Exception Handling

Introduction

- Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution.

Exception keyword in Python

- We are handling 3 keywords in python
  - Try → which contains main block of code which possibly causes error
  - Except → keyword which going to execute while Error
  - Finally → if exception happens or not it would execute

Standard Exception

Exception	Cause of Error
AssertionError	Raised when assert statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the input() functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when system operation causes system related error.
OverflowError	Raised when result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.

Standard Exception

StopIteration	Raised by next() function to indicate that there is no further item to be returned by iterator.
SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.
TabError	Raised when indentation consists of inconsistent tabs and spaces.
SystemError	Raised when interpreter detects internal error.
SystemExit	Raised by sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translating.
ValueError	Raised when a function gets argument of correct type but improper value.
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

Syntax in try…except

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
```

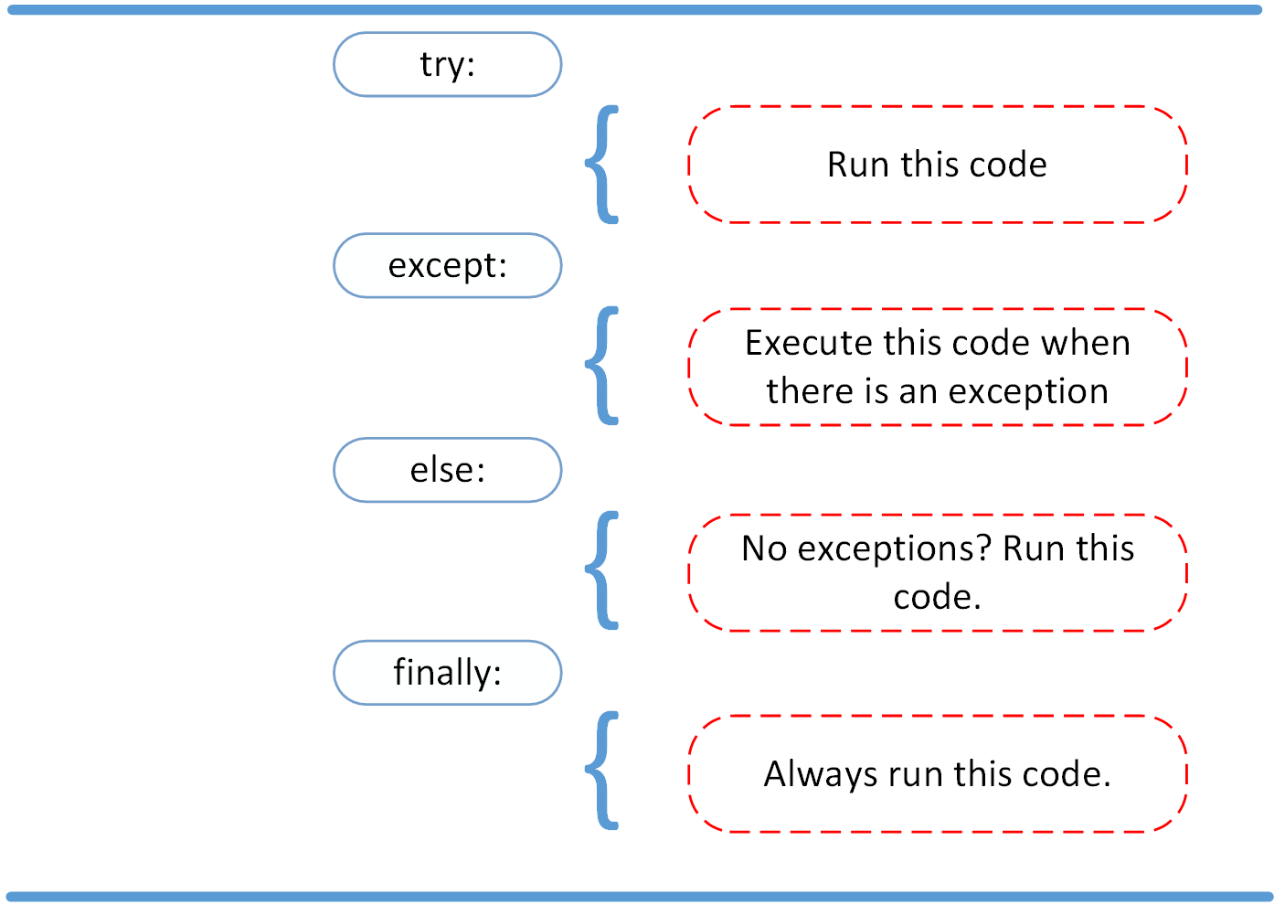
Syntax in try…except…else

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

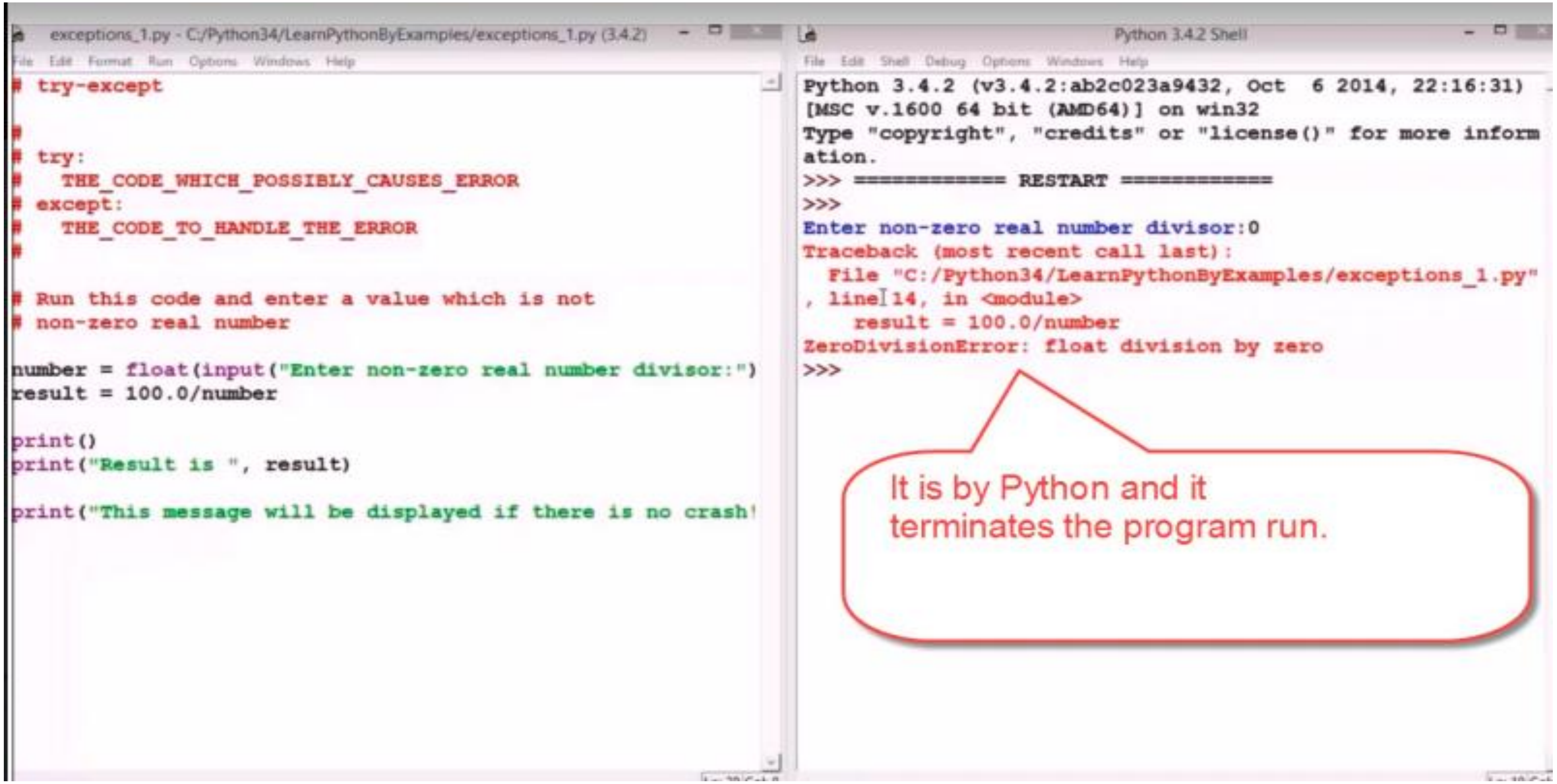
Try…except…

```
try:
    # do something
    pass
except ValueError:
    # handle ValueError exception
    pass
except (TypeError, ZeroDivisionError):
    # handle multiple exceptions
    # TypeError and ZeroDivisionError
    pass
except:
    # handle all other exceptions
    pass
```

Flow Chart


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.

Getting Exception

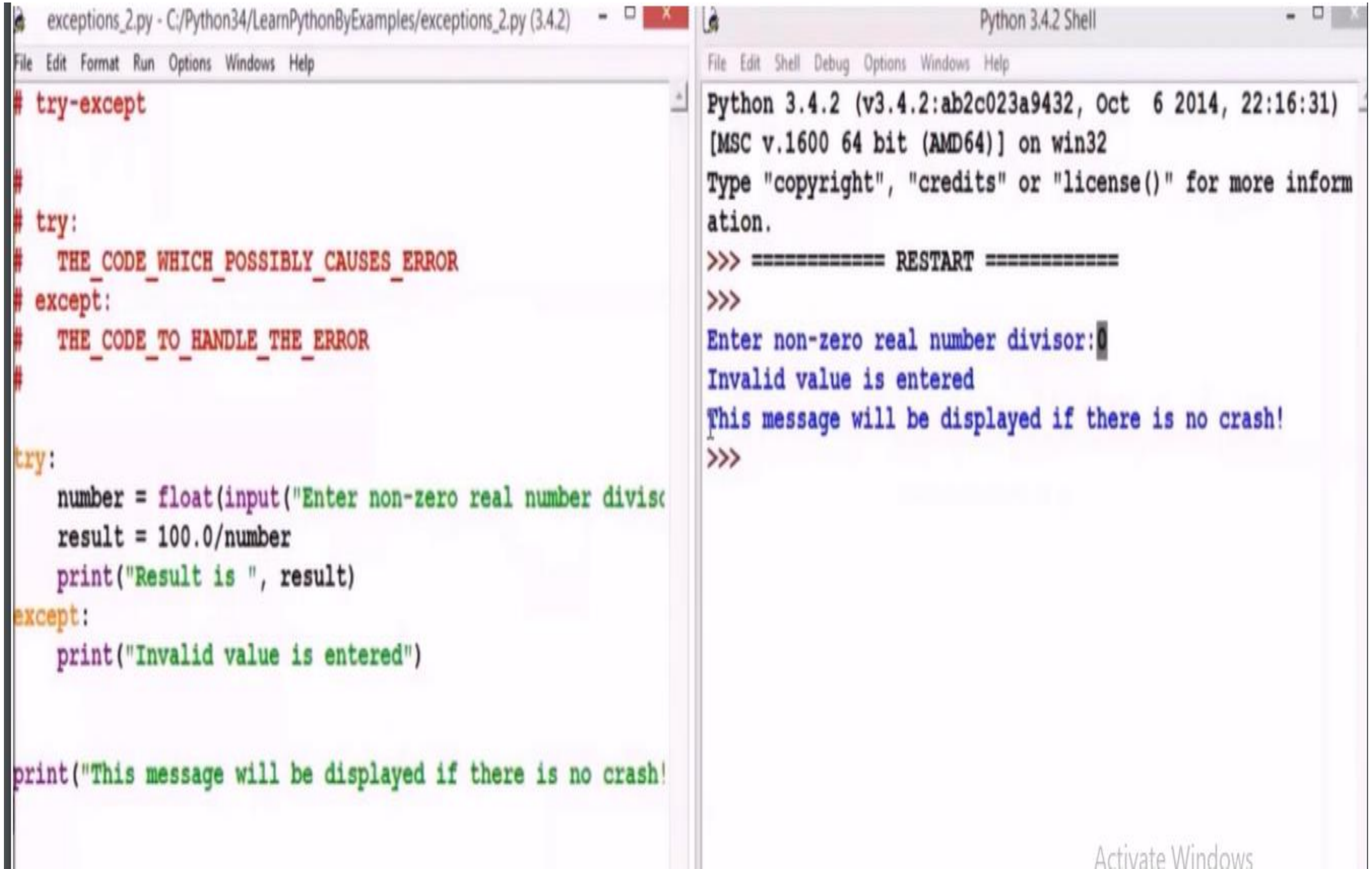


```
# try-except
#
# try:
#   THE_CODE_WHICH_POSSIBLY_CAUSES_ERROR
# except:
#   THE_CODE_TO_HANDLE_THE_ERROR
#

# Run this code and enter a value which is not
# non-zero real number

number = float(input("Enter non-zero real number divisor:"))
result = 100.0/number

print()
print("Result is ", result)

print("This message will be displayed if there is no crash!
```

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31)
[MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> ============ RESTART ============
>>>
Enter non-zero real number divisor:0
Traceback (most recent call last):
  File "C:/Python34/LearnPythonByExamples/exceptions_1.py"
, line 14, in <module>
    result = 100.0/number
ZeroDivisionError: float division by zero
>>>
```

exceptions_1.py - C:/Python34/LearnPythonByExamples/exceptions_1.py (3.4.2)
Python 3.4.2 Shell
It is by Python and it terminates the program run.

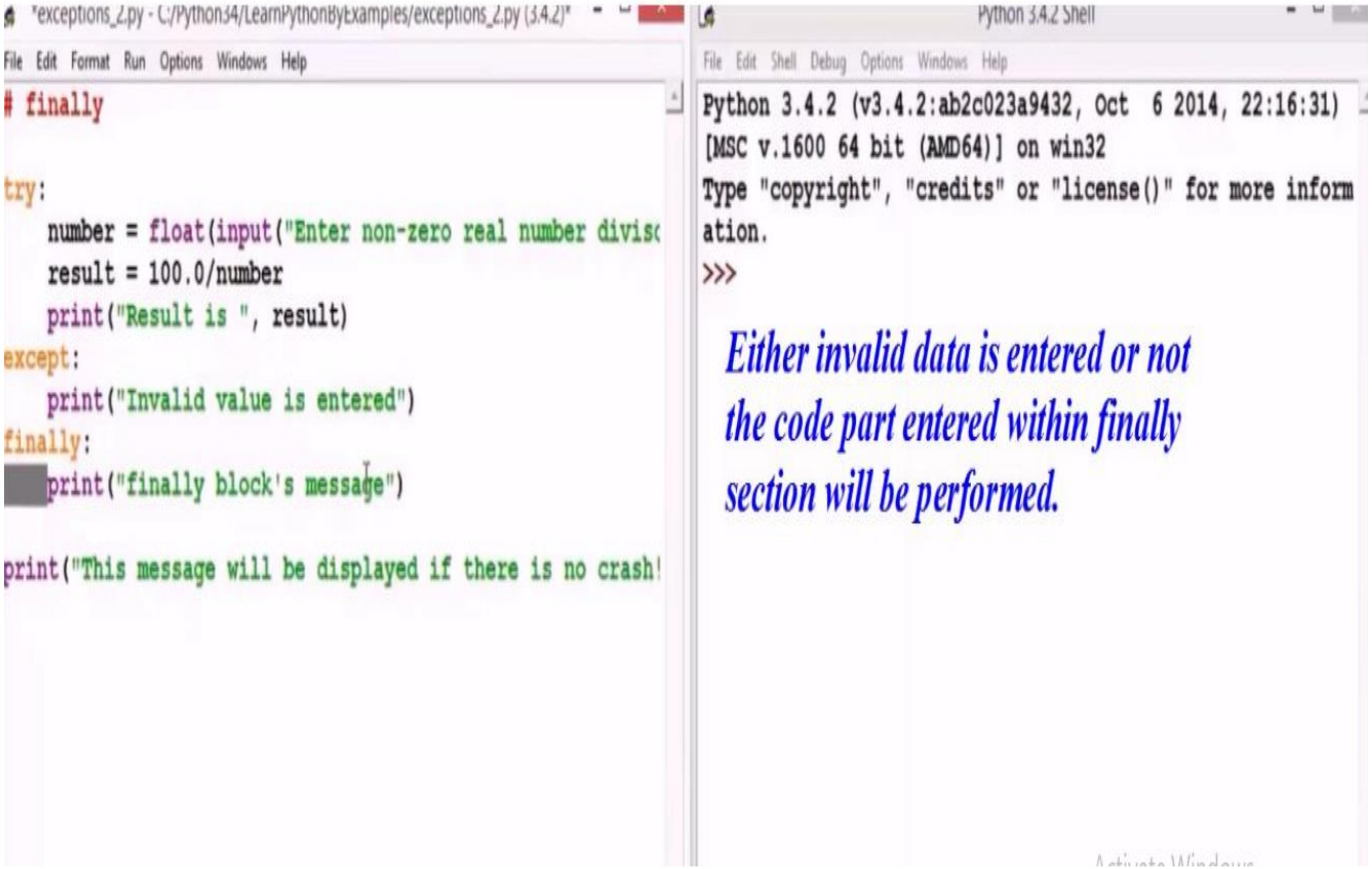
Except Example



```
# try-except

#
# try:
#   THE_CODE_WHICH_POSSIBLY_CAUSES_ERROR
# except:
#   THE_CODE_TO_HANDLE_THE_ERROR
#


try:
    number = float(input("Enter non-zero real number diviso
    result = 100.0/number
    print("Result is ", result)
except:
    print("Invalid value is entered")


print("This message will be displayed if there is no crash!
```

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31)
[MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> ============ RESTART ============
>>>
Enter non-zero real number divisor:0
Invalid value is entered
This message will be displayed if there is no crash!
>>>
```

Finally…example


"exceptions_2.py - C:/Python34/LearnPythonByExamples/exceptions_2.py (3.4.2)"
File Edit Format Run Options Windows Help
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
# finally

try:
    number = float(input("Enter non-zero real number divisc
    result = 100.0/number
    print("Result is ", result)
except:
    print("Invalid value is entered")
finally:
    print("finally block's message")

print("This message will be displayed if there is no crash!
```

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31)
[MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>>
```

Either invalid data is entered or not the code part entered within finally section will be performed.

Raising Exception

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.

- We can also optionally pass in value to the exception to clarify why that exception was raised.

Example

```
>>> try:
...     a = int(input("Enter a positive integer: "))
...     if a <= 0:
...         raise ValueError("That is not a positive number!")
... except ValueError as ve:
...     print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```

User Defined

```
class Error(Exception):
"""Base class for other exceptions"""
   pass
class ValueTooSmallError(Error):
   """Raised when the input value is too small"""
   pass
class ValueTooLargeError(Error):
   """Raised when the input value is too large"""
   pass
number = 10
while True:
   try:
       i_num = int(input("Enter a number: "))
       if i_num < number:
           raise ValueTooSmallError
       elif i_num > number:
           raise ValueTooLargeError
       break
except ValueTooSmallError:
       print("This value is too small, try again!")
       print()
   except ValueTooLargeError:
       print("This value is too large, try again!")
       print()
print("Congratulations! You guessed it correctly.")
```

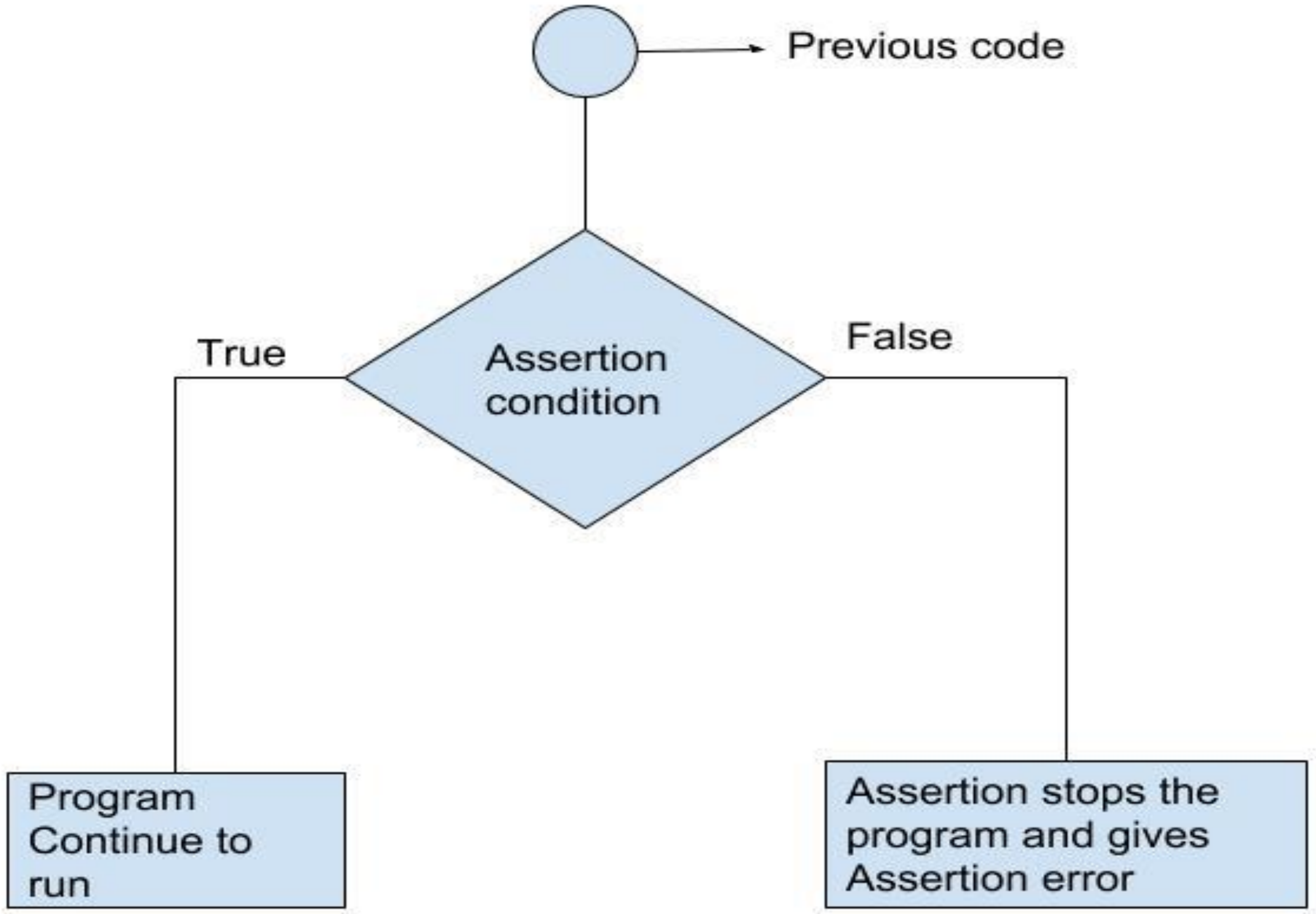
Argument of an Exception

- An exception can have an *argument*, which is a value that gives additional information about the problem.
- try:
- You do your operations here;
- ....
- except ExceptionType, Argument:
- You can print value of Argument here...

Assert

- Assertions are simply Boolean expressions that checks if the conditions return true or not. If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.

Flow chart


Previous code
Assertion condition
True
False
Program Continue to run
Assertion stops the program and gives Assertion error

Assert Statement

- built-in assert statement to use assertion condition in the program. assert statement has a condition or expression which is supposed to be always true. If the condition is false assert halts the program and gives an AssertionError.
  - assert <condition>
  - assert <condition>,<error message>

Assert Example:

- mark1=[1,2,3,4]
- mark2=[]
- assert len(mark2) != 0,"List is empty."

Standard Exception Example

- try:
-     mark1=[1,2,3,4]
-     mark2=[]
-     assert len(mark2) == 0,"List is empty."
- except AssertionError:
-     print("Asser Error:")