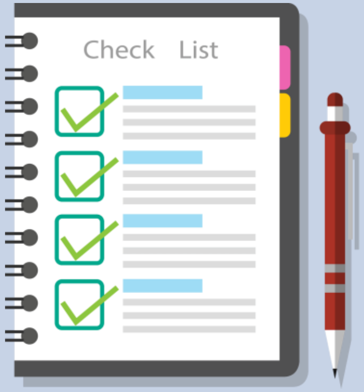
Python – Basics

Digital Lync
INNOVATION - EDUCATION - INCUBATION

Shah Ayub Quadri
aquadri@digital-lync.com

Index



Exception Handling

Exception

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

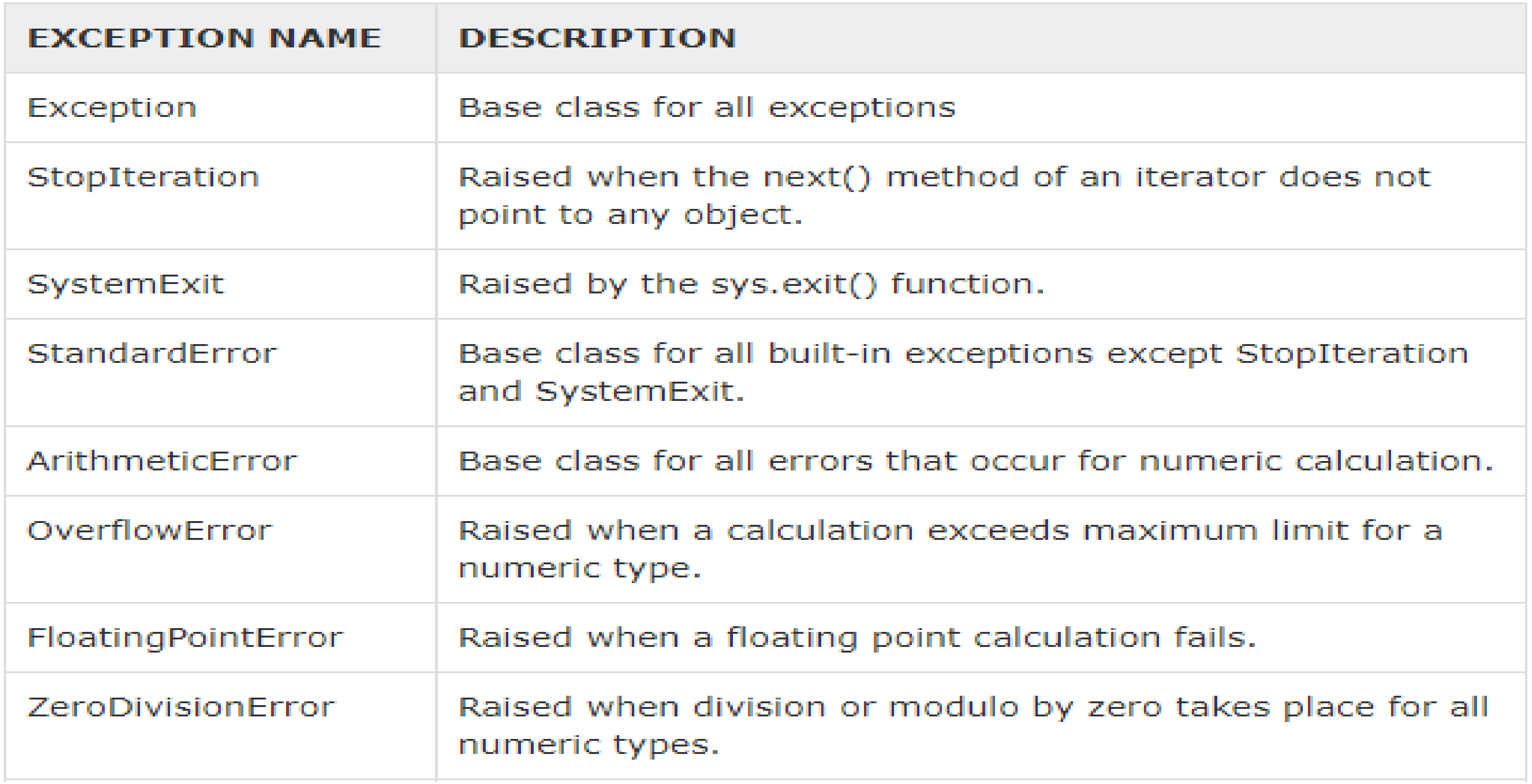
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.

- 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.

Exception Handling

List of Standard Exceptions:



EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.

Exception Handling

ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError	Base class for all lookup errors.
IndexError KeyError	Raised when an index is not found in a sequence. Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.

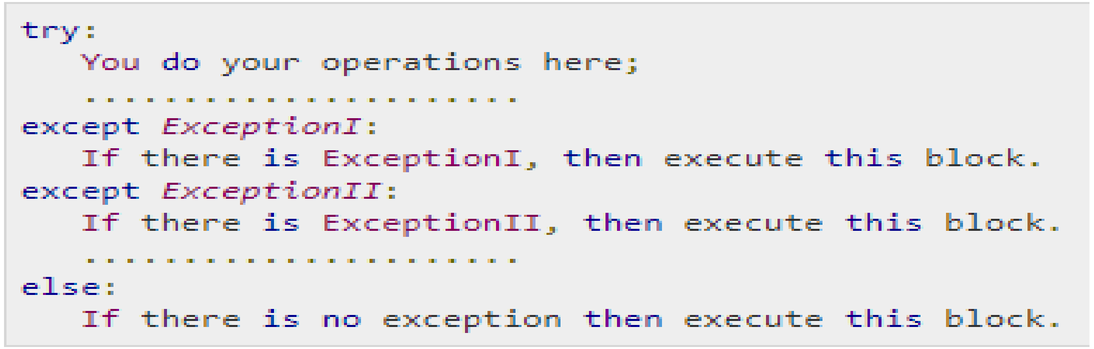
Exception Handling

Handling an exception:

- If you have some *suspicious* code that may raise an exception, you can defend your 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.

Syntax:



```
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.
```

Exception Handling

Few important points to remember about exceptions:

- 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.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you 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.

Examples:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print "Error: can\'t find file or read data"
else:
   print "Written content in the file successfully"
   fh.close()
```

```
Written content in the file successfully
```

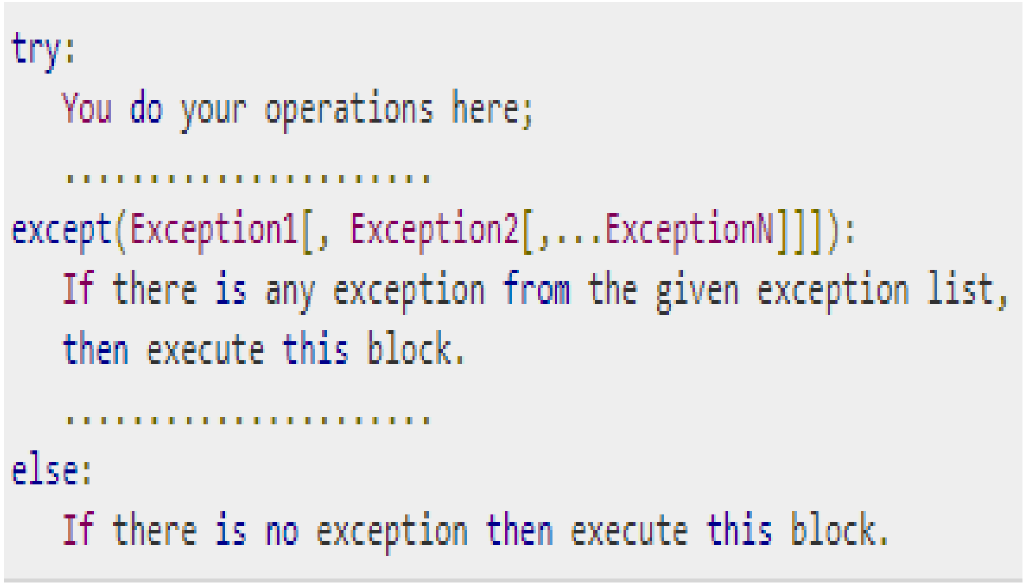
```
try:
   fh = open("testfile", "r")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print "Error: can\'t find file or read data"
else:
   print "Written content in the file successfully"
```

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

Exception Handling

The *except* Clause with Multiple Exceptions:

- You can also use the same *except* statement to handle multiple exceptions

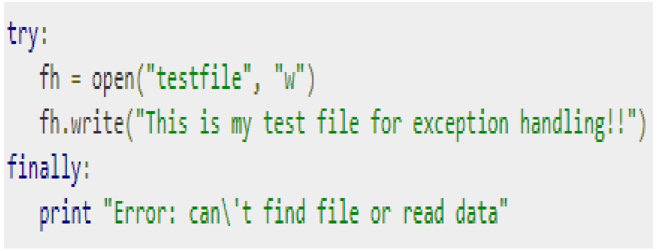


```
try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

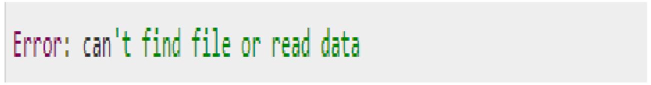
Exception Handling

try-finally Clause:

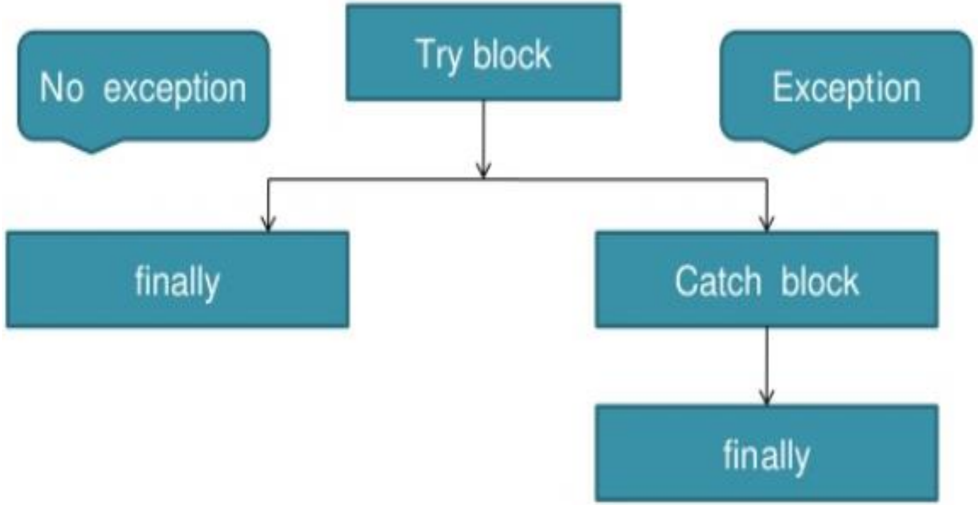
- You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.



```
try:
  fh = open("testfile", "w")
  fh.write("This is my test file for exception handling!!")
finally:
  print "Error: can\'t find file or read data"
```



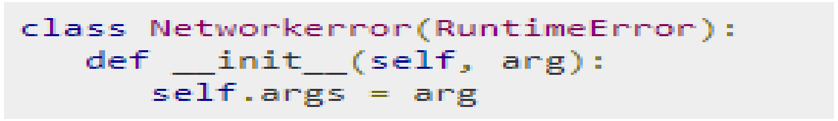
```
Error: can't find file or read data
```



Exception Handling

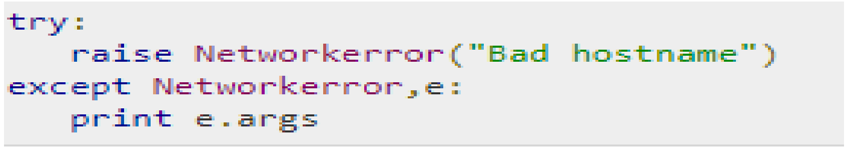
User-Defined Exceptions:

- 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*.



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

- So once you defined above class, you can raise the exception



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

Assignment

Q1. Write a program to do arithmetic operation using Exception handling (divide by zero)

Q2. Write a program to display different types of exceptions that can occur

a) When your trying to access a file which is not present at the given location
b) When your trying to write a file which is read only
c) When your trying to append a file which is read only

Note: through various exceptions and in finally block close the file connection

Q3. Write a program which includes try-multiple catch & finally blocks on