Scripting Programming Languages and their Applications

Jan Gaura

October 4, 2017

Exceptions

- everything what's wrong is an exception, e.g. concatenation of string and number

```
In [1]: 'Hello, World!' + 1
---------------------------------------------------------------
TypeError                 Traceback (most recent call last)

/home/user/<ipython console> in <module>()

TypeError: cannot concatenate 'str' and 'int' objects
```

Exception Handling I

- all the errors you made will result in an exception
- Python has exception handling build in
- Python uses try and except block to handle exception
- raise to generate exception

Syntax:

```
try:
    #some code that can raise an exception
    ...
except <exception>:
    #code to hadle exception
finally:
    #code to be executed in the end
```

Exception Handling II

- Python uses try and except block to handle exception
- raise to generate exception

```
In [2]: try:
   ...:     'Hello, World!' + 1
   ...: except TypeError:
   ...:     print "You cannot do that!"
   ...:
   ...:
You cannot do that!
```

Exception Handling III

- Python uses try and except block to handle exception
- raise to generate exception

```
In [8]: raise TypeError("I raised TypeError!")
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

/home/user/<ipython console> in <module>()

TypeError: I raised TypeError!
```

User Defined Exceptions I

- user defined exceptions should derive from Exception class
- do not use BaseException class

User Defined Exceptions II

```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

try:
    raise MyError(2*2)
except MyError as e:
    print 'My exception occurred, value:', e.value

My exception occurred, value: 4

raise MyError('oops!')

Traceback (most recent call last):
  File "ex_test.py", line 12, in <module>
    raise MyError('oops!')
__main__.MyError: 'oops!'
```

Exception Documentation

For more info on exceptions, see documentation at:
`http://docs.python.org/tutorial/errors.html`

I/O Functions

Let's read a file:

```
my_file = open("file.txt", "rt")

#print all lines
for line in my_file:
    print line

my_file.close()
```

I/O Functions

Let's write a file:

```
my_file = open("file.txt", "wt")

#print all lines
fleet = {'BS62': 'Pegasus', 'BS75': 'Galactica',
    'BS36': 'Valkyrie'}
for designated_no in fleet.keys():
    my_file.write(fleet[designated_no] + '\n')

my_file.close()
```

Don't forget about newline!

I/O Functions

What happen if file doesn't exist?

```
1    my_file = open("non_existing_file.txt", "rt")
```

```
IOError: [Errno 2] No such file
or directory: 'non_existing_file.txt'
```

I/O with Exception Handling

Reading file and handle exception.

```
try:
    my_file = open("non_existing_file.txt", "rt")
    try:
        for line in my_file:
            print line
    finally:
        my_file.close()
except IOError as e:
    #print what's wrong
    print e
```

I/O with Exception Handling

And a bit compicated soulution.

```
my_file = None
try:
    my_file = open("non_existing_file.txt", "rt")

    for line in my_file:
        print line

except IOError as e:
    print e
finally:
    if my_file is not None:
        my_file.close()
```

The Easy way with "with" stament

```
with open("test.py", 'rt') as my_file:
    for line in my_file:
        print line
```

- can be used for files, DB connections, or other resources
- no need to close resource after end of work
- just leave the with block