Exception Handling in Python

Course Contents

- What is Exception?
- Handling an exception
- The except Clause with No Exceptions
- The except Clause with Multiple Exceptions
- The try-finally Clause
- Argument of an Exception
- Raising an Exception

What is Exception?

• Python provides very important feature to handle any unexpected error in your Python programs and to add debugging capabilities in them called Exception.

• An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

• In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.

• An exception is a Python object that represents an error.

• When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception - I

• If we have some suspicious code that may raise an exception, we can defend our program by placing the suspicious code in a try: block.

• After the **try:** block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.
• Here is simple syntax of try....except...else blocks −

```
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.
```

Handling an exception - I

```
fh = open("example_Read.txt", "r")
for line in fh:
    print(line.rstrip())
fh.close()
```

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

```
try:
    fh = open("example_Read.txt", "r")
except IOError:
    print ("Error: can\'t find file or read data")
else:
    print ("Read content of the file successfully")
    fh.close()
print('Normal Execution again')
```

```
Error: can't find file or read data
Normal Execution again
```

- After the except clause(s), one can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.

- The else-block is a good place for code that does not need the try: block's protection.

Handling an exception - II

• A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.

• We can also provide a generic except clause, which handles any exception.

```
try:
    fh = open("example_Read.txt", "r")
    print ("Read content of the file successfully")
    x = 'ss' + 3
    print ("Going to close the file")
    fh.close()
except IOError:
    print ("Error: can\'t find file or read data")
except TypeError:
    print ('Type error generated')
    fh.close()
```

```
Error: can't find file or read data
```

The except Clause with No Exceptions

- One can also use the except statement with no exceptions defined as follows −

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

- This kind of a try-except statement catches all the exceptions that occur.

The except Clause with No Exceptions

- Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

```
try:
    fh = open("example_Read.txt", "r")
except :
    print ("Error: can\'t find file or read data")
else:
    print ("Read content of the file successfully")
    fh.close()
print('Normal Execution again')
```

```
Error: can't find file or read data
Normal Execution again
```

The except Clause with Multiple Exceptions

• One can also use the same except statement to handle multiple exceptions as follows −

```
try:
   You do your operations here
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

The except Clause with Multiple Exceptions

```
try:
    fh = open("test.txt", "r")
    print ("Read content of the file successfully")
    x = 'ss' + 3
    print ("Going to close the file")
    fh.close()
except (IOError, TypeError):
    print ("We have exception")
else:
    fh.close()
```

```
We have exception
```

The try-finally Clause

- One can use a finally: block along with a try: block.

- The finally: block is a place to put any code that must execute, whether the try-block raised an exception or not.

- The syntax of the try-finally statement is this −

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be skipped.
finally:
   This would always be executed.
   ......................
```

The try-finally Clause

```
try:
   fh = open("test.txt", "r")
   print ("Read content of the file successfully")
except IOError:
   print ("Error: can\'t find file or read data")
finally:
   print ("Done")
   fh.close()
```

```
Read content of the file successfully
Done
```

- One can provide except clause(s), with finally clause.
- One can use else clause as well along with a finally clause.

The try-finally Clause

```
try:
    fh = open("test.txt", "r")
    try:
        print ("Read content of the file successfully")
        x = 'ss' + 3
    finally:
        print ("Going to close the file")
        fh.close()
except IOError:
    print ("Error: can\'t find file or read data")
except TypeError:
    print ('Type error generated')
```

```
Read content of the file successfully
Going to close the file
Type error generated
```

Argument of an Exception - I

- 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.
- One capture an exception's argument by supplying a variable in the except clause as follows −

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

Argument of an Exception - II

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

```
try:
    fh = open("example_1.txt", "r")
    print ("Read content of the file successfully")
    x = 'ss' + 3
    print ("Going to close the file")
    fh.close()
except (IOError,TypeError) as Argument:
    print ("We have exception -- ", Argument)
    fh.close()
```

```
We have exception --  [Errno 2] No such file or directory: 'example_1.txt'
```

Argument of an Exception - II

```
# Define a function here.
def temp_convert(var):
      return int(var)

# Call above function here.
temp_convert("xyz")
```

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

```
# Define a function here.
def temp_convert(var):
    try:
        return int(var)
    except ValueError as Argument:
        print ("The argument does not contain numbers\n", Argument)

# Call above function here.
temp_convert("xyz")
```

```
The argument does not contain numbers
 invalid literal for int() with base 10: 'xyz'
```

Raising an Exception - I

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

```
>>> 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.

Raising an Exception - II

- 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 Exception (level)
      # The code below to this would not be executed
      # if we raise the exception
    return level
```

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

```
try:
    x = functionName(-10)
    print ("level = ",x)
except Exception as e:
    print ("error in level argument",e.args[0])
```

```
error in level argument -10
```

Raising User-defined Exceptions

```
class MyError(Exception):
    pass
```

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

```
try:
    x = functionName(-10)
    print ("level = ",x)
except MyError as e:
    print ("error in level argument",e.args[0])
```

```
error in level argument -10
```

Summary

- We have implemented Handling an exception
- We tried with The except Clause with No Exceptions and The except Clause with Multiple Exceptions
- Also tried with The try-finally Clause , Argument of an Exception and Raising an Exception.