Python Programming

EDUCATION FOR PEACE

(ESTD-1995)

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

EXCEPTION HANDLING - 1

Guido Van Rossum

Agenda:

1. Introduction

2. Syntax Error vs Runtime Error

3. Three Important Questions about Exception Handling

1. Introduction

- Exception handling is most valuable concept for any technology. It deals with, if something goes wrong, how you can handle the situation.
- Don't think that everything goes according to our wish. It's impossible, definitely some where, something goes wrong. It is inevitable. If something goes wrong, how can you handle the situation? That concept is called as Exception Handling.
- If everything is going based on your wish, definitely you won't be in this class.

- In general, there may be 'n' number of things may happen which are opposite to our wish. It is very common whether it is in the life or program.
- In our life or program, if we are expected something, at runtime, some unexpected thing may happen. Then, our program should not terminate or our life should not terminate. We have some alternative way to continue. That topic itself is nothing but exception handling.

- Sometimes while executing a Python program, the program does not execute at all or the program executes but generates unexpected output or behaves abnormally.
- These things will occur when there are syntax errors, runtime errors or logical errors in the code.
- In Python, exceptions are errors that get triggered automatically.
- However, exceptions can be forcefully triggered and handled through program code. In this chapter, we will learn about exception handling in Python programs.

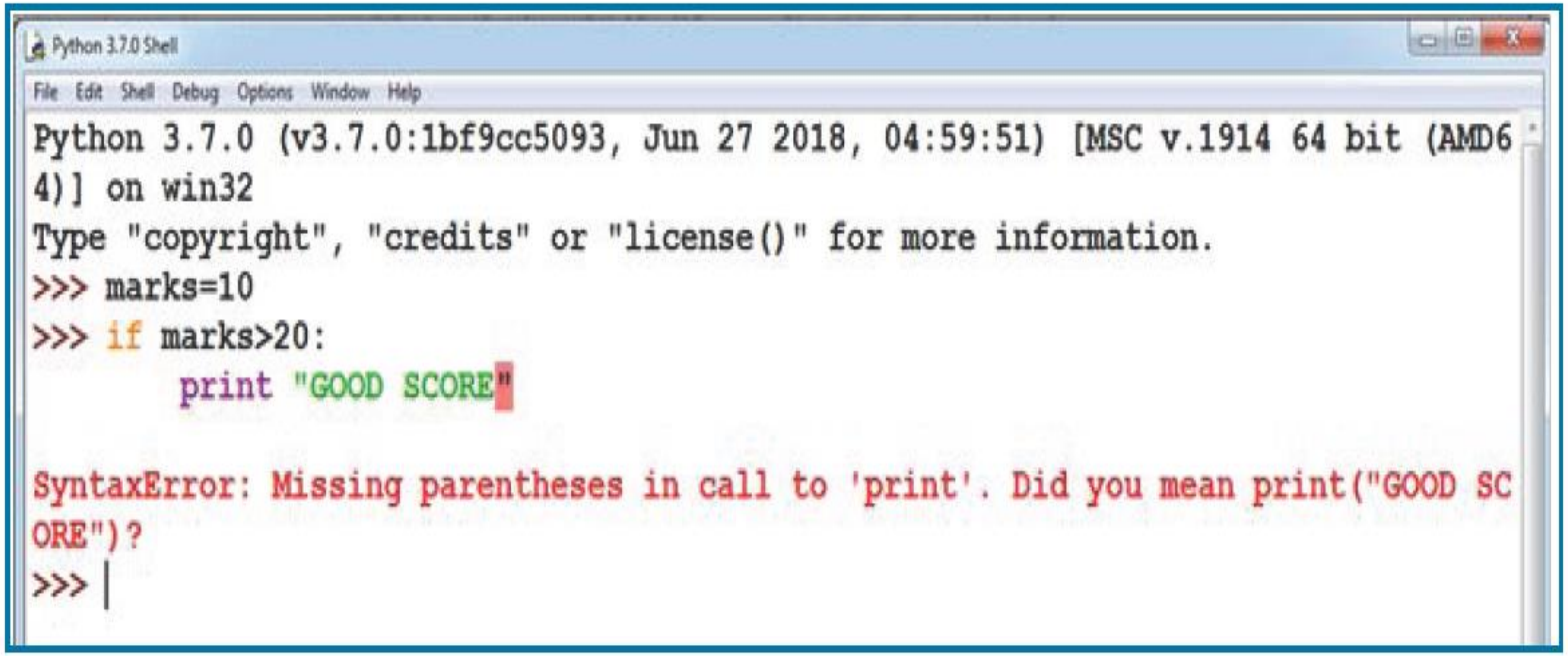
2. Syntax Error vs Runtime Error

In any programming language there are 2 types of errors are possible.

1. Syntax Errors
2. Runtime Errors

1. Syntax Errors:

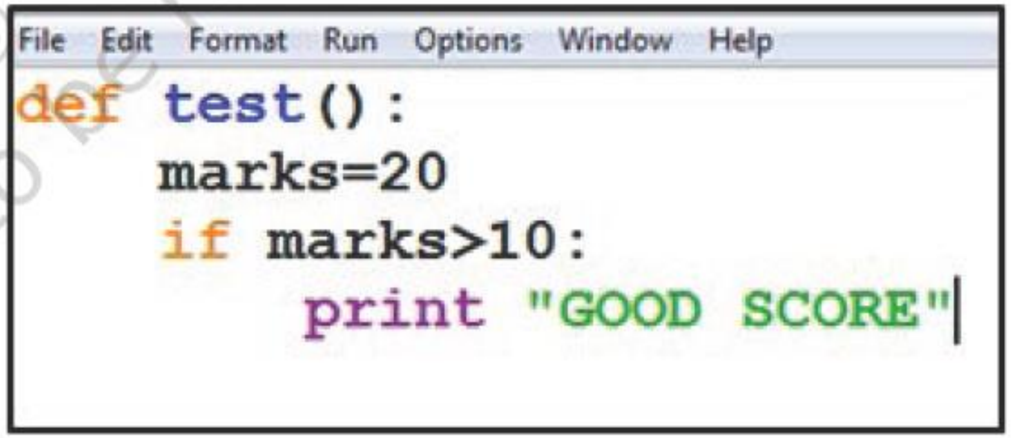
- Syntax errors are detected when we have not followed the rules of the particular programming language while writing a program.
- These errors are also known as parsing errors. On encountering a syntax error, the interpreter does not execute the program unless we rectify the errors, save and rerun the program.
- When a syntax error is encountered while working in shell mode, Python displays the name of the error and a small description about the error as shown in Figure 1.1.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> marks=10
>>> if marks>20:
        print "GOOD SCORE"

SyntaxError: Missing parentheses in call to 'print'. Did you mean print("GOOD SC
ORE")?
>>> 
```

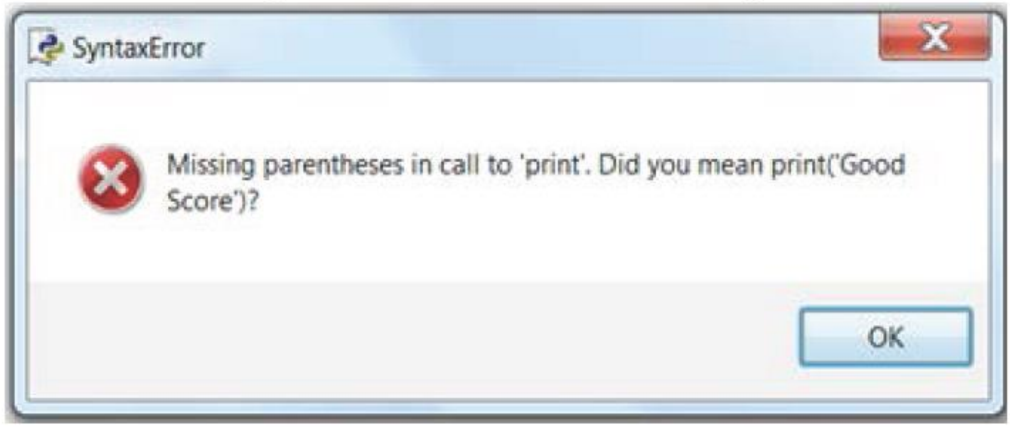
Figure 1.1: A syntax error displayed in Python shell mode

- So, a syntax error is reported by the Python interpreter giving a brief explanation about the error and a suggestion to rectify it.
- Similarly, when a syntax error is encountered while running a program in script mode as shown in Figure 1.2, a dialog box specifying the name of the error (Figure 1.3) and a small description about the error is displayed.



```
def test():
    marks=20
    if marks>10:
        print "GOOD SCORE"
```

Figure 1.2: An error in the script

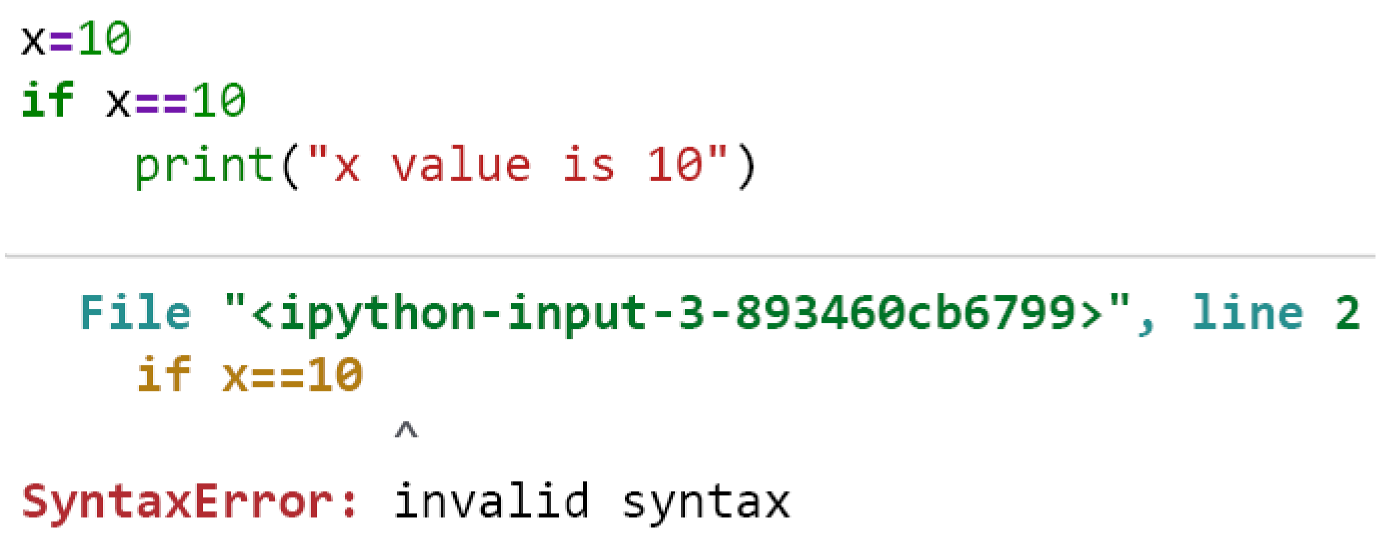


Missing parentheses in call to 'print'. Did you mean print('Good Score')?

Figure 1.3: Error dialog box

In simple words, the errors which occurs because of invalid syntax are called syntax errors.

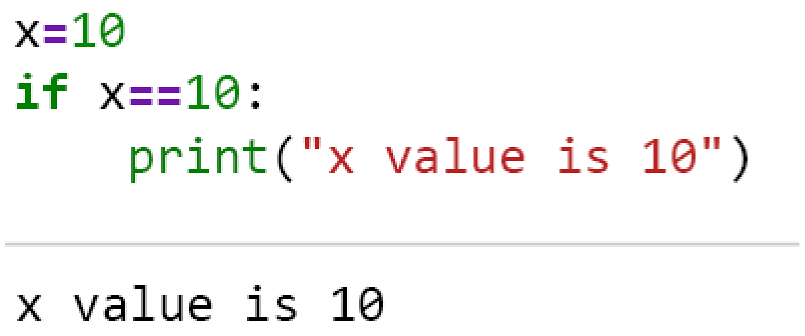
Eg:



```
x=10
if x==10
    print("x value is 10")
```

```
  File "<ipython-input-3-893460cb6799>", line 2
    if x==10
             ^
SyntaxError: invalid syntax
```

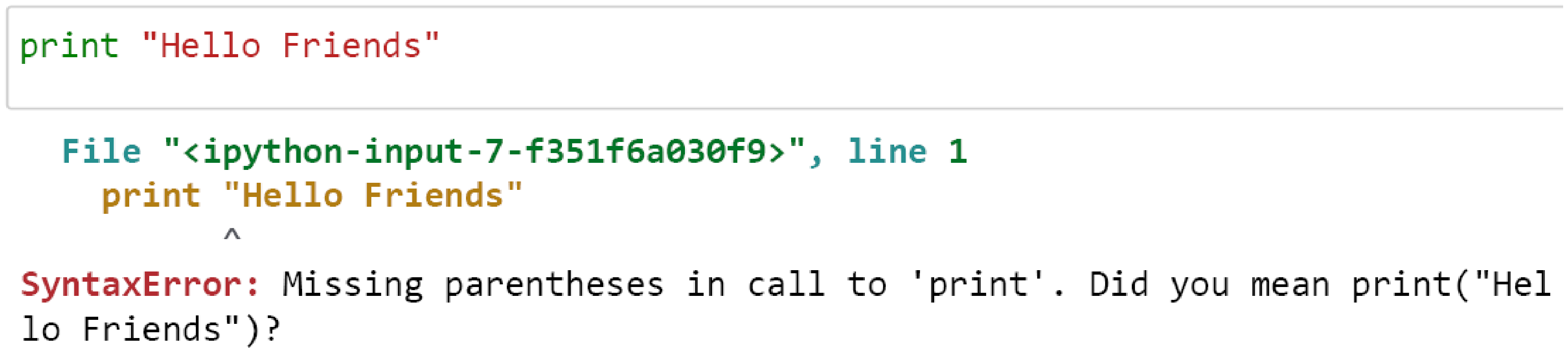
Eg:



```
x=10
if x==10:
    print("x value is 10")
```

```
x value is 10
```

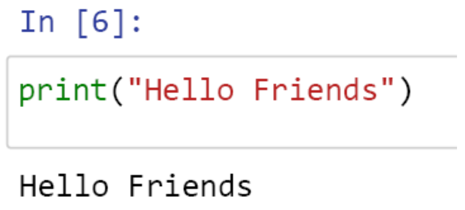
Eg:



```
print "Hello Friends"
```

```
  File "<ipython-input-7-f351f6a030f9>", line 1
    print "Hello Friends"
                        ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hel
lo Friends")?
```

Eg:



```
In [6]:
print("Hello Friends")

Hello Friends
```

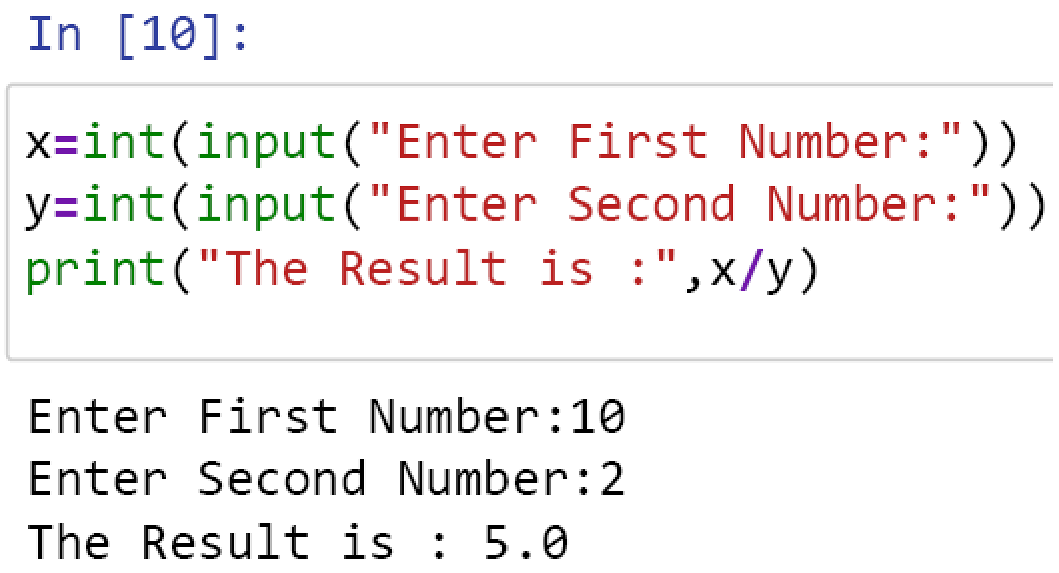
Note:

- Programmer is responsible to correct these syntax errors. Once all syntax errors are corrected then only program execution will be started.

2. Runtime Errors:

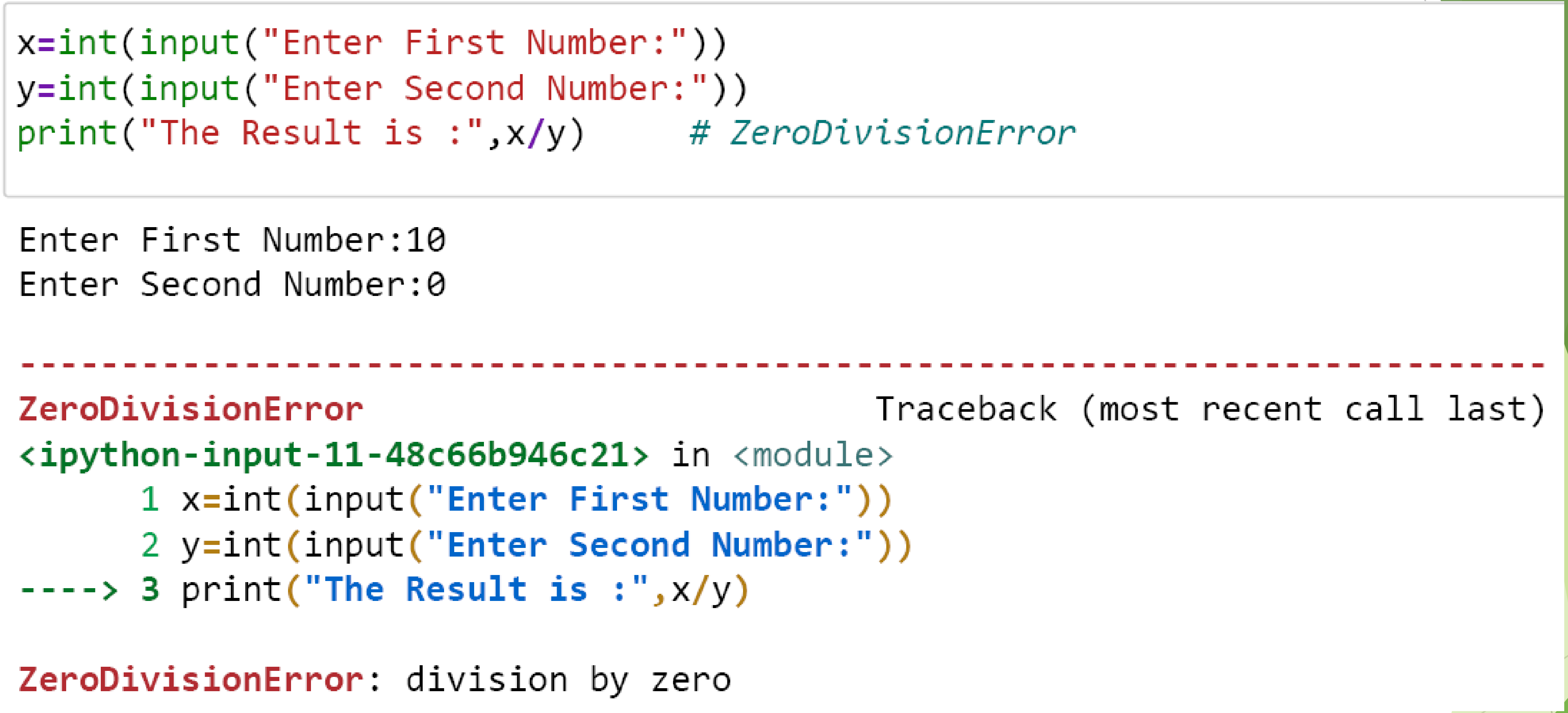
- These errors also known as Exceptions or Logical Errors.
- While executing the program, if something goes wrong because of end user input or programming logic or memory problems etc., then we will get Runtime Errors.

Eg 1:



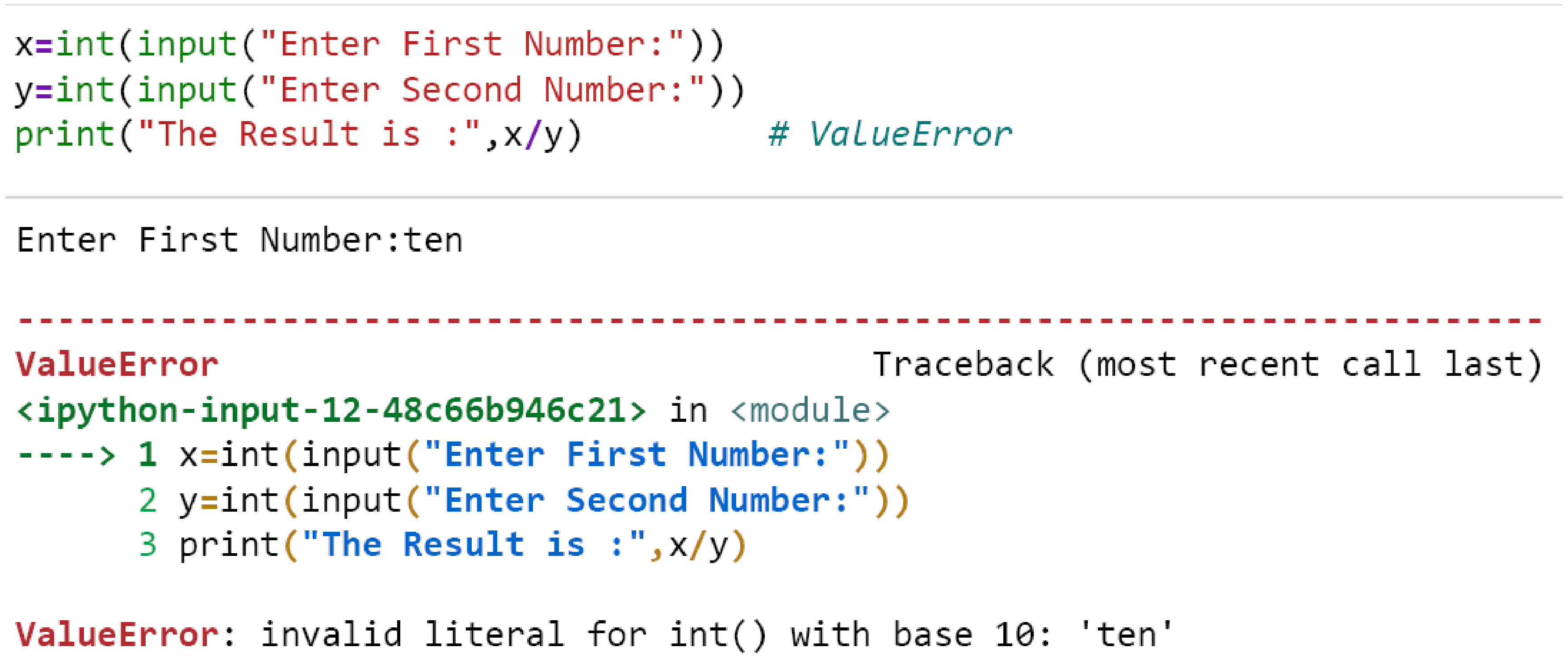
```
In [10]:
x=int(input("Enter First Number:"))
y=int(input("Enter Second Number:"))
print("The Result is :",x/y)
```

```
Enter First Number:10
Enter Second Number:2
The Result is : 5.0
```



```
x=int(input("Enter First Number:"))
y=int(input("Enter Second Number:"))
print("The Result is :",x/y)     # ZeroDivisionError
```

```
Enter First Number:10
Enter Second Number:0

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-11-48c66b946c21> in <module>
      1 x=int(input("Enter First Number:"))
      2 y=int(input("Enter Second Number:"))
----> 3 print("The Result is :",x/y)

ZeroDivisionError: division by zero
```



```
x=int(input("Enter First Number:"))
y=int(input("Enter Second Number:"))
print("The Result is :",x/y)         # ValueError
```

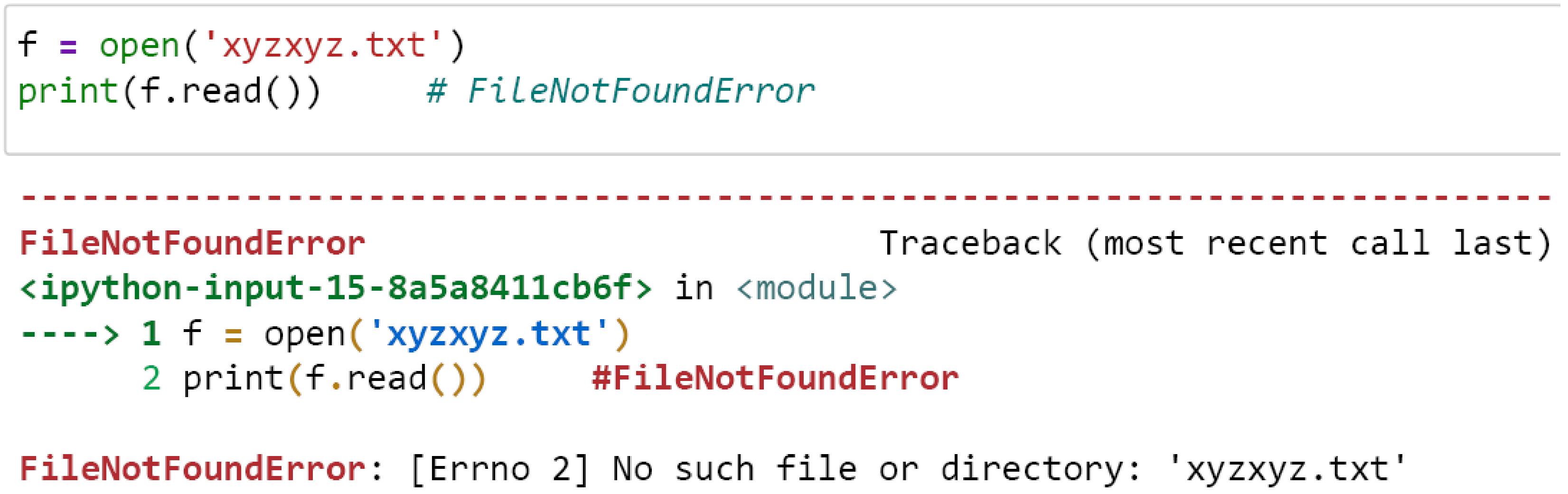
```
Enter First Number:ten

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-48c66b946c21> in <module>
----> 1 x=int(input("Enter First Number:"))
      2 y=int(input("Enter Second Number:"))
      3 print("The Result is :",x/y)

ValueError: invalid literal for int() with base 10: 'ten'
```

Here, ten can't convert into 'int' value,that's why we are getting **Value Error**.

In the previous example, we are getting following runtime errors:

1. ZeroDivisionError
2. ValueError

Eg 2:



```
f = open('xyzxyz.txt')
print(f.read())      # FileNotFoundError
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-15-8a5a8411cb6f> in <module>
----> 1 f = open('xyzxyz.txt')
      2 print(f.read())      #FileNotFoundError

FileNotFoundError: [Errno 2] No such file or directory: 'xyzxyz.txt'
```

Note: Exception Handling concept applicable for Runtime Errors but not for syntax errors.

Q. In Python, both syntax and logical errors occur at runtime. How to differentiate them???

- Assume that, there are 10000 lines python code. Actually Python is the interpreted programming language, directly we are going to execute the code, there is no compilation.
- Whenever, we are trying to execute this code, first PVM is going to check, is there any syntax errors in this code. Once all syntax errors are corrected, then only execution will be started.

- In Python, PVM take care about everything (synatx error checking also take care by PVM), but on other programming languages like C, C++ and Java, we have two separate components are there:

  1. Compiler ==> Syntax Checking
  2. Interpretter ==> Execution

3. Three Important Questions about Exception Handling

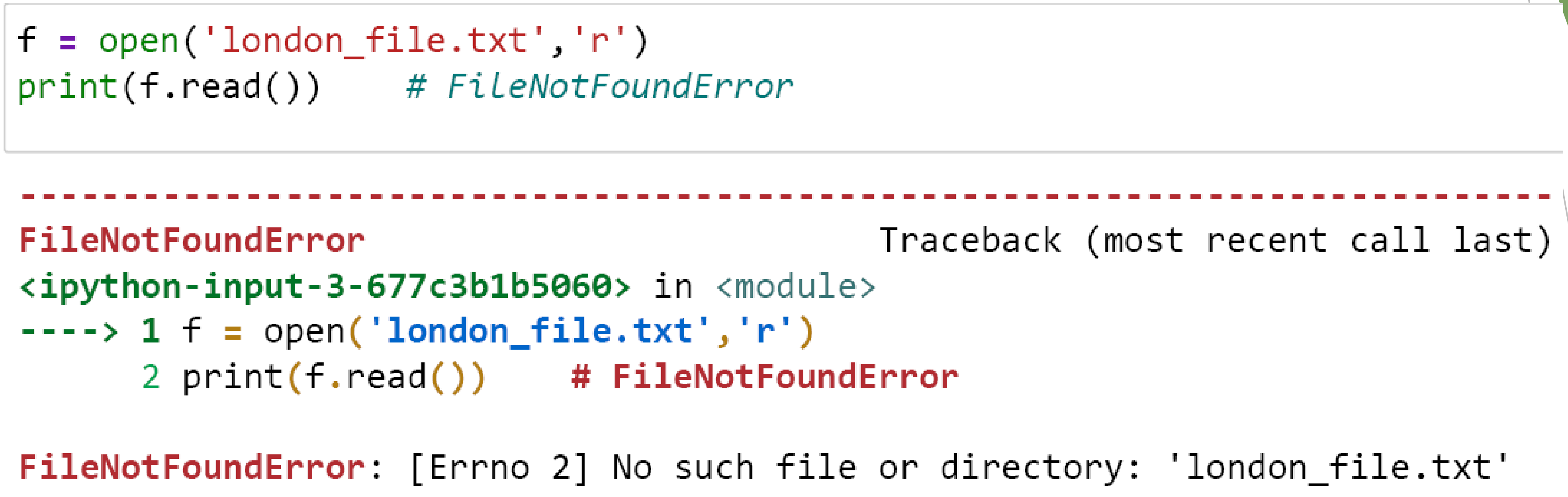
1. What is an Exception?

- Even if a statement or expression is syntactically correct, there might arise an error during its execution.
- **For example**, trying to open a file that does not exist, division by zero and so on. Such types of errors might disrupt the normal execution of the program and are called exceptions.
- In simple words, An unwanted and unexpected event that disturbs normal flow of program is called exception.

Examples of exceptions:

- InternetError
- SleepingError
- TyrePuncturedError
- ValueError
- FileNotFoundError
- TypeError
- ZeroDivisionError

- An exception is a Python object that represents an error. When an error occurs during the execution of a program, an exception is said to have been raised.
- Such an exception needs to be handled by the programmer so that the program does not terminate abnormally.
- Therefore, while designing a program, a programmer may anticipate such erroneous situations that may arise during its execution and can address them by including appropriate code to handle that exception.



```python
f = open('london_file.txt','r')
print(f.read())     # FileNotFoundError
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-3-677c3b1b5060> in <module>
----> 1 f = open('london_file.txt','r')
      2 print(f.read())     # FileNotFoundError

FileNotFoundError: [Errno 2] No such file or directory: 'london_file.txt'
```

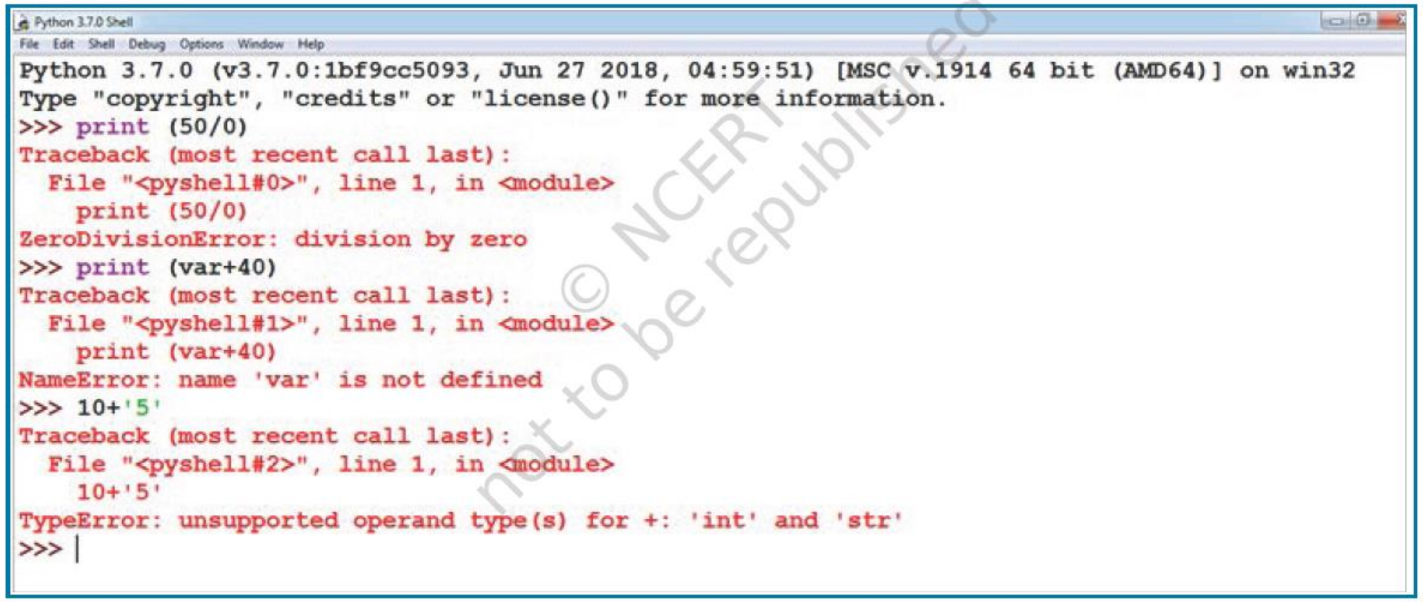
Built-in Exceptions:

- Commonly occurring exceptions are usually defined in the compiler/interpreter. These are called built-in exceptions.
- Python's standard library is an extensive collection of built-in exceptions that deals with the commonly occurring errors (exceptions) by providing the standardized solutions for such errors.
- On the occurrence of any built-in exception, the appropriate exception handler code is executed which displays the reason along with the raised exception name. The programmer then has to take appropriate action to handle it.
- Some of the commonly occurring built-in exceptions that can be raised in Python are explained in Table 1.1.

Table 1.1 Built-in exceptions in Python

S. No	Name of the Built-in Exception	Explanation
1.	SyntaxError	It is raised when there is an error in the syntax of the Python code.
2.	ValueError	It is raised when a built-in method or operation receives an argument that has the right data type but mismatched or inappropriate values.
3.	IOError	It is raised when the file specified in a program statement cannot be opened.
4	KeyboardInterrupt	It is raised when the user accidentally hits the Delete or Esc key while executing a program due to which the normal flow of the program is interrupted.
5	ImportError	It is raised when the requested module definition is not found.
6	EOFError	It is raised when the end of file condition is reached without reading any data by input().
7	ZeroDivisionError	It is raised when the denominator in a division operation is zero.
8	IndexError	It is raised when the index or subscript in a sequence is out of range.
9	NameError	It is raised when a local or global variable name is not defined.
10	IndentationError	It is raised due to incorrect indentation in the program code.
11	TypeError	It is raised when an operator is supplied with a value of incorrect data type.
12	OverFlowError	It is raised when the result of a calculation exceeds the maximum limit for numeric data type.

Figure 1.4 shows the built-in exceptions viz, ZeroDivisionError, NameError, and TypeError raised by the Python interpreter in different situations.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print (50/0)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print (50/0)
ZeroDivisionError: division by zero
>>> print (var+40)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print (var+40)
NameError: name 'var' is not defined
>>> 10+'5'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    10+'5'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> |
```

Figure 1.4: Example of built-in exceptions

A programmer can also create custom exceptions to suit one's requirements. These are called user-defined exceptions . We will learn how to handle such exceptions in the next sessions.

2. What is the main objective of Exception Handling?

- It is highly recommended to handle exceptions. The main objective of exception handling is Graceful Termination of the program (i.e., we should not block our resources and we should not miss anything).

3. What is the meaning of Exception Handling?

- Exception handling does not mean repairing exception. We have to define alternative way to continue rest of the program normally. This way of defining alternative is nothing but exception handling.
- For example, our programming requirement is reading data from remote file locating at London. At runtime if london_file is not available then the program should not be terminated abnormally. We have to provide local file to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

Eg: Consider the following two statements,

S1 : This weekend, I am planning to go to my native place by Bus.

S2 : This weekend, I am planning to go to my native place by Bus. If Bus is not available, then I will try for train. If train is not available then I will try for flight. Still it is not available, then I will go for Cab.

- First statement is the way of coding by the Fresher.
- Second statement is the way of coding by the Experienced Person. Because, he knows what kind of common problems are there. He is already ready with alternative ways to complete his task in the beginning only.

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You