Content Delivered in Class_13_22-August-2016

- Chapter 6: Modules
  - eval, exec, execfile, compile and py_compile
- Chapter 7: Exceptions
  - Types of Errors
  - Error Handling in Python
  - The Exception Model
  - Exception Hierarchy
  - Handling Multiple Exceptions
  - raise
  - assert
  - settrace

Assignments Given

Assignment 1: Try to find out the various ways of getting true division in python 2.x , and in python 3.x individually.

---

eval,exec, execfile, compile and py_compile

eval(str,globals,locals) - This function executes an expression string and returns the result.

```
In [2]: eval('2+3')
```

```
Out[2]: 5
```

```
In [3]: eval("'Python'*3") # observe both single(') and double(") quotes
```

```
Out[3]: 'PythonPythonPython'
```

```
In [4]: eval("print 'Hello World!'")  # statement execution is not possible
```

```
  File "<string>", line 1
    print 'Hello World!'
        ^
SyntaxError: invalid syntax
```

exec(filename,globals,locals) - function executes the statements

In [5]:
```
exec("print 'Hello World!'")  # exec() executes a string containing arbitrary
 python code
```

```
Hello World!
```

In [6]:
```
exec('2+34')
```

In [7]:
```
a=[1,2,3,45]
exec "for i in a: print i"
```

```
1
2
3
45
```

In [8]:
```
exec("'Hello World!'")
```

execfile(filename,globals,locals) - function executes the contents of a file

In [9]:
```
#!/usr/bin/python
# fileName.py
print "HelloWorld!"

birds=['parrot','hen','hyna']
for b in birds:
        print b
```

```
HelloWorld!
parrot
hen
hyna
```

In [10]:
```
import os;
os.listdir(os.getcwd())
```

Out[10]:
```
['.ipynb_checkpoints',
 'class_13_21-August-2016.ipynb',
 'fileName.py',
 'subprocessModule.py']
```

Observe that 'fileName.py' is placed in the current working directory

In [11]:
```
execfile("fileName.py")  # To execute the python script
```

```
HelloWorld!
parrot
hen
hyna
```

In [12]:
```
globals={'x':7,'y':10,'birds':['parrot','pigeon','sparrow']}
```

In [14]:
```
locals={'x':2,'y':20}
```

In [15]:
```
a=eval("3*x+4*y",globals,locals)     # locals are preferred
```

In [16]:
```
a
```

Out[16]: 86

In [17]:
```
locals ={}
```

In [20]:
```
a=eval("3*x+4*y",globals,locals)   # In the absence of locals, globals are cho
osen
```

In [19]:
```
a
```

Out[19]: 61

In [21]:
```
exec "for b in birds: print b" in globals,locals
```

```
parrot
pigeon
sparrow
```

In [22]:
```
execfile("fileName.py",globals,locals)   # Values present within script are pr
eferred
```

```
HelloWorld!
parrot
hen
hyna
```

Observe that the values for 'birds' is preferred to from 'fileName.py'

In [23]:
```
#!/usr/bin/python
# fileName.py
print "HelloWorld!"


for b in birds:
        print b
```

```
HelloWorld!
parrot
hen
hyna
```

In [29]:
```
del birds
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-24db1c1aae6d> in <module>()
----> 1 del birds

NameError: name 'birds' is not defined
```

In [33]:
```
#!/usr/bin/python
# fileName.py
print "HelloWorld!"


for b in birds:
        print b
```

```
HelloWorld!

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-74f39257017d> in <module>()
      4
      5
----> 6 for b in birds:
      7         print b

NameError: name 'birds' is not defined
```

In [35]:
```
execfile("fileName.py",globals,locals)
```

```
HelloWorld!
parrot
hen
hyna
```

In [38]:
```
who    # results in the list of identifier in the interpreter heap
```

```
a        b       globals         i       locals  os
```

In [40]:
```
whos   # same as 'who', but with description
```

```
No variables match your requested type.
```

when a string is passed to exec,eval(), or execfile(), parser first compiles to create bytecode.

To remove this redundant process every time, compile will create precompiled bytecode, which can be used everytime, till the code is not changed

compile (str, filename, kind) function a string compiled into byte code, str is the string to be compiled, the filename is to define the string variable file, the kind parameter specifies the type of code is compiled

- ' Single 'refers to a single statement,
- ' exec 'means more than one statement,
- ' eval 'means an expression.

compile () function returns a code object, the object, of course, can also be passed to the eval () function and the exec statement to perform, for example, :

In [42]:
```
str = "for i in range (0,10): print i,"
c = compile (str,'', 'exec') # compiled to byte code object
exec c # execution
```

```
0 1 2 3 4 5 6 7 8 9
```

In [56]:
```
str2 = "3 * x + 4 * y"
c2 = compile (str2,'', 'eval') # compiled expression
```

In [58]:
```
print c2, type(c2) # code object
```

```
<code object <module> at 045DCC38, file "", line 1> <type 'code'>
```

In [55]:
```
exec c2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-55-62a1bf112bbb> in <module>()
----> 1 exec c2

x=2, y=3 in <module>()

NameError: name 'x' is not defined
```

we are unable to pass the values to it

py_compile - It is a module to create bytecode file, *.pyc*

In [59]:
```
import py_compile
file = raw_input ("Please enter filename:")
```

```
Please enter filename:fileName.py
```

In [60]:
```
py_compile.compile(file)
```

In [61]:
```
import os
os.listdir(os.getcwd())  # observe the .pyc file
```

Out[61]:
```
['.ipynb_checkpoints',
 'class_13_21-August-2016.ipynb',
 'fileName.py',
 'fileName.pyc',
 'subprocessModule.py']
```

In Python 2.x, input(...) is equivalent to eval(raw_input(...))

In Python 3.x, raw_input was renamed input

Exceptions

Almost all programming languages, except shell scripting and some scripting languages, possess exception handling capabilities.

There are two kinds of errors in Python.

1. error code - If something went wrong, the resulting error code is -1 to indicate the failure of a call.
2. Exception - Used to handle exceptional cases.

In Python, the errors are handled by the interpreter by raising an exception and allowing that exception to be handled.

Exceptions indicate errors and break out of the normal control flow of a program. An exception is raised using raise statement.

```
Syntax:
    try:
        logic
        ...
    except <ExceptionName1>, <alias identifier>:
        logic to handle that exception
        ...
    except <ExceptionName2> as <alias identifier>:
        logic to handle that exception.
        This logic gets executed, if error is not covered
        in ExceptionName1 exception
        ...
    else:
        logic to execute if
        there is no exception
        ...
    finally:
        logic to execute either
        if exception occurs are not
        ...
```

Note : *try* and *except* are mandatory blocks. And, *else* and *finally* are optional blocks.

In [63]:
```
try:
    result = 21/0
except:
    print 'An error occurred!'
```

```
An error occurred!
```

This code handles all exceptions. But, we should know the error to do corresponding action

In [64]:
```
try:
    result = 21/0
except:
    print 'An error occurred'
    print 'The error is %s'%(ex)
```

```
An error occurred

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-64-e878884284c3> in <module>()
      3 except:
      4     print 'An error occurred'
----> 5     print 'The error is %s'%(ex)
      6

NameError: name 'ex' is not defined
```

Here, the exception was resulted in the exception block

In [66]:
```
try:
    result = (1+2.3)/(2*4*0)
except:
    print 'An error occurred'
    try:
        print 'The error is %s'%(ex)
    except:
        print 'variable "ex" is not defined'
```

```
An error occurred
variable "ex" is not defined
```

In [67]:
```
import exceptions
try:
        result = (1+2.3)/(2*4*0)
except exceptions.Exception, ex:
        print "The error is ", ex
```

```
The error is  float division by zero
```

In [68]:
```
result = (1+2.3)/(2*4*0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-68-2840635fb8b3> in <module>()
----> 1 result = (1+2.3)/(2*4*0)

ZeroDivisionError: float division by zero
```

In [75]:
```
try:
        result = (1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The error is ", repr(ex)
        print "The error is ", ex
```

```
The error is  ZeroDivisionError('float division by zero',)
The error is  float division by zero
```

In [76]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The error is ", ex
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-76-d6eb5a1e6e41> in <module>()
      1 try:
----> 2         result = (w)*(1+2.3)/(2*4*0)
      3 except ZeroDivisionError, ex:  # better handling of exception
      4         print "The error is ", ex

NameError: name 'w' is not defined
```

Ensure that all the exceptions are handled in the code

In [77]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The error is ", ex
except NameError, ex:
    print "The error is ", ex
```

```
The error is  name 'w' is not defined
```

There should be a except to accept any unknown exception

In [79]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:
        print "The other error is ", ex
```

```
The NameError is  name 'w' is not defined
```

```
control flow:
    when there is no exception:
        try -> else -> finally
    when there is exception:
        try -> except -> finally
```

In [80]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
The NameError is  name 'w' is not defined
Completed the try exception block
```

In [81]:
```
try:
        result = (1+2.3)/(2*4*10)
except ZeroDivisionError, ex:  # better handling of exception
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
try block executed successfully
The result is  0.04125
Completed the try exception block
```

As 'finally' gets executed in either case, it is seldom used. finally is generally used for cleanup action

In [82]:
```
try:
    x = float(raw_input("Your number: "))
    inverse = 1.0 / x
finally:
    print("There may or may not have been an exception.")
print "The inverse: ", inverse
```

```
Your number: 12321
There may or may not have been an exception.
The inverse:  8.11622433244e-05
```

In [83]:
```
try:
    x = float(raw_input("Your number: "))
    inverse = 1.0 / x
finally:
    print("There may or may not have been an exception.")
print "The inverse: ", inverse
```

```
Your number: 'Python'
There may or may not have been an exception.

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-83-cd0c760fc506> in <module>()
      1 try:
----> 2     x = float(raw_input("Your number: "))
      3     inverse = 1.0 / x
      4 finally:
      5     print("There may or may not have been an exception.")

ValueError: could not convert string to float: 'Python'
```

Here, exception is throwed, as except block is missing

In [85]:
```
try:
    x = float(raw_input("Your number: "))
    inverse = 1.0 / x
except exceptions.Exception, ex:
        print "The error is ", ex
finally:
    print("There may or may not have been an exception.")
print "The inverse: ", inverse
```

```
Your number: 'Python'
The error is  could not convert string to float: 'Python'
There may or may not have been an exception.
The inverse:  8.11622433244e-05
```

Built-in exceptions:

```
- Exception                              - root of all exceptions
    - SystemExit                         - generated by sys.exit()
    - StopIteration                      - Raised to stop iteration
    - StandardError                      - Base of all built-in exceptions
        - ArithmeticError                - Base for arithmetic exceptions
            - FloatingPointError         - faliture of a floating-point operation
            - OverflowError              - Arithmetic overflow
            - ZeroDivisionError          - Division or modulus operation with 0
        - AssertionError                 - Raised by assert statement
        - AttributeError                 - Raised when an attribute name is inval
id
        - EnvironmentError               - Errors that occur externally to python

            - IOError                    - I/O or file-related error
            - OSError                    - Operating System error
        - EOFError                       - Raised when end of file is reached
        - ImportError                    - Failure of import statement
        - KeyboardInterrupt              - Generated by interrupt key (ctrl+C or
 ctrl+D)
        - LookupError                    - Indexing and key errors
            - IndexError                 - Out-of-range sequence offset
            - KeyError                   - Non-existent dictionary key
        - MemoryError                    - Out of memory error
        - NameError                      - Failure to find a local or global name
            - UnboundLocalError          - unbound local variable
        - ReferenceError                 - Weak reference used after referent des
troyed
        - RuntimeError                   - A generic catch-all error
            - NotImplementedError        - unimplemented feature
        - SyntaxError                    - Parsing Error
            - IndentationError           - Indentation error
            - TabError                   - Inconsistent tab usage(generated with
 -tt option)
        - SystemError                    - Non-fatal system error in interpreter
        - TypeError                      - Passing an inappropriate type to an op
eration
        - ValueError                     - Invalid type
            - UnicodeError               - unicode error
                - UnicodeDecodeError     - unicode decoding error
                - UnicodeEncodeError     - unicode encoding error
                - UnicodeTranslateError  - unicode translation error
```

In [89]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)   # This statement contains both NameErro
r and ZeroDivisionError
except NameError, ex:
    print "The NameError is ", ex
except ZeroDivisionError, ex:      # better handling of exception
        print "The ZeroDivisionError is ", ex
except StandardError, ex:
    print "All Standard Errors are handled here"
    print "The error is ",ex
except exceptions.Exception, ex:
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
The NameError is  name 'w' is not defined
Completed the try exception block
```

In the above example, notice that NameError caught the exception, rather than StandardError, due to its placement preceeding StandardError exception.

In [90]:
```
import exceptions
try:
        result = (12)*(1+2.3)/(2*4*0)   # This statement contains ONLY ZeroDiv
isionError
except exceptions.Exception, ex:    # It handles all exceptions
   print "ALL the errors are handled here"
   print "The other error is ", ex
except StandardError, ex:           # handles only Standard Errors
    print "All Standard Errors are handled here"
    print "The error is ",ex
except ZeroDivisionError, ex:
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
ALL the errors are handled here
The other error is  float division by zero
Completed the try exception block
```

In the above scenario, notice that Exception method of exceptions module is handling all the exceptions, rather than the StandardError exception, due to its heirarchy.

In [91]:
```
import exceptions
try:
        result = (12)*(1+2.3)/(2*4*0)   # This statement contains ONLY ZeroDiv
isionError
except StandardError, ex:           # handles only Standard Errors
    print "All Standard Errors are handled here"
    print "The error is ",ex
except ZeroDivisionError, ex:
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:    # It handles all exceptions
   print "ALL the errors are handled here"
   print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
All Standard Errors are handled here
The error is  float division by zero
Completed the try exception block
```

Observe that the placement of execpt blocks, will effect the exception handling.

In [92]:
```
#!/usr/bin/python
# exceptionsExample1.py
import exceptions

try:
  a = int(input('please enter your number 1:'))
except NameError:
  print "we should not enter zero"         # It is a logic error. Logical erro
rs can't be handled
else:
  print "the number a:%d" %(a)
finally:
  print "finally clause"
```

```
please enter your number 1:python
we should not enter zero
finally clause
```

In [95]:
```
#!/usr/bin/python
# exceptionsExample1.py
import exceptions

try:
  a = int(input('please enter your number 1:'))
except NameError as ex:
  print 'NameError occurred'
  print ex
else:
  print "the number a:%d" %(a)
finally:
  print "finally clause"
```

```
please enter your number 1:python
NameError occurred
name 'python' is not defined
finally clause
```

In [98]:
```
#!/usr/bin/python
# exceptionsExample2.py

try:
  age = input('please enter your age:')
  print "my age is:", age
except NameError,error:
  print "please enter your age in numeric"
except:
    print 'someElse error'


print "----- report --------"
print "We got the following error : %s" %(error)     # identifier 'error' is b
eing used outside of the try-except blocks
```

```
please enter your age:ram
please enter your age in numeric
----- report --------
We got the following error : name 'ram' is not defined
```

In [99]:
```
#!/usr/bin/python
import sys

try:
 value1 = input("Please enter the value1:")
 value2 = input("Please enter the value2:")
except NameError,error:
 print "Please enter only numbers !!!"

print "ERROR: %s" %(error)
```

```
Please enter the value1:12
Please enter the value2:car
Please enter only numbers !!!
ERROR: name 'car' is not defined
```

In [101]:
```
#!/usr/bin/python

try:
  a = input('please enter your number 1:')
  b = input('please enter your number 2:')
except NameError,error:
  print "please enter numbers only \n"
  print "exception:", error
else:
  print "You enter the right number a:%d b:%d" %(a,b)


#try:
#  print "division of both numbers:" , a/b
#except ZeroDivisionError:
#  print "we cannot divide by zero"
```

```
please enter your number 1:123
please enter your number 2:machine
please enter numbers only

exception: name 'machine' is not defined
```

In [102]:
```
#!/usr/bin/python

try:
  a = input('please enter your number 1:')
  b = input('please enter your number 2:')
except NameError,error:
  print "please enter numbers only \n"
  print "exception:", error
else:
  print "You enter the right number a:%d b:%d" %(a,b)


#try:
#  print "division of both numbers:" , a/b
#except ZeroDivisionError:
#  print "we cannot divide by zero"
```

```
please enter your number 1:123
please enter your number 2:234
You enter the right number a:123 b:234
```

In [104]:
```
#!/usr/bin/python

try:
  num1 = int(raw_input("please enter a number1:"))
  num2 = int(raw_input("please enter a number2:"))
except ValueError:
  print "Please enter a number[0-9]"
else:
  print num1,num2
```

```
please enter a number1:nano car
Please enter a number[0-9]
```

In [2]:
```
#!/usr/bin/python
import sys


try:
 value1 = input("Please enter the value1:")
 value2 = input("Please enter the value2:")
except NameError:
 print "Please enter only numbers !!!"
 sys.exit(1)
except exceptions.Exception, ex:    # It handles all exceptions
   print "ALL the errors are handled here"
   print "The other error is ", ex
else:
 print "The division of two numbers:" , value1/value2
```

```
Please enter the value1:12
Please enter the value2:python
Please enter only numbers !!!

An exception has occurred, use %tb to see the full traceback.

SystemExit: 1

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\python27\lib\site-packages\IPython\core\interactiveshell.pyc in run_code(s
elf, code_obj, result)
   2875                result.error_in_exec = e
   2876            self.showtraceback(exception_only=True)
-> 2877            warn("To exit: use 'exit', 'quit', or Ctrl-D.", level=1)
   2878        except self.custom_exceptions:
   2879            etype, value, tb = sys.exc_info()

TypeError: 'level' is an invalid keyword argument for this function
```

observe 'SystemExit: 1'

In [3]:
```
#!/usr/bin/python

try:
  value1 = input("Please enter your first number")
  value2 = input("please enter your second number")
except NameError:
  print "please enter numbers only \n"

try:
  print "division of two numbers is : " , float(value1)/value2
except (NameError,ZeroDivisionError):
  print " \n please enter a valid number"
```

```
Please enter your first number12
please enter your second number34
division of two numbers is :  0.352941176471
```

In [5]:
```
del value1, value2
```

In [8]:
```
#!/usr/bin/python

try:
  value1 = input("Please enter your first number: ")
  value2 = input("please enter your second number: ")
except NameError:
  print "please enter numbers only \n"
  try:
    print "division of two numbers is : " , float(value1)/value2
  except (NameError,ZeroDivisionError):             # handling these two execp
tions together
    print " \n please enter a valid number"
```

```
Please enter your first number: zeep
please enter numbers only

division of two numbers is :
 please enter a valid number
```

Multiple Exception Handling:

```
try:
    <logic>
except TypeError, ex:
    <logic to handle Type errors>
except IOError, ex:
    <logic to handle IO errors>
except NameError, ex:
    <logic to handle name errors>
except Exception, ex:
    <logic to handle any other error>

Also, single handler can catch multiple exception types.
try:
    <logic>
except (IOError, TypeError, NameError), ex:
    <logic to handle IOError, TypeError, NameError>
```

In [9]:
```
#!/usr/bin/python

try:
    value1 = int(raw_input("please enter number 1:"))
    value2 = int(raw_input("please enter number 2:"))
except ValueError:
    print "Buddy.. its number \n"
    try:
      print "division of numbers", value1/value2
    except  ValueError,error:
      print "Buddy .. no number given .. try again .. \n"
      print "ERROR:{}".format(error)
    except ZeroDivisionError,error:
      print "Zero is not the number you would enter \n"
      print "ERROR:{}".format(error)
    except NameError,error:
      print "value is not defined"
      print "ERROR:{}".format(error)
    else:
      print "The division of numbers is success \n"
```

```
please enter number 1:45
please enter number 2:jupyter
Buddy.. its number

division of numbers value is not defined
ERROR:name 'value2' is not defined
```

In [10]:
```
#!/usr/bin/python

try:
    value1 = int(raw_input("please enter number 1:"))
    value2 = int(raw_input("please enter number 2:"))
except ValueError:
    print "Buddy.. its number \n"
    try:
      print "division of numbers", value1/value2
    except  (ValueError,ZeroDivisionError, NameError),error:
      print "Buddy .. it is either ValueError,ZeroDivisionError, NameError ..
 try again .. \n"
      print "ERROR:{}".format(error)
    else:
      print "The division of numbers is success \n"
```

```
please enter number 1:45
please enter number 2:jupyter
Buddy.. its number

division of numbers Buddy .. it is either ValueError,ZeroDivisionError, NameE
rror .. try again ..

ERROR:name 'value2' is not defined
```

It is not recommended, as it doesn't specify the exact problem

Raising exceptions

raise instance

raise class

In [11]:
```
raise
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-26814ed17a01> in <module>()
----> 1 raise

TypeError: exceptions must be old-style classes or derived from BaseExceptio
n, not NoneType
```

In [12]:
```
raise IOError
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-12-f6cf32b12b43> in <module>()
----> 1 raise IOError

IOError:
```

In [13]:
```
raise KeyboardInterrupt
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-13-7d145351408f> in <module>()
----> 1 raise KeyboardInterrupt

KeyboardInterrupt:
```

In [14]:
```
raise MemoryError("The memory is running out of space")
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-14-9c0834b76ffd> in <module>()
----> 1 raise MemoryError("The memory is running out of space")

MemoryError: The memory is running out of space
```

In [15]:
```
try:
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
except ValueError as ve:
    print(ve)
```

```
Enter a positive integer: -23
That is not a positive number!
```

In [16]:
```
#!/usr/bin/python

size = int(raw_input("please enter the size"))

if size < 50:
  print "we have good amount of space"
if size > 50 and size < 80:
  raise UserWarning,"We are hitting 80 percentage on disk"
if size > 90:
  raise UserWarning,"we have a issue with 100 full space"
```

```
please enter the size53

---------------------------------------------------------------------------
UserWarning                               Traceback (most recent call last)
<ipython-input-16-bd6304f90c07> in <module>()
      6   print "we have good amount of space"
      7 if size > 50 and size < 80:
----> 8   raise UserWarning,"We are hitting 80 percentage on disk"
      9 if size > 90:
     10   raise UserWarning,"we have a issue with 100 full space"

UserWarning: We are hitting 80 percentage on disk
```

Raising custom exceptions

In [17]:
```
print 'Yash'
```

```
Yash
```

In [18]:
```
print Yash
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-508b0b8c7cfb> in <module>()
----> 1 print Yash

NameError: name 'Yash' is not defined
```

In [19]:
```
raise NameError
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-a586bde47246> in <module>()
----> 1 raise NameError

NameError:
```

In [20]:
```
raise NameError("This is a name error")   # python 2 and python 3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-b2d352285537> in <module>()
----> 1 raise NameError("This is a name error")   # python 2 and python 3

NameError: This is a name error
```

In [21]:
```
raise NameError, "This is a name error" # only in python 2.x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-1da8dbca750e> in <module>()
----> 1 raise NameError, "This is a name error" # only in python 2.x

NameError: This is a name error
```

exception need to be a class object

In [22]:
```
class Yash(Exception(" This is Yash exception")):
    pass
```

In [23]:
```
raise Yash
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-23-0f7e1d67f2be> in <module>()
----> 1 raise Yash

Exception: ('Yash', (Exception(' This is Yash exception',),), {'__module__':
 '__main__'})
```

In [24]:
```
import exceptions
class Yash(exceptions.Exception(" This is Yash exception")):
    pass
```

In [25]:
```
raise Yash
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-25-0f7e1d67f2be> in <module>()
----> 1 raise Yash

Exception: ('Yash', (Exception(' This is Yash exception',),), {'__module__':
 '__main__'})
```

In [27]:
```
class Ankit(exceptions.Exception):
    pass
```

In [30]:
```
raise Yash
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-30-0f7e1d67f2be> in <module>()
----> 1 raise Yash

Exception: ('Yash', (Exception(' This is Yash exception',),), {'__module__':
 '__main__'})
```

In [31]:
```
who
```

```
Ankit    Yash    a       error   exceptions      size    sys     value1  ve
```

In [32]:
```
del exceptions
```

In [33]:
```
raise Yash
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-33-0f7e1d67f2be> in <module>()
----> 1 raise Yash

Exception: ('Yash', (Exception(' This is Yash exception',),), {'__module__':
 '__main__'})
```

Defining custom exceptions

- All built-in exceptions are defined in terms of classes.
- To create a new exception, create a new calss definition that inherits from exceptions.Exception

In [38]:
```
#Ex1:
class Networkerror(RuntimeError):
   def __init__(self, arg):
      self.args = arg

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

```
('B', 'a', 'd', ' ', 'h', 'o', 's', 't', 'n', 'a', 'm', 'e')
```

In [41]:
```
#Ex2:
import exceptions
#Exception class
class NetworkError(exceptions.Exception):
        def __init__(self, errno, msg):
                self.args = (errno, msg)
                self.errno = errno
                self.errno      = msg

# Raises an exception (multiple arguments)
def error2():
        raise NetworkError(1, 'Host not found')

# Raises an exception (multiple arguments supplied as a tuple)
def error3():
        raise NetworkError, (1, 'Host not found')
```

In [42]:
```
error2() # function call
```

```
---------------------------------------------------------------------------
NetworkError                              Traceback (most recent call last)
<ipython-input-42-c024a2970039> in <module>()
----> 1 error2() # function call

<ipython-input-41-8077a043d45e> in error2()
     10 # Raises an exception (multiple arguments)
     11 def error2():
---> 12         raise NetworkError(1, 'Host not found')
     13
     14 # Raises an exception (multiple arguments supplied as a tuple)

NetworkError: (1, 'Host not found')
```

In [43]:
```
error3() # function call
```

```
---------------------------------------------------------------------------
NetworkError                              Traceback (most recent call last)
<ipython-input-43-efd211b6c1fe> in <module>()
----> 1 error3() # function call

<ipython-input-41-8077a043d45e> in error3()
     14 # Raises an exception (multiple arguments supplied as a tuple)
     15 def error3():
---> 16         raise NetworkError, (1, 'Host not found')

NetworkError: (1, 'Host not found')
```

Assert

- used to introduce debugging code into the program.
- If the testlogic evaluates to False, assert raises an AssertionError exception with the optional data supplied.

```
Syntax:
    if not <some_test>:
        raise AssertionError(<message>)

    (or)

    assert <some_test>, <message>
```

In [35]:
```
# Ex1:
x = 5
y = 3
assert x<y, "x has to be smaller than y"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-35-9bec1dfa8fff> in <module>()
      2 x = 5
      3 y = 3
----> 4 assert x<y, "x has to be smaller than y"

AssertionError: x has to be smaller than y
```

In [36]:
```
# Ex2:
def writeData(file, data):
    try:
        assert file, "writeData: file is None!"
    except AssertionError, ex:
        print "file not found"
```

In [37]:
```
writeDate(myFile.tsf, 'some Data')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-37-725e5443ebbe> in <module>()
----> 1 writeDate(myFile.tsf, 'some Data')

NameError: name 'writeDate' is not defined
```

Assertions are not checked when python runs in optimized mode (i.e., with -o option).

```
python -o scriptName.py
```

Hint: assert should not be used to catch programming errors like x / 0, because Python traps such programming errors itself!.

Assert should be used for trapping user-defined constraints!