PYTHON PROGRAMMING - II

Unit - 2

Python Exception Handling and Regular Expression

By-

Prof. A. P. Chaudhari

(M.Sc. Computer Science, SET)

HOD,

Department of Computer Science

S.V.S's Dadasaheb Rawal College, Dondaicha

Introduction

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.

Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Common Exceptions

- ZeroDivisionError: Occurs when a number is divided by zero.
- NameError: It occurs when a name is not found. It may be local or global.
- IndentationError: If incorrect indentation is given.
- IOError: It occurs when Input Output operation fails.
- EOFError: It occurs when end of the file is reached and yet operations are being performed. etc..

Handling Exception:

The suspicious code can be handled by using the try block. Enclose the code which raises an exception inside the try block. The try block is followed except statement. It is then further followed by statements which are executed during exception and in case if exception does not occur.

Syntax:

```
try:
    malicious code
except Exception1:
        execute code
except Exception2:
        execute code
        ....
        ....
except ExceptionN:
        execute code
else:
    In case of no exception, execute the else block code.
```

Handling Exception:

Example:

```
try:
    a=10/0
    print a
except ArithmeticError:
        print "This statement is raising an exception"
else:
    print "Welcome"
```

Output:

This statement is raising an exception

Handling Exception:

Python Exception(Except with no Exception) Example:

Except statement can also be used without specifying Exception.

Syntax:

```
try:
    code
except:
    code to be executed in case exception occurs.
else:
    code to be executed in case exception does not occur.
```

Example:

```
try:
    a=10/0;
except:
    print "Arithmetic Exception“
else:
    print "Successfully Done"
```

Output:

Arithmetic Exception

Multiple Except Clauses:

Declaring Multiple Exception in Python

Python allows us to declare multiple exceptions using the same except statement.

Syntax:

```
try:
    code
except Exception1,Exception2,Exception3,..,ExceptionN
    execute this code in case any Exception of these occur.
else:
    execute code in case no exception occurred.
```

Example:

```
try:
    a=10/0;
except ArithmeticError, StandardError:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

Output:

Arithmetic Exception

try … finally:

Finally Block:

In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.

Syntax:

```
try:
    Code
finally:
    code which is must to be executed.
```

Example

```
try:
    a=10/0;
    print "Exception occurred"
finally:
    print "Code to be executed"
```

Output:

```
>>>
Code to be executed
Traceback (most recent call last):
  File "C:/Python27/noexception.py", line 2, in <module>
    a=10/0;
ZeroDivisionError: integer division or modulo by zero
>>>
```

In the above example finally block is executed. Since exception is not handled therefore exception occurred and execution is stopped.

Raising Exception:

You can explicitly throw an exception in Python using ?raise? statement. raise will cause an exception to occur and thus execution control will stop in case it is not handled.

There are three forms of the raise statement:

raise

raise *E1*

raise *E1*, *E2*

Syntax:

raise Exception_class,<value>

Example:

```
try:
    a=10
    print a
    raise NameError("Hello")
except NameError as e:
        print "An exception occurred"
        print e
```

Output:

```
10
An exception occurred
Hello
```

Raising Exception:

i) To raise an exception, raise statement is used. It is followed by exception class name.

ii) Exception can be provided with a value that can be given in the parenthesis. (here, Hello)

iii)To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

User Defined Exception:

Creating your own Exception class or User Defined Exceptions are known as Custom Exception.

Example

```
class ErrorInCode(Exception):
    def __init__(self, data):
      self.data = data
    def __str__(self):
        return repr(self.data)

try:
    raise ErrorInCode(2000)
except ErrorInCode as ae:
    print "Received error:", ae.data
```

Output:

```
>>>
Received error : 2000
>>>
```

List of Standard Exceptions:

Sr.No.	Exception Name & Description
1	**Exception** Base class for all exceptions
2	**StopIteration** Raised when the next() method of an iterator does not point to any object.
3	**SystemExit** Raised by the sys.exit() function.
4	**StandardError** Base class for all built-in exceptions except StopIteration and SystemExit.
5	**ArithmeticError** Base class for all errors that occur for numeric calculation.
6	**OverflowError** Raised when a calculation exceeds maximum limit for a numeric type.
7	**FloatingPointError** Raised when a floating point calculation fails.
8	**ZeroDivisionError** Raised when division or modulo by zero takes place for all numeric types.
9	**AssertionError** Raised in case of failure of the Assert statement.

List of Standard Exceptions:

10	**AttributeError** Raised in case of failure of attribute reference or assignment.
11	**EOFError** Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	**ImportError** Raised when an import statement fails.
13	**KeyboardInterrupt** Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	**LookupError** Base class for all lookup errors.
15	**IndexError** Raised when an index is not found in a sequence.
16	**KeyError** Raised when the specified key is not found in the dictionary.
17	**NameError** Raised when an identifier is not found in the local or global namespace.
18	**UnboundLocalError** Raised when trying to access a local variable in a function or method but no value has been assigned to it.

List of Standard Exceptions:

19	**EnvironmentError** Base class for all exceptions that occur outside the Python environment.
20	**IOError** Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	**OSError** Raised for operating system-related errors.
22	**SyntaxError** Raised when there is an error in Python syntax.
23	**IndentationError** Raised when indentation is not specified properly.
24	**SystemError** Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	**SystemExit** Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
26	**TypeError** Raised when an operation or function is attempted that is invalid for the specified data type.

List of Standard Exceptions:

27	**ValueError** Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
28	**RuntimeError** Raised when a generated error does not fall into any category.
29	**NotImplementedError** Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.