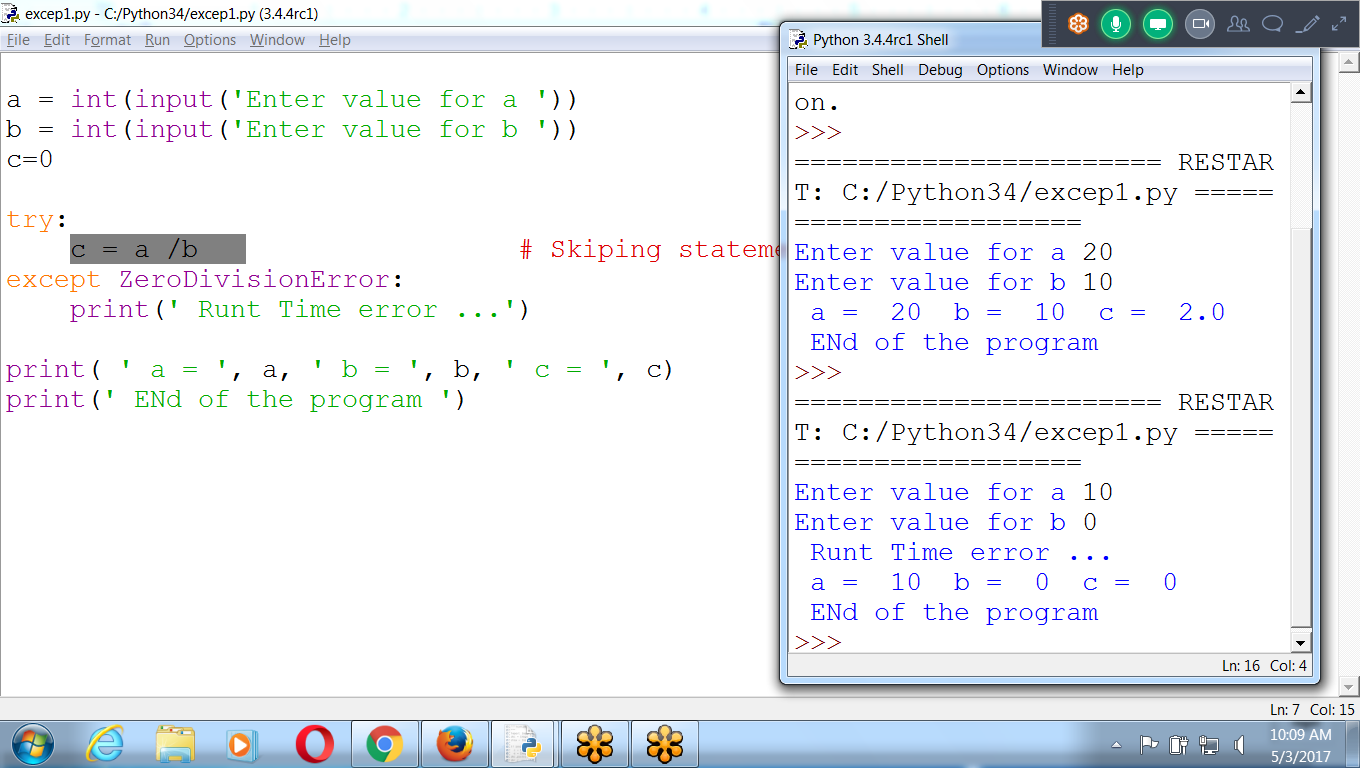
**print(a,b,c)**

**Option 1: Skip the statement, continue Remaining Statements**

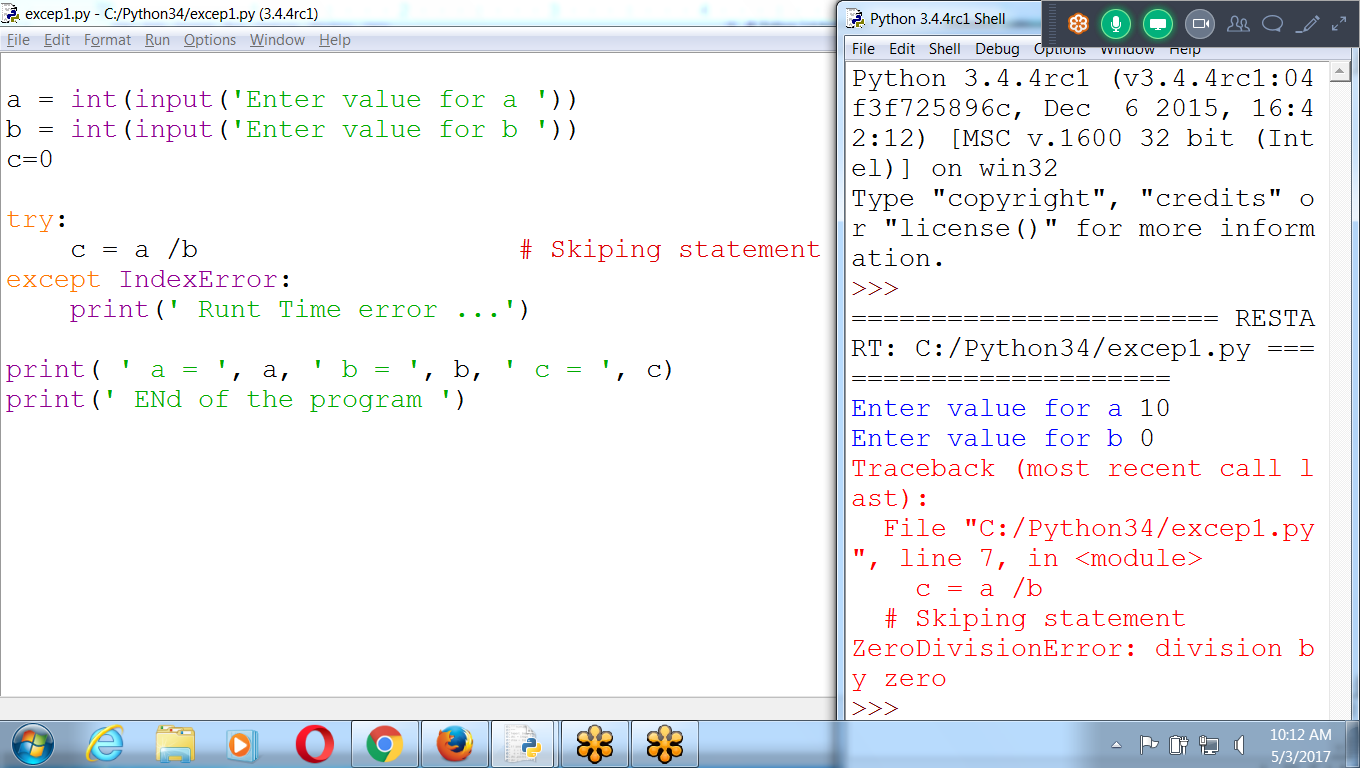
**If any Runtime error encounters, searches corresponding except block,**

**If found executes “EXCEPT” block and continues remaining statements**

**If not found abnormal termination**

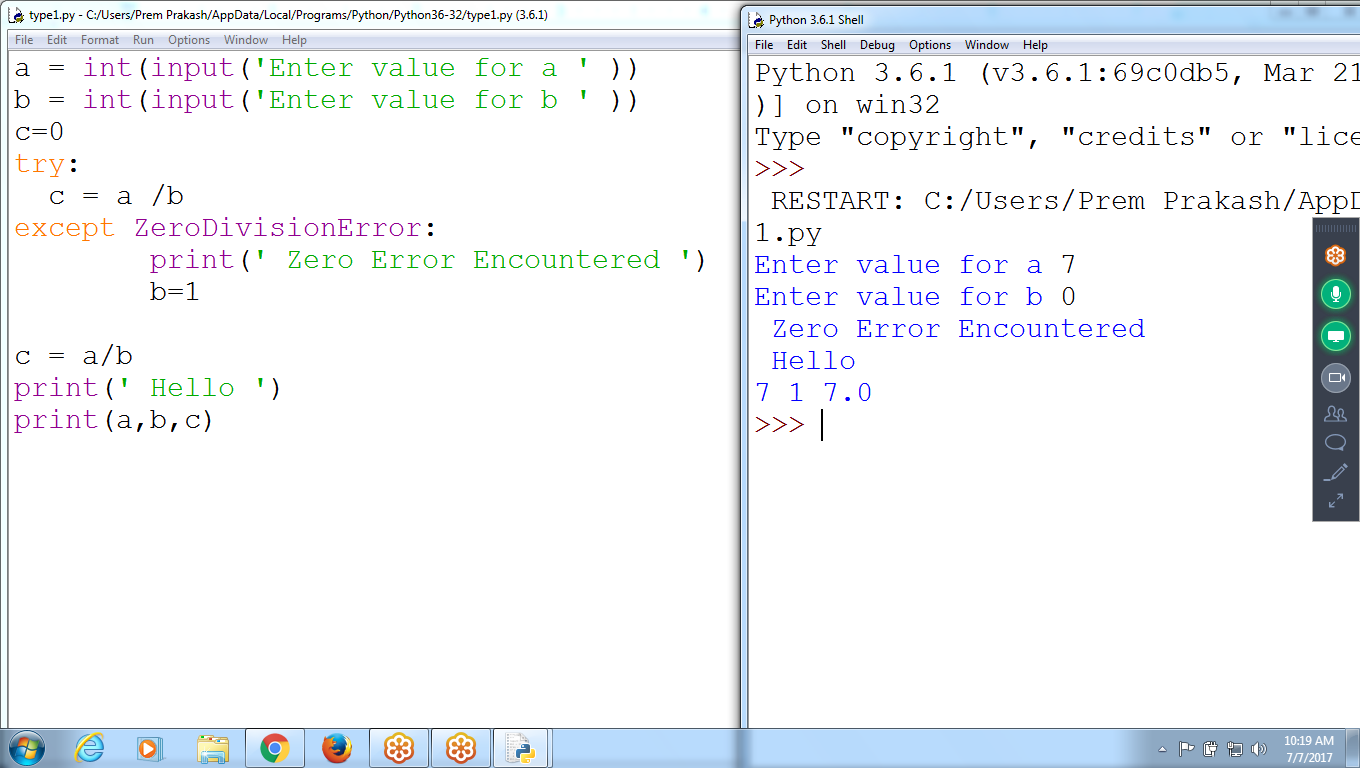
****

If Corresponding ‘except” not existing :: Abnormal Termination



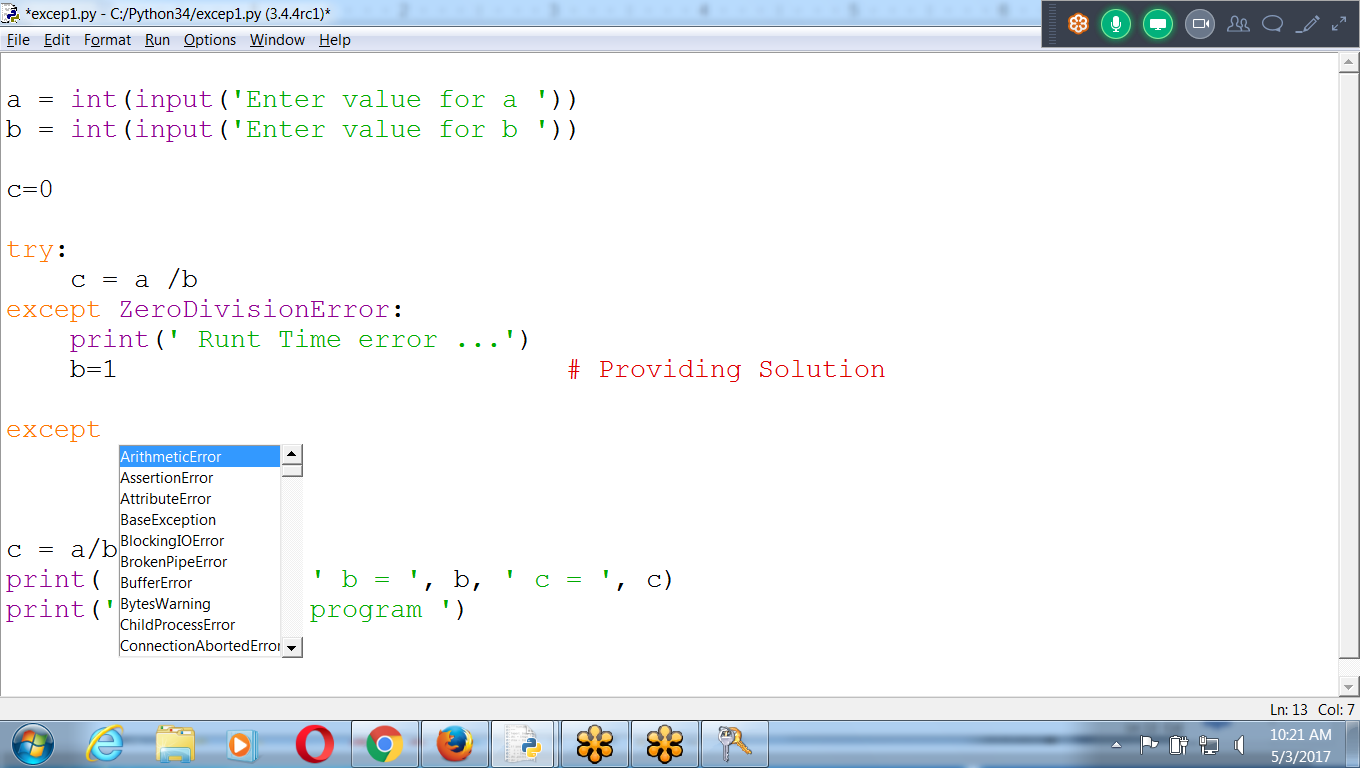
Option 2: Provide Solution

By assigning new value 1 to b

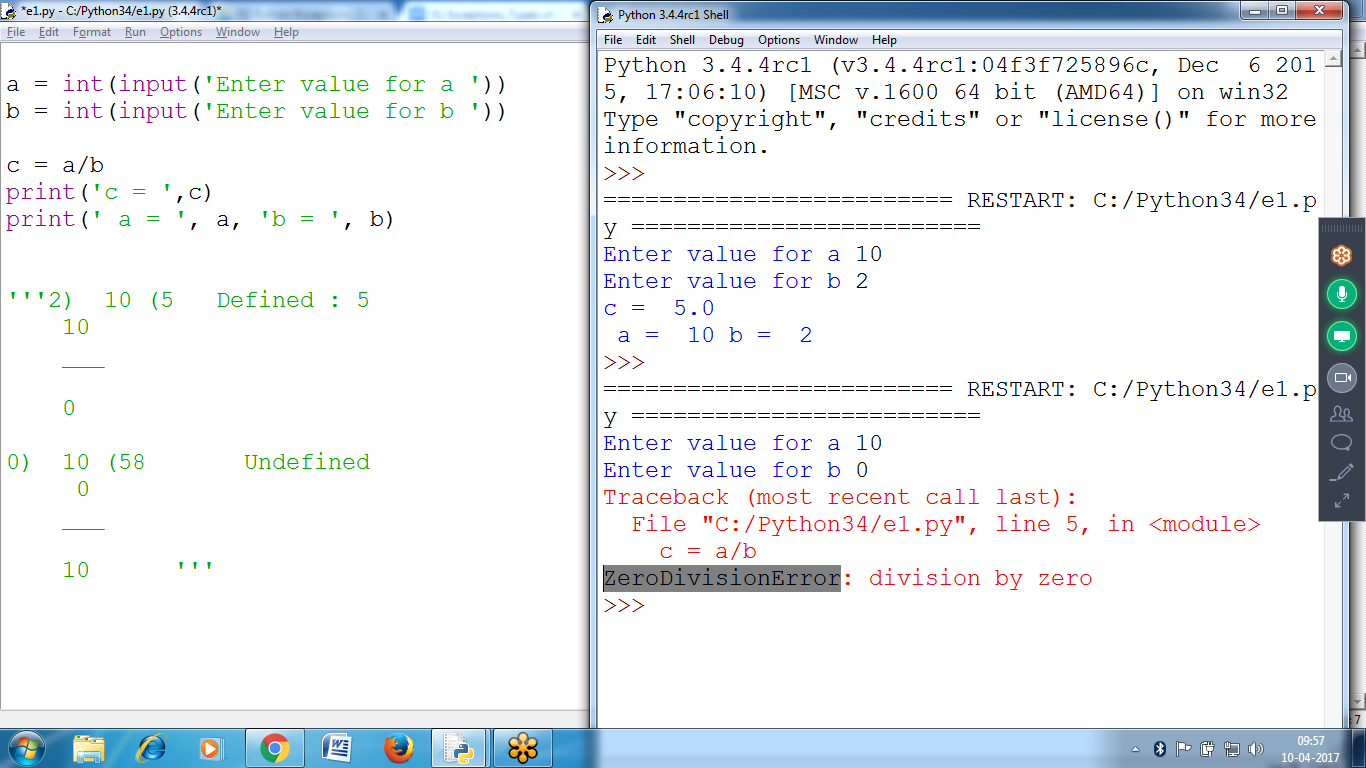


To Find List of RUN TIME Errors

Control + space



Zero Division Runtime Error



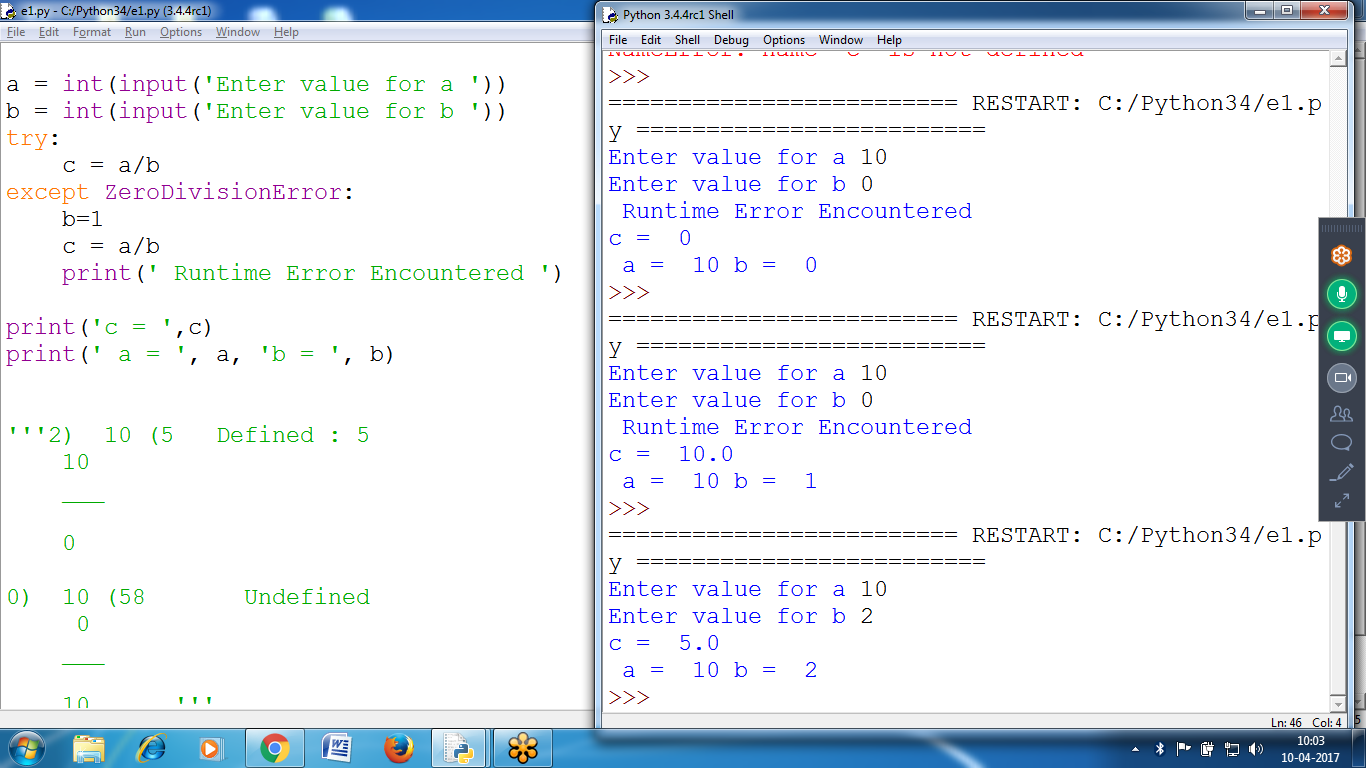
ZeroDivisionError: division by zero

ZeroDivisionError Means ::: Runtime Error Name

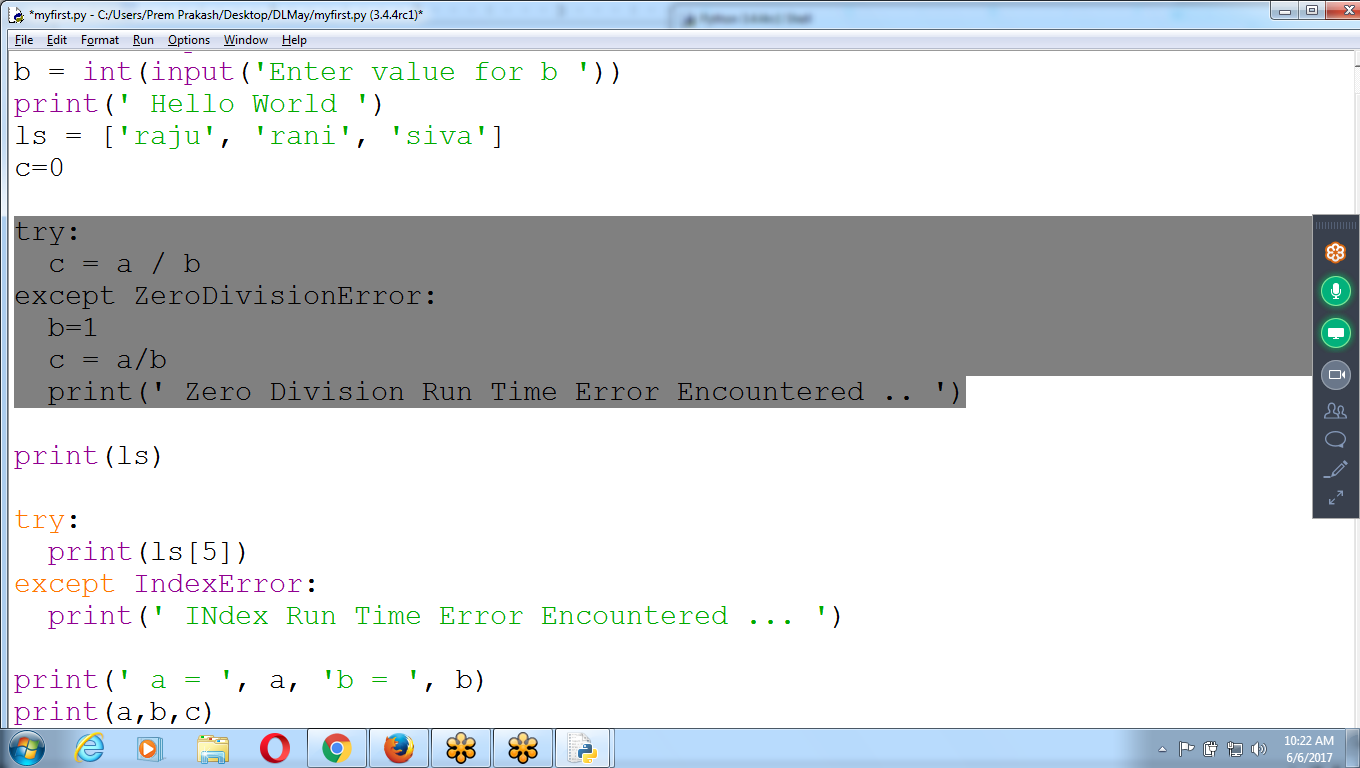
Division by Zero :: Runtime Description

If  **NO Runtime Error**, control moves after except block, Normal Termination

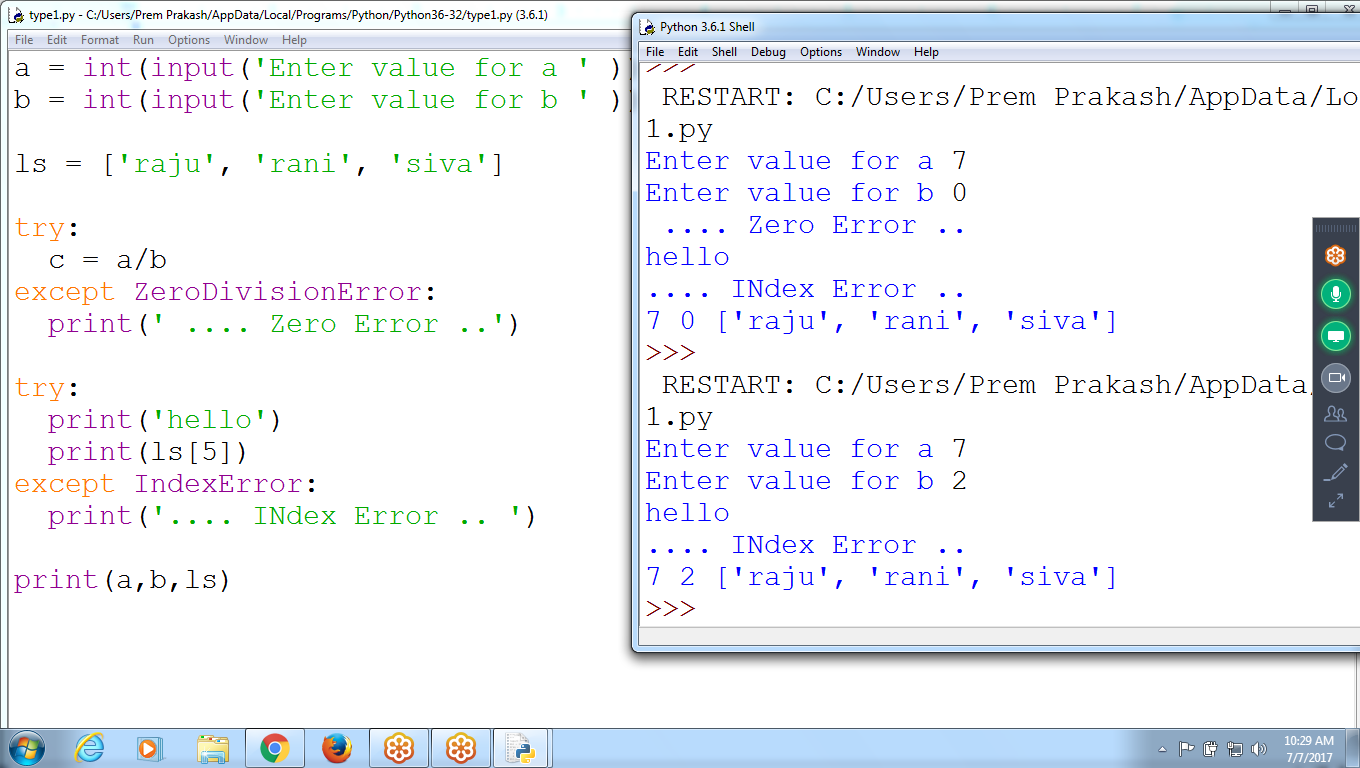
If Runtime Error, checks for corresponding except block



Successfully Termination

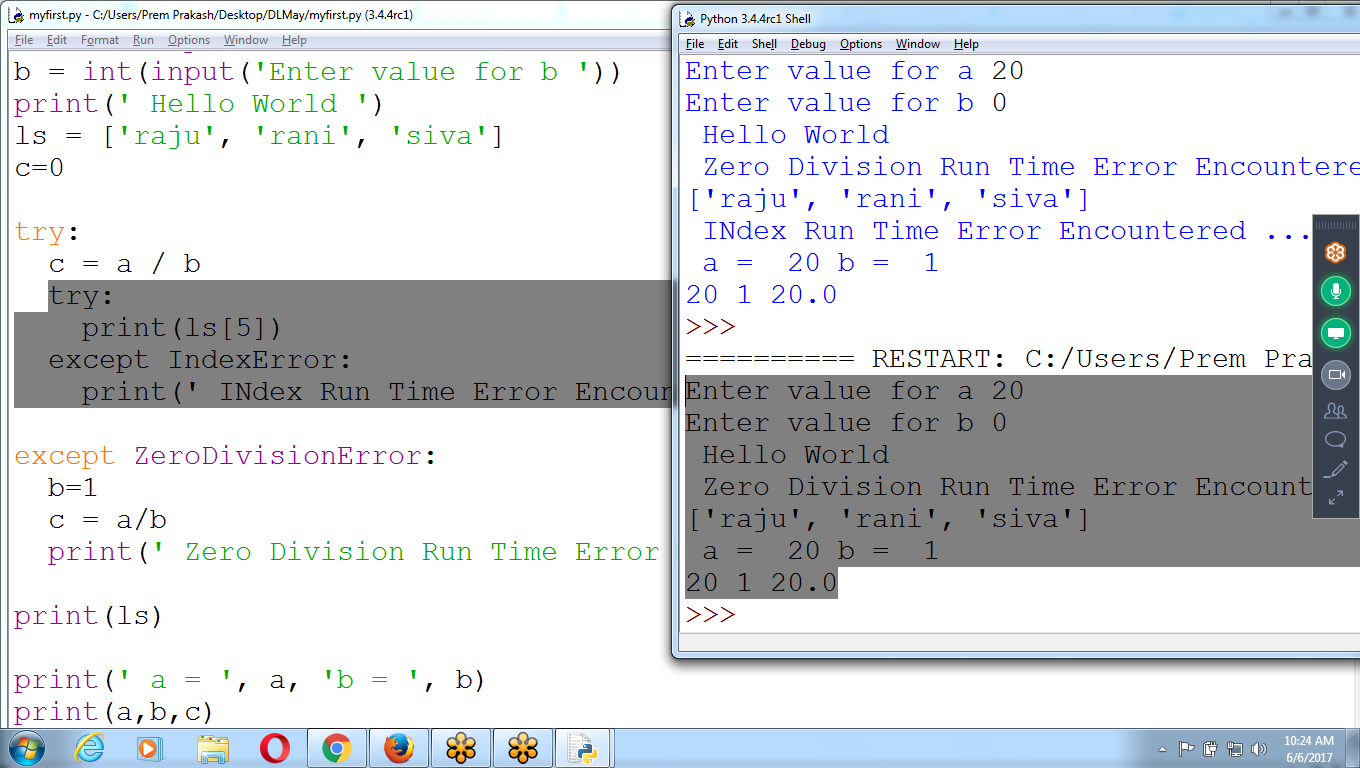


Place all Runtime Errors in separate Try BLock



Inner Try Blocks

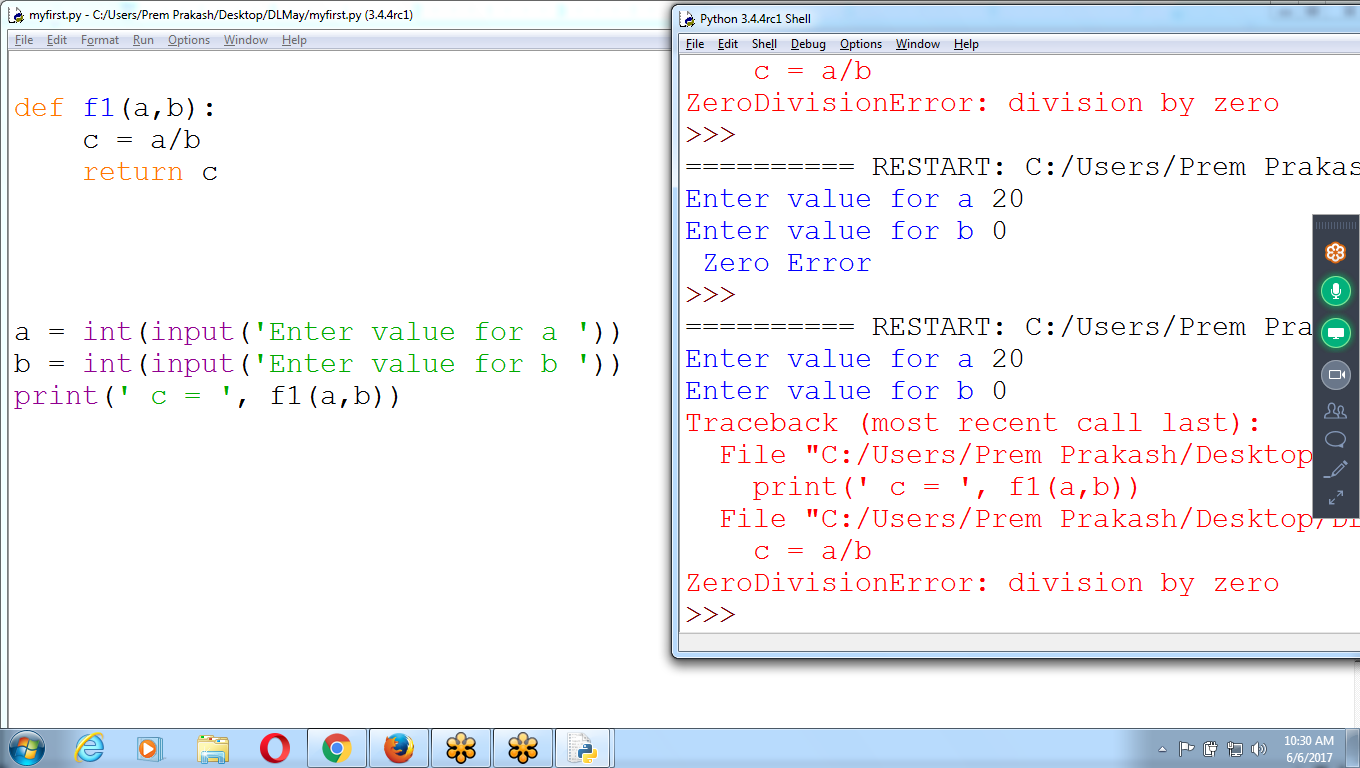
In Example : INner Try is not Executing



Using Functions without Exceptions

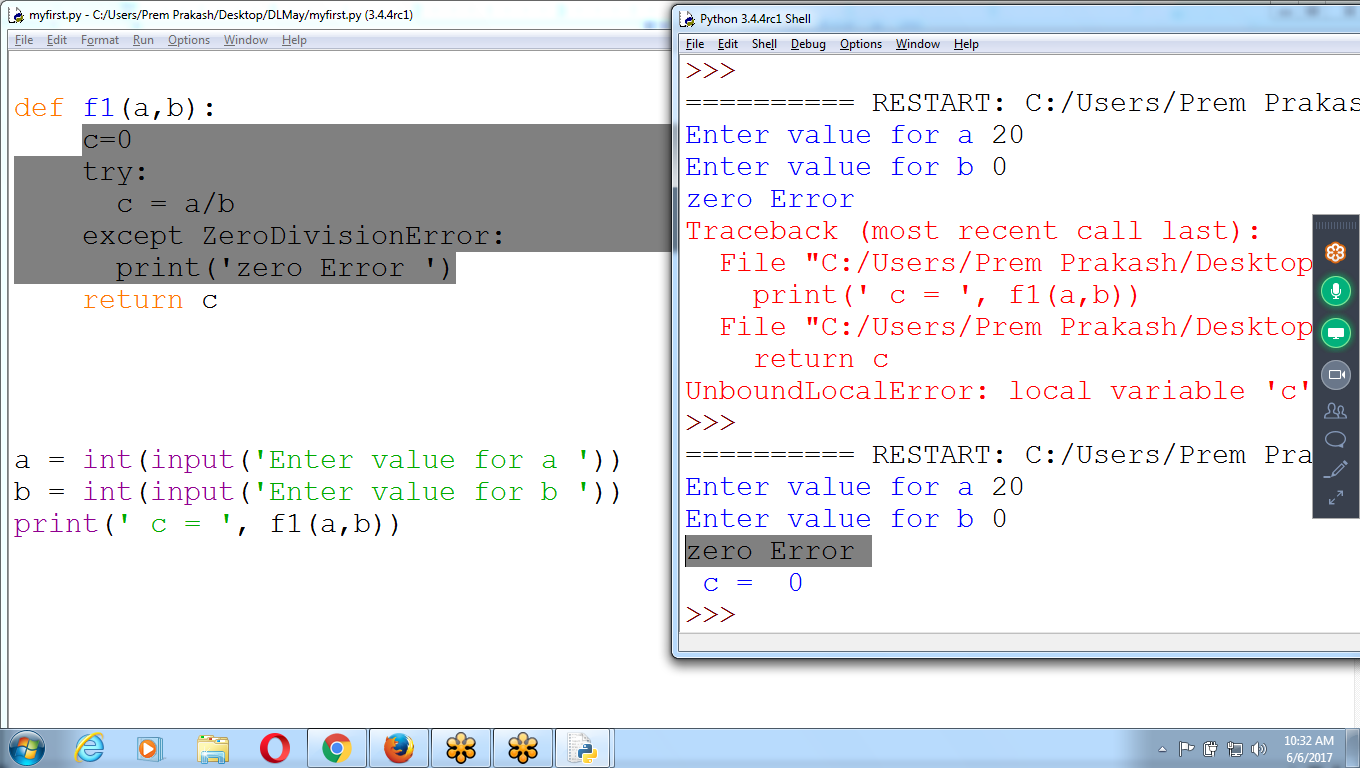


Inner Trace Back

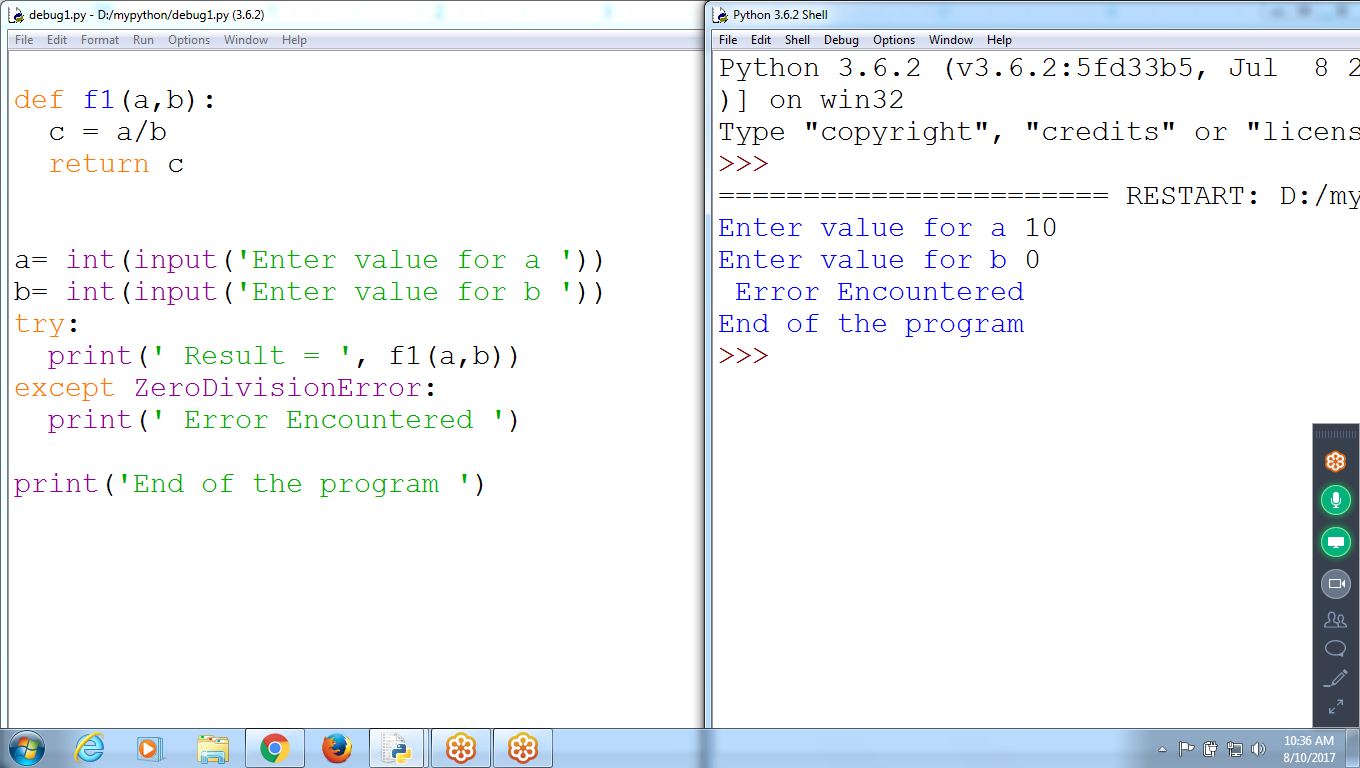


Place Try - Exception inside the function Definition

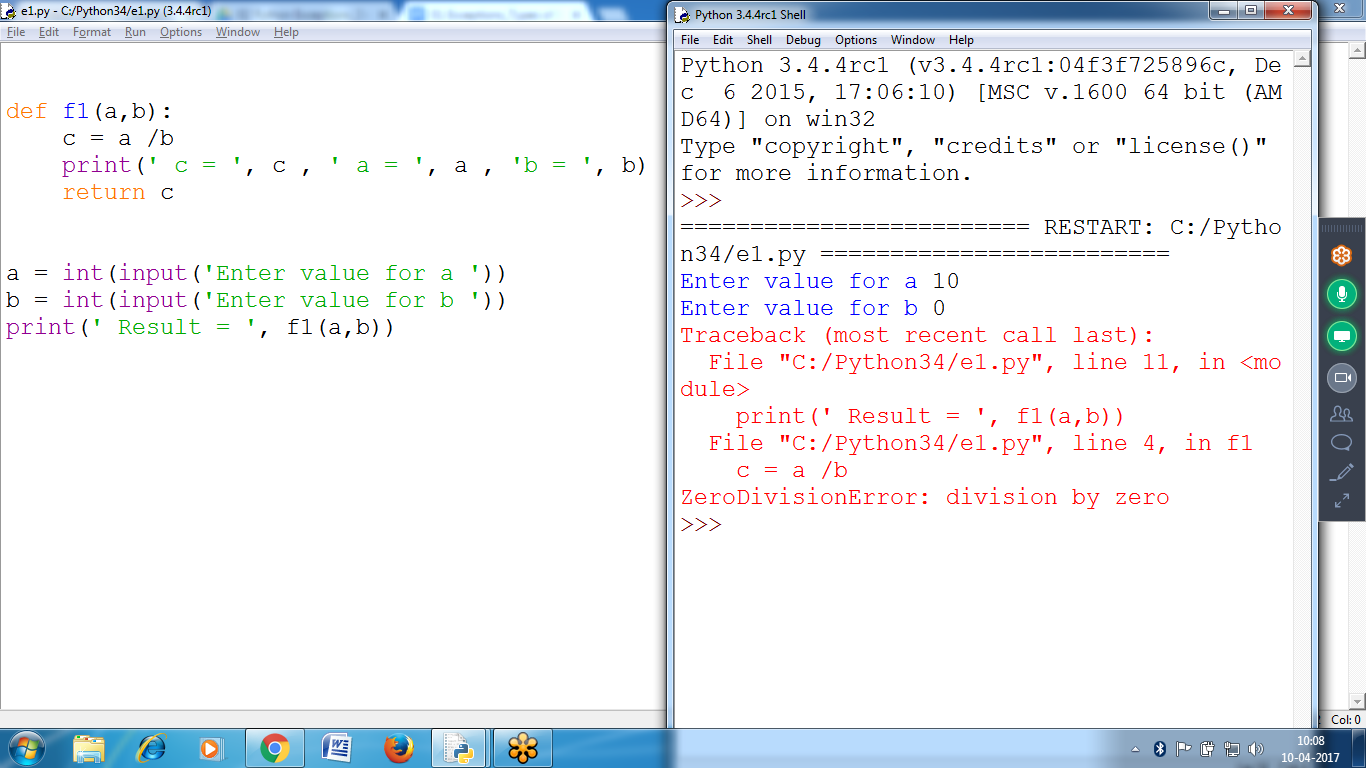
Don’t place in Function calling



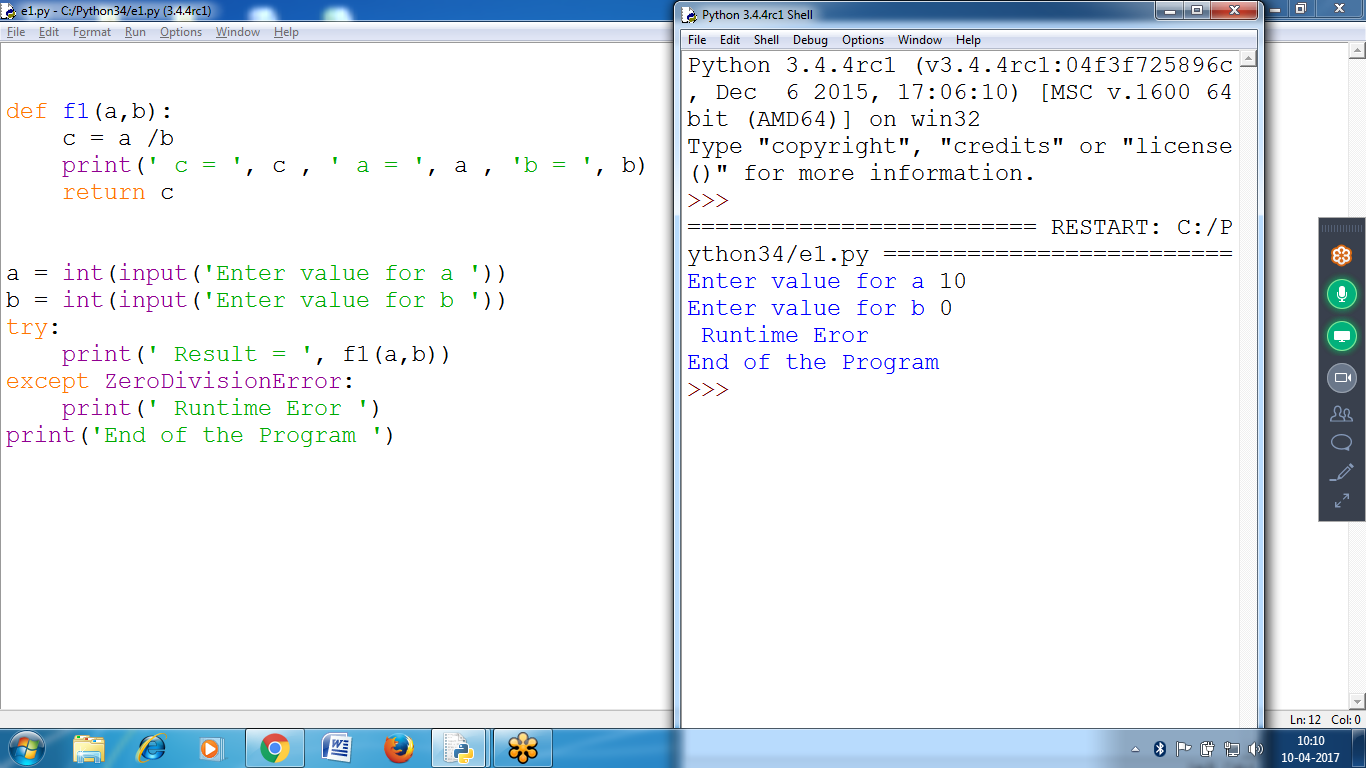
Place Try Block for Print Statement, at the time function calling



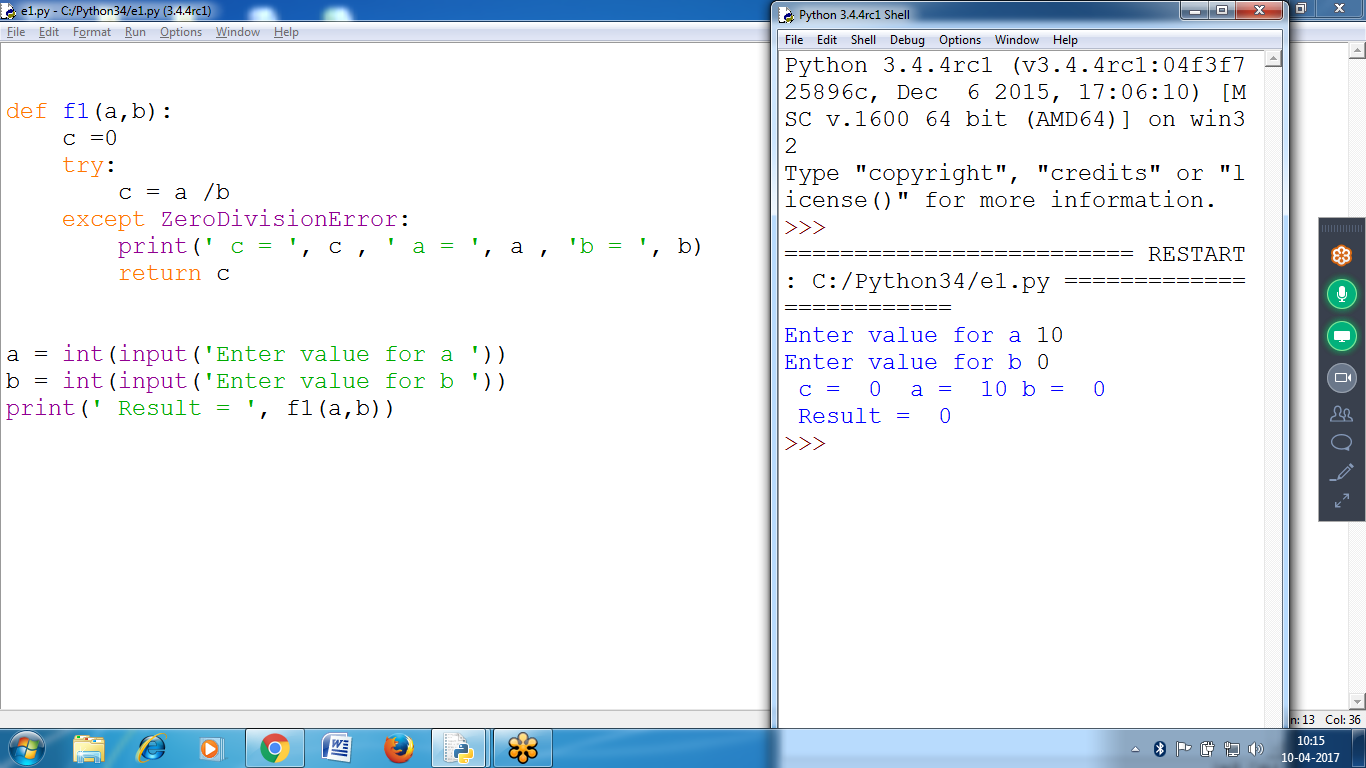
Using Functions and Exceptions



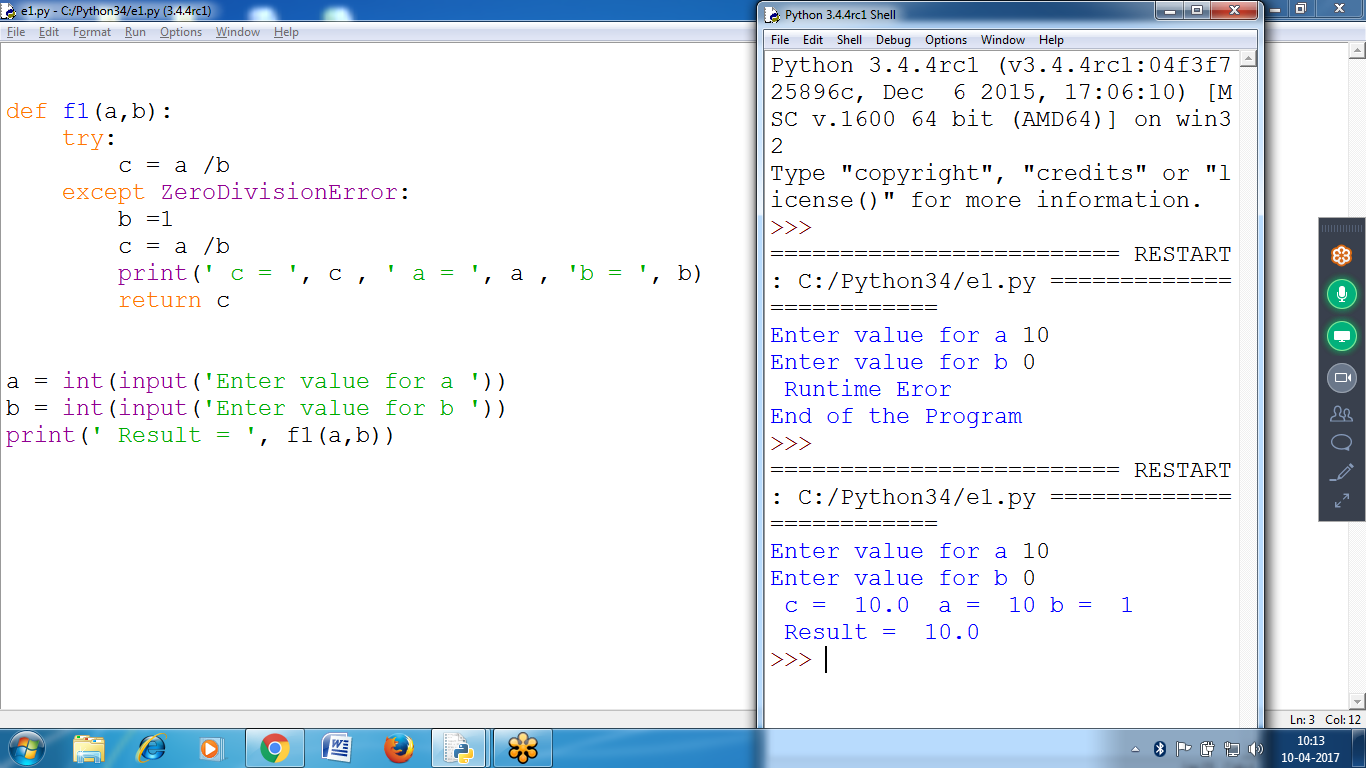
Try block, within the Print statement for calling function



Try block, within the Function



(or)



A ZeroDivisionError happens whenever you try to divide a number by zero

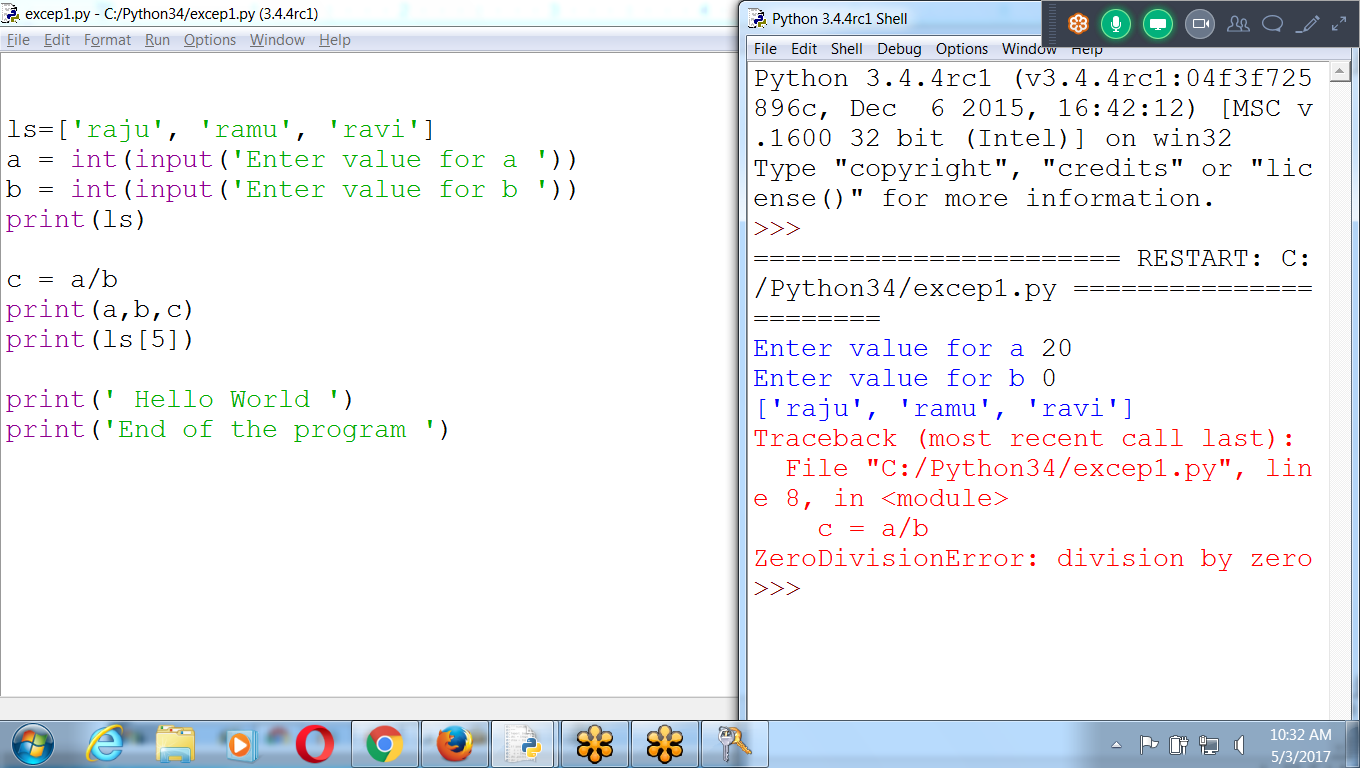
* Errors can be handled with try and except statements.
* The code that could have an error is put in a try clau

place **divide-by-zero code** in a **try clause** and except clause contain code to handle what happens when **error occurs**.

# 

# 

Multiple Runtime Errors

****

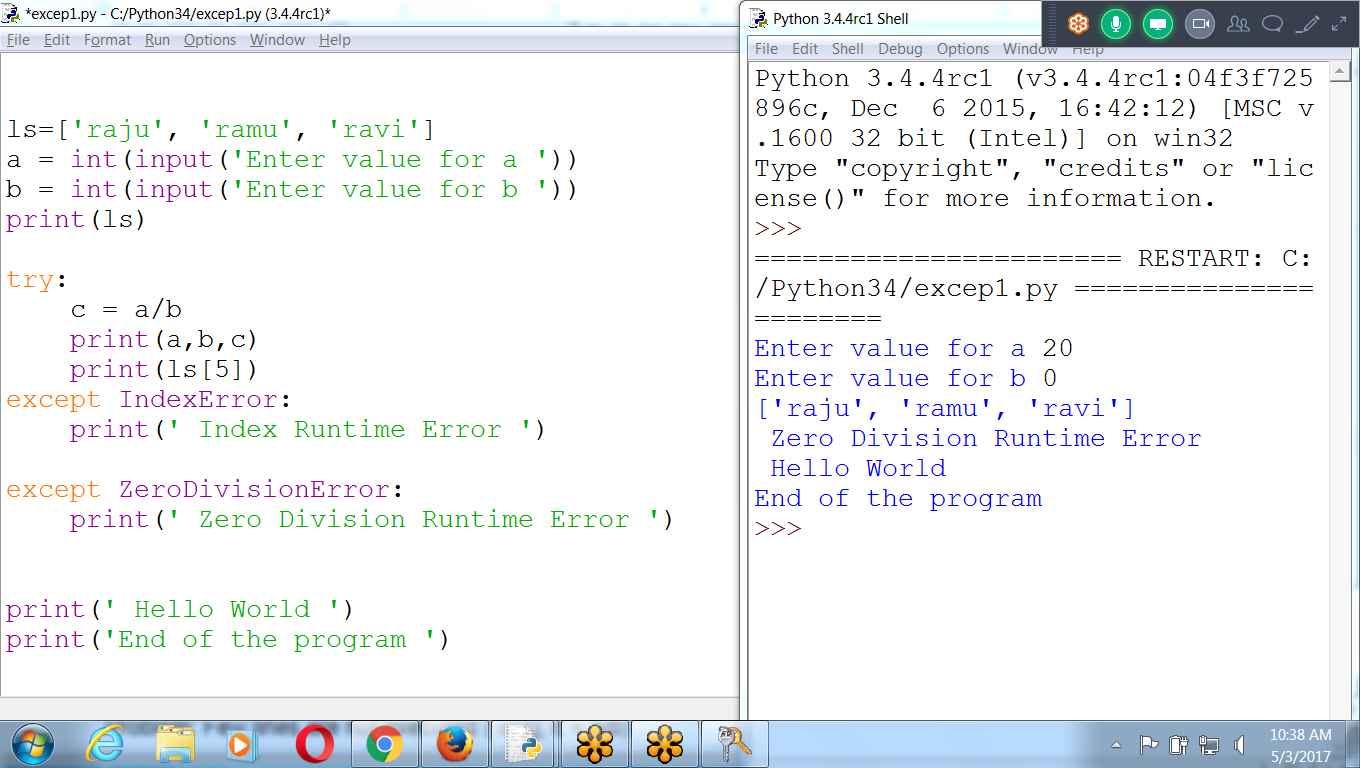
Try block must contain statements which arise Runtime Error

**Few statements are not Executing** (between Try Error statement and except block)

**Program contains Multiple Exception**

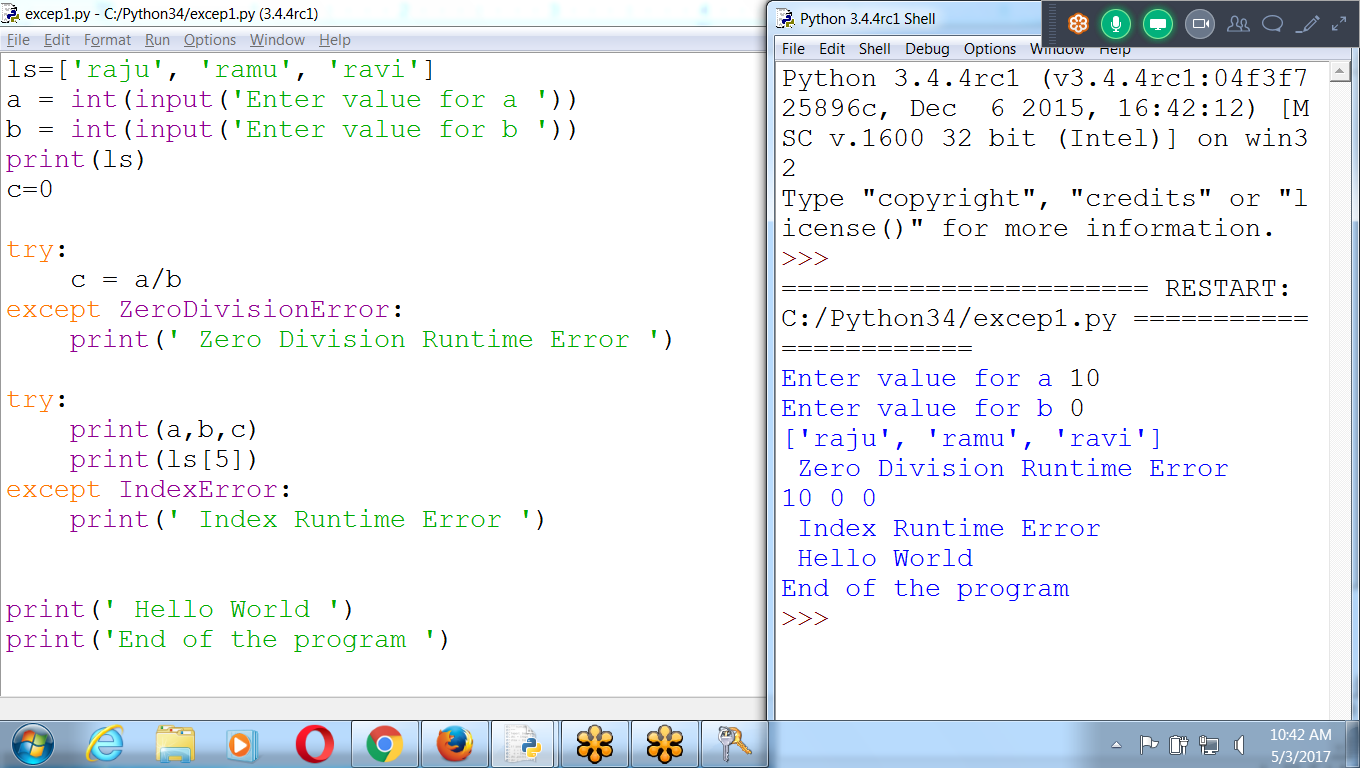
**First checks for Index error, Zero division Error**

**But, few lines are not executed**



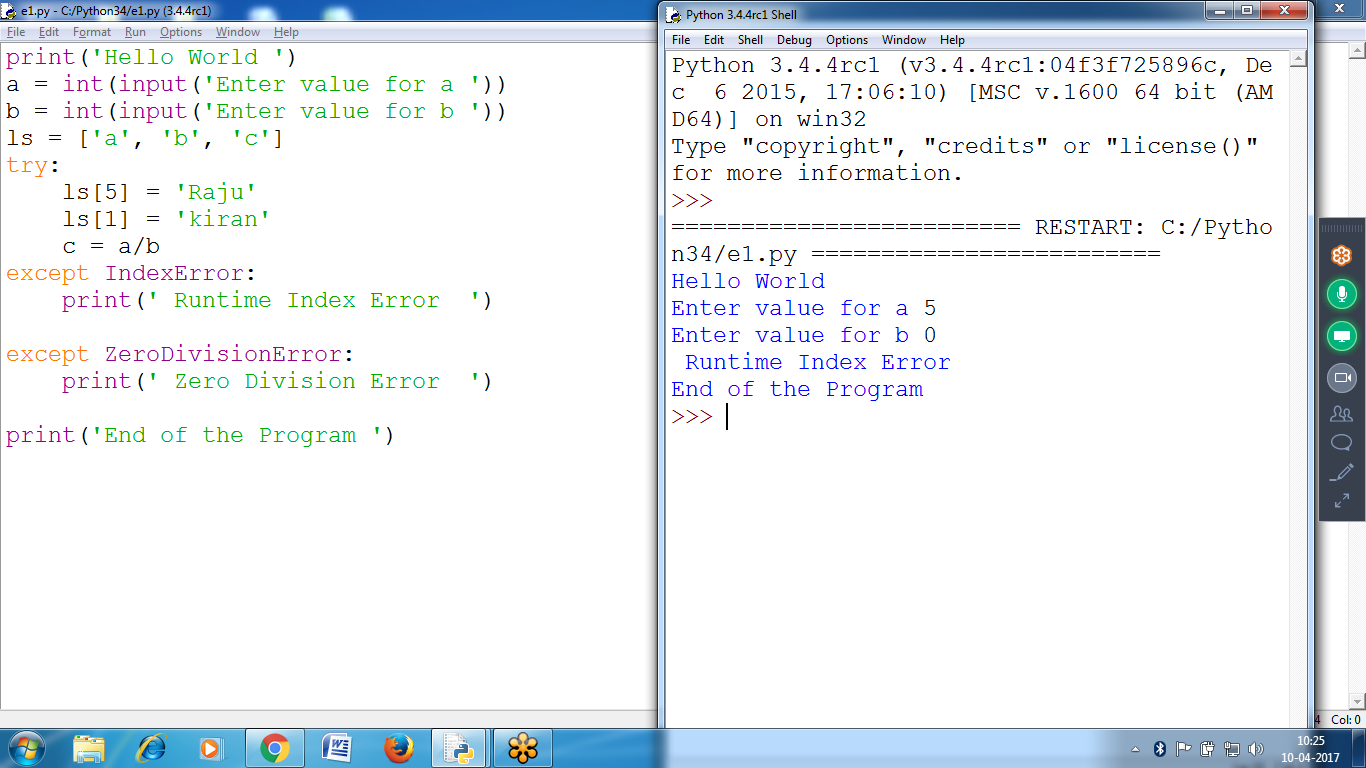
Replace the program with Multiple TRy’s

**Result: All Runtime errors, All statements executed, NORMAL Termination**



Displaying only one Exception:: Single Type and Multiple Exceptions

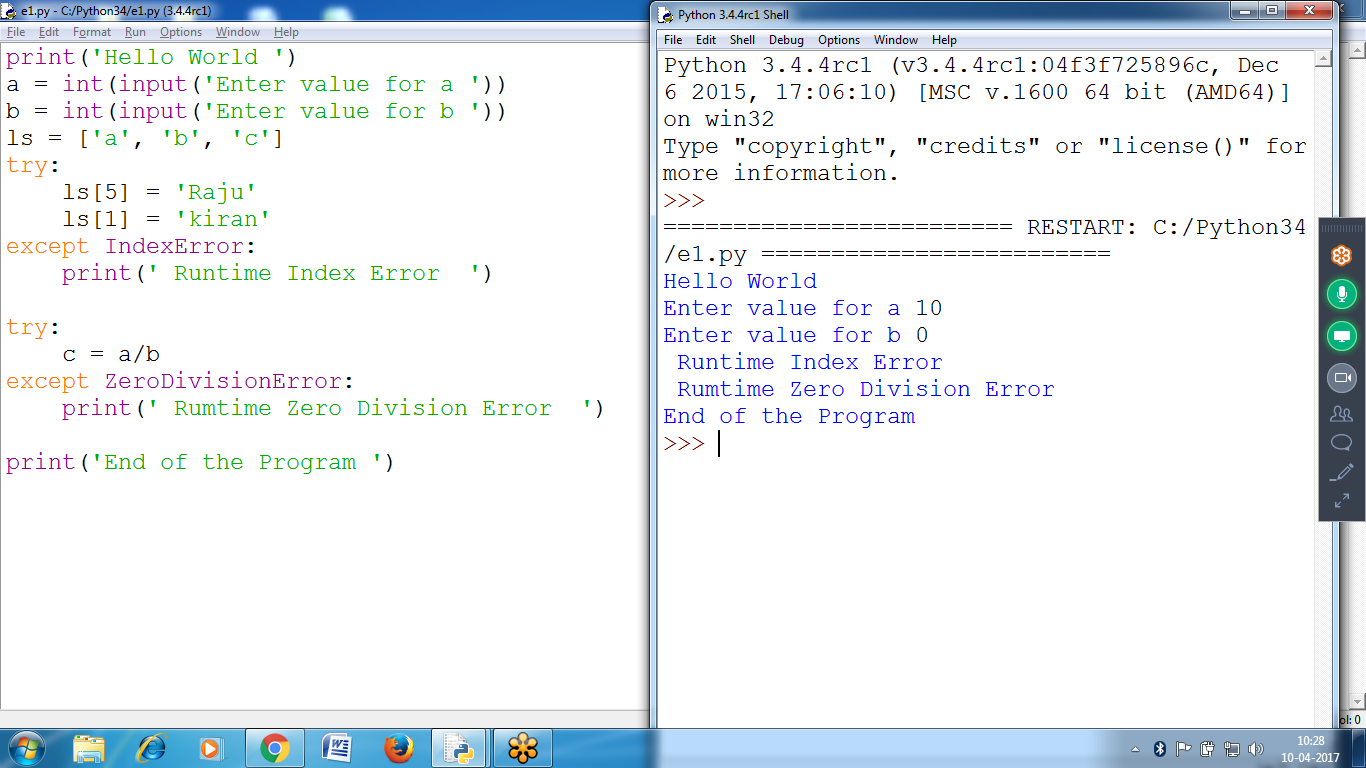
Problem: Few lines are not executed ( ls[1], c = a/b)



How to Handle Multiple Exceptions:

Each Runtime Error Must be Placed in **separate try block**

**Problem: Few lines are missing “ ls[1]”**



print('Hello World ')

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

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

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

try:

ls[5] = 'Raju'

ls[1] = 'kiran'

except IndexError:

print(' Runtime Index Error ')

try:

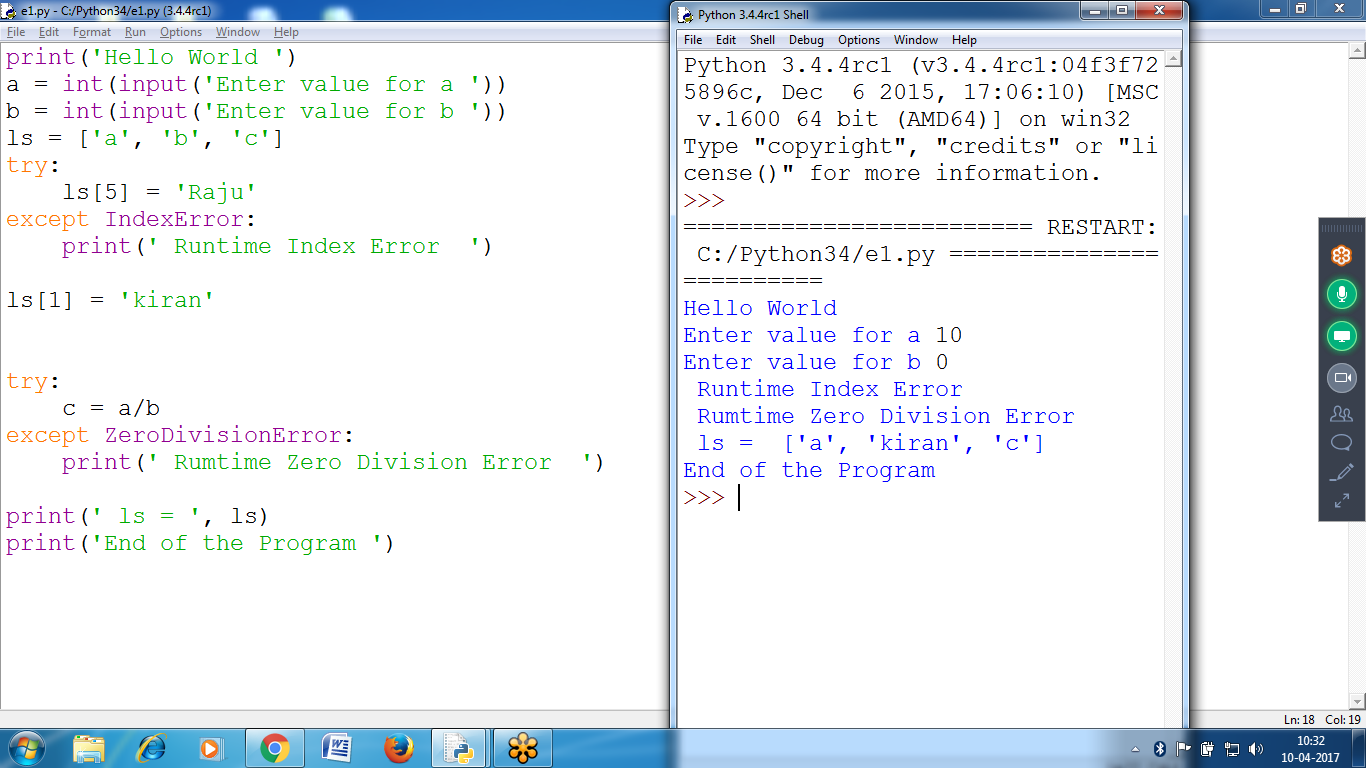
c = a/b

except ZeroDivisionError:

print(' Rumtime Zero Division Error ')

print('End of the Program ')

Place ls[1] as separate statement, outside TRY Block



print('Hello World ')

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

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

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

try:

ls[5] = 'Raju'

except IndexError:

print(' Runtime Index Error ')

**ls[1] = 'kiran'**

try:

c = a/b

except ZeroDivisionError:

print(' Rumtime Zero Division Error ')

print(' ls = ', ls)

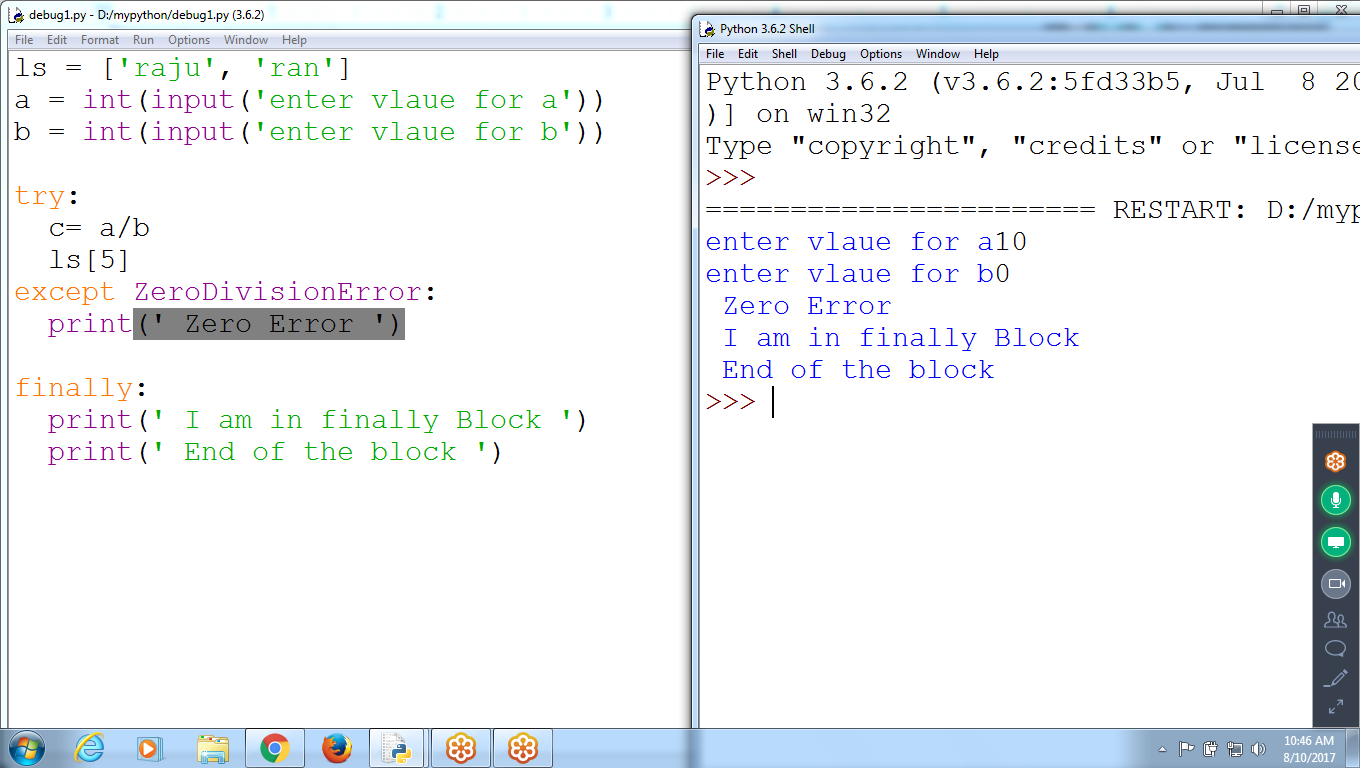
print('End of the Program ')

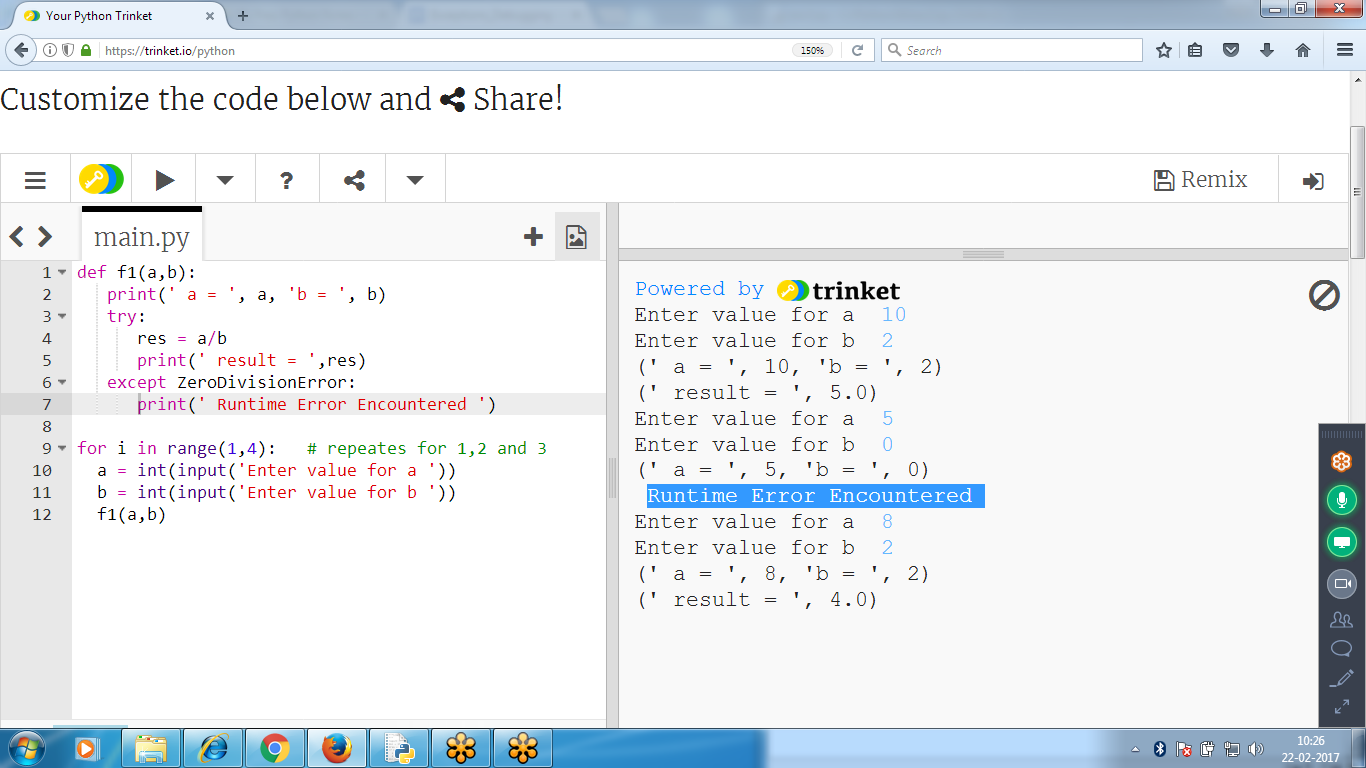
**Finally Block Statements : Compulsory execution block, when runtime error encouters**

Requires: Closing Files for databases



To Write Finally Block, must contain try-except block

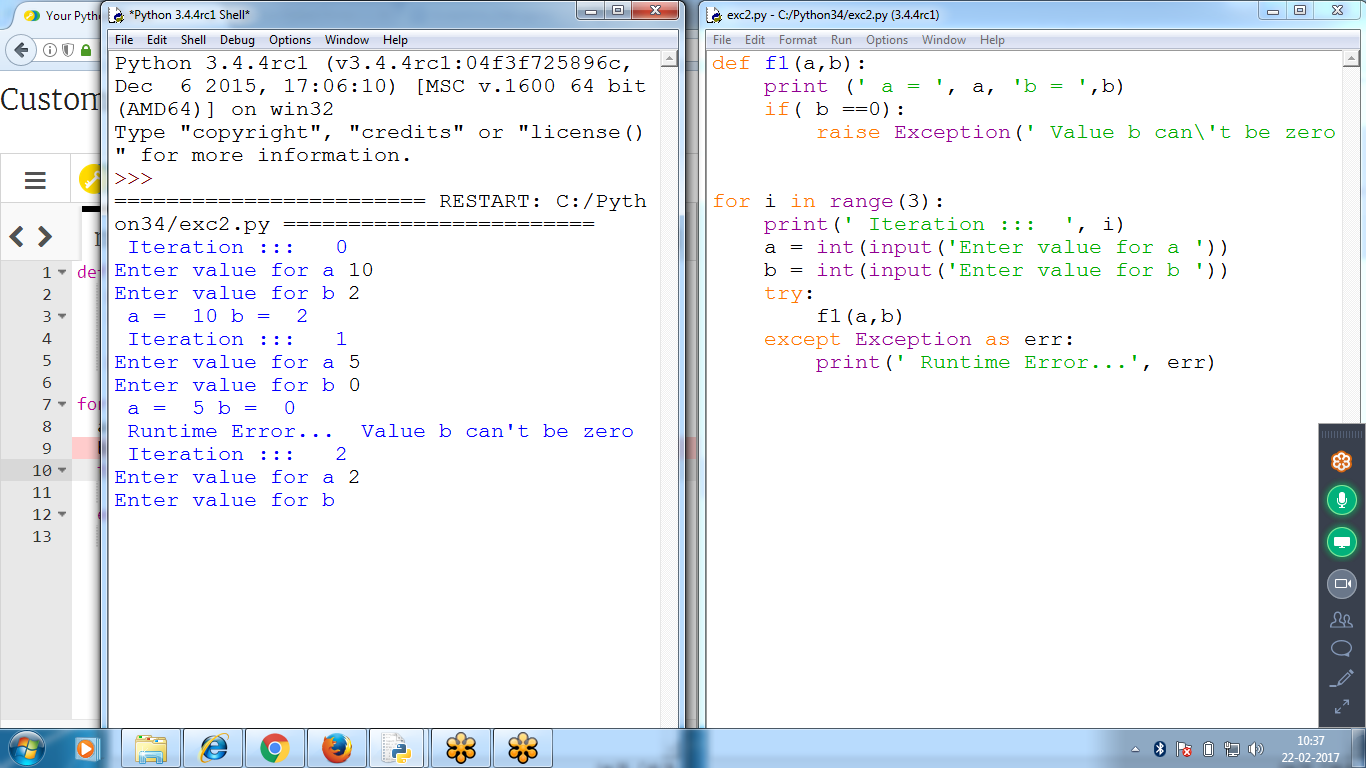


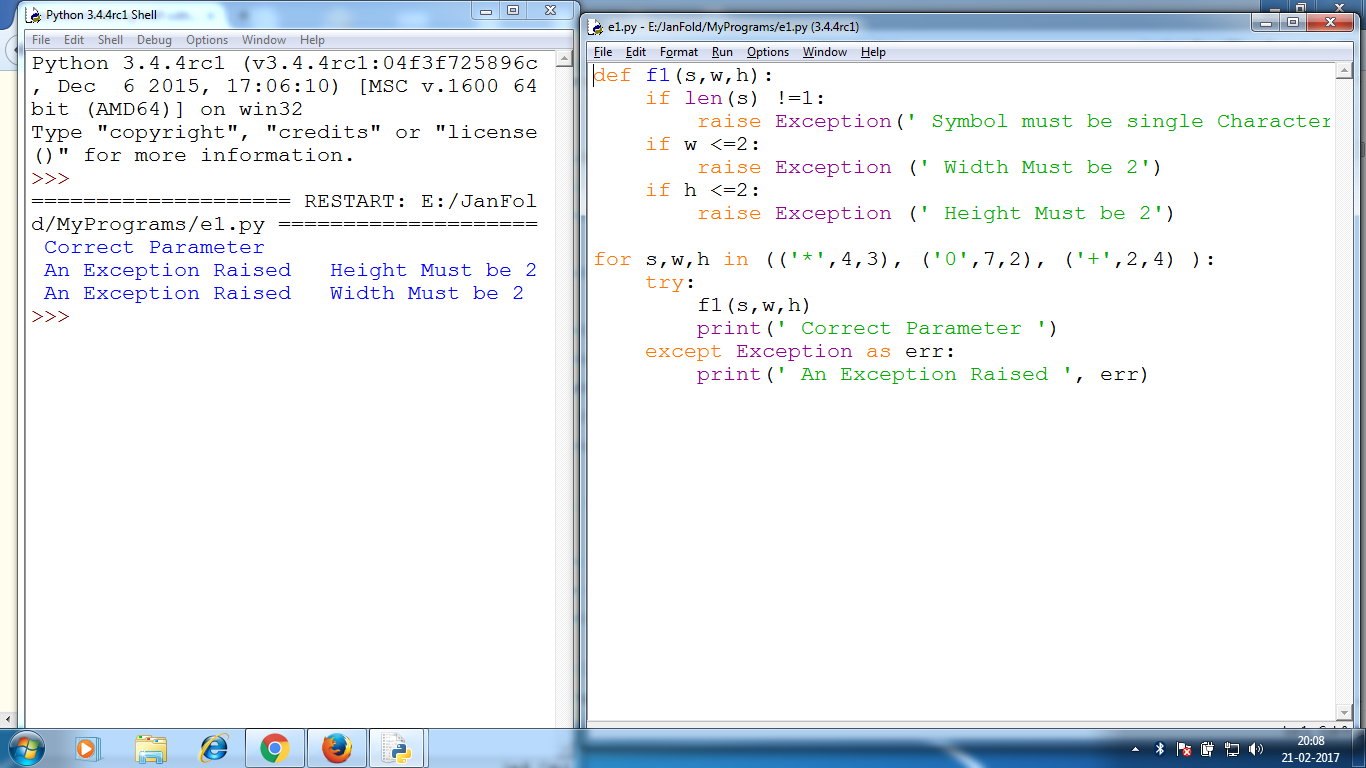
https://trinket.io/python

# **Debugging**

* Tools and Techniques for **finding bugs** in to help in fixing bugs faster and with less effort.
* The **debugger** is a feature of IDLE that executes a program one instruction at a time, giving chance to inspect the values in variables while code runs, and track how the valueschange.
* it is helpful to see the **actual values** in a program while it runs

# **Raising Exceptions**

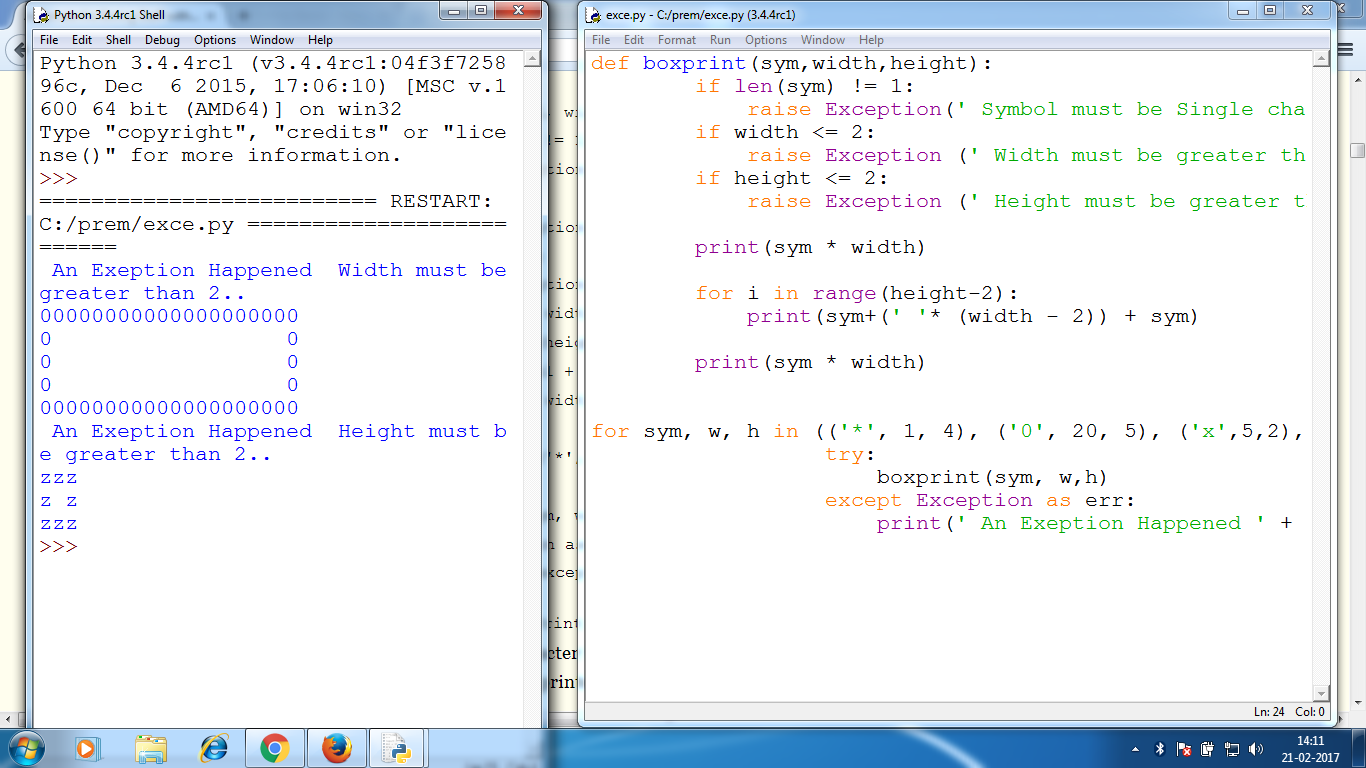




# 

# 

Python raises an exception whenever it tries to execute invalid code



* When Python encounters an error, it produces a error information called the ***traceback*.**
* The traceback includes the error message, the line number of the line that caused the error, and the sequence of the function calls that led to the error. This sequence of calls is called the ***call stack*.**