Argelander-Institut für Astronomie

Python for Astronomers

Errors and Exceptions

Exercise

Create a module textstat that contains the functions

- openfile(filename, readwrite=False): opens the specified file (readonly or readwrite) and returns the open file object
- isopen(file): returns True or False respectively
- closefile(file): closes the file, if open
- wordcount(file): returns the number of words in the file object
- linecount(file): returns the number of lines in the file object
- charcount(file): returns the number of characters in the file object

The module should contain a main program that uses these functions to print some statistics of mobydick.txt

Write a script that imports textstat and prints the same statistics of mobydick.txt

Exercise

What happens if you want to work with a not-existing file?

- Remove or rename monydick.txt with

  ```
  > mv -i mobydick.txt mobydick.dat
  ```

- Execute the script with

  ```
  > python solution7.py
  ```

  and study the result.

- In an interactive Python session, import textstat and try to open notExisting.dat:

  ```
  >>> import textstat
  >>> textstat.openfile("nonExisting.dat")
  ```

  compare the result to the above result.

Various types of Errors

```
>>> open("nonExisting.dat")         # IOError
>>> import nonExisting              # ImportError
>>> a b                             # SyntaxError
>>> l = []; l[1]                    # IndexError
>>> d = {}; d['x']                  # KeyError
>>> "abc" + 2                       # TypeError
>>> import math; math.sqrt(-2)      # ValueError
>>> 10.**1000                       # OverflowError
>>> 1 / 0                           # ZeroDevisionError
```

See http://docs.python.org/library/exceptions.html for a complete list of (built-in) types of Errors

Handling Exceptions

```
>>> try:
...     x, y = 1, 0
...     z = x / y            # Exception raised here
...     print "Result is %f..." % z
... except:
...     print "Oops!"        # Exception caught here
...     print "Something went wrong..."
>>> print "...but life goes on"


>>> # Repeat with x, y = 1, 2
```

Handling Exceptions

```
>>> def div(x, y):
...    # No error handling here
...    z = x / y
...    print "%f / %f = %f" % (x, y, z)

>>> try:
...    div(1, 0)
... except:
...    print "Something wrong in function div"

>>> div(1, 0)     # Exception not caught
```

Handling Exceptions

```
>>> def catchTest(l):
...   try:
...     print l[0] / l[1]
...   except IndexError:
...     print "Caught IndexError"
...   except TypeError:
...     print "Caught TypeError"

>>> catchTest([])          # IndexError
>>> catchTest(["a", "b"])  # TypeError
>>> catchTest([1, 0])      # Not caught
```

Handling Exceptions

```
>>> def catchTest(l):
...   try:
...     print l[0] / l[1]
...   except IndexError:
...     print "Caught IndexError"
...   except TypeError:
...     print "Caught TypeError"
...   except:
...     # Catches any exceptions uncaught so far
...     print "Caught unexpected exception"

>>> catchTest([1, 0])         # "Unexpected"
```

Handling Exceptions

```
>>> def catchTest(l):
...   try:
...     res = l[0] / l[1]
...   except TypeError:
...     print "Caught TypeError"
...   except:
...     print "Caught unexpected exception"
...   else:
...     # Run if no error occured:
...     print "Result is %f" % res

>>> catchTest([1, 2])
```

Exceptions as objects

```
>>> try:
...   1 / 0
... except Exception, e: # Catches any exception
...   # e is now Exception object:
...   print e

>>> e.<TAB>
>>> e.args      # Present for any exception
>>> e.message   # Present for any exception
```

Exceptions as objects

```
>>> # Additional information may be available
>>> # for some types of exceptions:

>>> try:
...   open("notExisting.dat")
... except IOError, e:
...   print "Error for file %s:" % e.filename
...   print e.strerror
... except Exception, e:
...   print "Unexpected error:"
...   print e
```

Raising Exceptions

```
>>> def f(x):
...     # x must be non-negative:
...     if x < 0:
...         raise ValueError("negative argument in f")
...     # Continue with function f

>>> try:
...     f(-2)
... except ValueError, e:
...     print e
```

Raising Exceptions

```
>>> def div(x, y):
...   try:
...     z = x / y
...   except:
...     print "Error caught and raised again"
...     raise

>>> try:
...   div(1, 0)
... except:
...   print "Error also caught here"
```

User-defined Exceptions

With class-related techniques (derive a subclass from class Exception) it is easy to define own types of exceptions:

```
>>> class MyException(Exception):
...     pass

>>> try:
...     raise MyException("Just for fun")
... except MyException, e:
...     print e
```

The finally clause

```
>>> def raiseException(doRaise=True):
...   if doRaise:
...     raise Exception("Raised just for fun")

>>> try:
...   raiseException()
... except:
...   print "Exception caught"
... finally:
...   print "Executing finally clause"

>>> # Repeat with raiseException(False)
```

The finally clause

```
>>> # Usefull to release resources:
>>> import numpy
>>> a = numpy.arange(10000000)
>>> try:
...   doSomething(a)
... except:
...   print "Somethimg went wrong"
... else:
...   print "Everything worked fine"
... finally:
...   del a
```

The assert statement

```
>>> x = -1
>>> assert x > 0      # Raises AssertionError
>>> x = 1
>>> assert x > 0
>>> print __debug__   # True

> python -O
>>> print __debug__   # False
>>> # Repeat above tests
```