BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on Exception Handling in Python.

List of Standard Exceptions in Python –

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.

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.

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 **IOError:-** 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.

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.

Argument of an Exception:-

An exception can have an argument, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause as follows −

Syntax:-

```
try:
   You do your operations here;
   ......................
except ExceptionType, Argument:
   You can print value of Argument here...
```

If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception.

This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location.

Example:- Following is an example for a single exception –

```
#!/usr/bin/python
# Define a function here.
def temp_convert(var):
   try:
```

```
      return int(var)
   except ValueError, Argument:
    print "Argument does not contain numbers\n", Argument
# Call above function here.
temp_convert("xyz");
```

Output:-

The argument does not contain numbers

invalid literal for int() with base 10: 'xyz'

Raising an Exceptions:-

You can raise exceptions in several ways by using the raise statement. The general syntax for the raise statement is as follows.

Syntax:-

```
raise [Exception [, args [, traceback]]]
```

Here, Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Example:-

An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows−

```
def functionName( level ):
   if level < 1:
      raise "Invalid level!", level
      # The code below to this would not be executed
      # if we raise the exception
```

Note: In order to catch an exception, an "except" clause must refer to the same exception thrown either class object or simple string. For example, to capture above exception, we must write the except clause as follows −

```
try:
   Business Logic here...
except "Invalid level!":
   Exception handling here...
else:
```

Rest of the code here...

User-Defined Exceptions:-

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Here is an example related to RuntimeError. Here, a class is created that is subclassed from RuntimeError. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

Example:-

```
class Networkerror(RuntimeError):
  def __init__(self, arg):
    self.args = arg
```

So once you defined above class, you can raise the exception as follows −

```
try:
  raise Networkerror("Bad hostname")
except Networkerror,e:
  print e.args
```