EXCEPTIONS

Exceptions occur when *exceptional* situations occur in your program. For example, what if you are going to read a file and the file does not exist? Or what if you accidentally deleted it when the program was running? Such situations are handled using **exceptions**.

Similarly, what if your program had some invalid statements? This is handled by Python which **raises** its hands and tells you there is an **error**.

Errors

Consider a simple `print` function call. What if we misspelt `print` as `Print`? Note the capitalization. In this case, Python *raises* a syntax error.

```
>>> Print "Hello World"
  File "<stdin>", line 1
    Print "Hello World"
                  ^
SyntaxError: invalid syntax
>>> print "Hello World"
Hello World
```

Observe that a SyntaxError is raised and also the location where the error was detected is printed. This is what an **error handler** for this error does.

Exceptions

We will **try** to read input from the user. Press ctrl-d and see what happens.

```
>>> s = raw_input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Python raises an error called EOFError which basically means it found an **end of file** symbol (which is represented by ctrl-d) when it did not expect to see it.

The with statement

Acquiring a resource in the try block and subsequently releasing the resource in the finally block is a common pattern. Hence, there is also a with statement that enables this to be done in a clean manner:

Save as exceptions_using_with.py:

```
with open("poem.txt") as f:
    for line in f:
```

```
    print line,
```

How It Works

The output should be same as the previous example. The difference here is that we are using the `open` function with the `with` statement - we leave the closing of the file to be done automatically by `with open`.

What happens behind the scenes is that there is a protocol used by the `with` statement. It fetches the object returned by the `open` statement, let's call it "the file" in this case.

It *always* calls the file.*enter* function before starting the block of code under it and *always* calls the file.*exit* after finishing the block of code.

So the code that we would have written in a `finally` block should be taken care of automatically by the *exit* method. This is what helps us to avoid having to use explicit `try..finally` statements repeatedly.

Source: http://www.swaroopch.com/notes/python/