Programming for Science
Workshop 8

This workshop gives you some practice with input and exceptions.

1. Following the sort of code discussed in lectures, write a program to ask the user numbers until the user types the string `"average"` (without quotes of course) at which point the program should calculate and print the average of the numbers that the user has typed. If the user types `"q"` or `"Q"` the program should exit.

Make sure that your code is robust to the sorts of errors and inputs that the user might make. (What should the program do if the user asks for the average of no numbers?) Think about how to structure your program; there are solutions using one or two `while` loops.

2. We have seen how to open a file and read lines from it. However, like input/output with users, opening files can be error prone (if the file isn't there or cannot be read, for example). It's therefore prudent to enclose the opening and reading of a file in a `try ... except` block. For example:

```
filename = 'not_there'
try:
    f = open(filename, 'r')
    lines = f.readlines()
    f.close()      # Good practice to close the file after we've
                   # finished with it
except:
    print 'Something wrong reading', filename
```

Investigate what happens if you try to open a file that doesn't exist or a directory. Write a program to ask the user for a filename and then find the longest word in the file, making sure that you catch and deal with possible exceptions gracefully.

3. In addition to catching exceptions, it is possible for your code to `raise` an exception. This is accomplished by the `raise` statement; for example:

```
raise ValueError
```

which raises a `ValueError` exception. The following code defines a `bomb` function which counts down a randomly chosen number of seconds before throwing a `RuntimeError` exception. (Runtime errors are used to signify that something went wrong that's not covered by another exception). On average, one time in 6 the bomb doesn't explode, but just prints a message and the function returns normally.

```
from time import sleep
from dice import roll

def bomb():
    """
    Raise an exception after a waiting a few seconds
    """
    wait = roll()
```

```
    for i in range(wait, 0, -1):
        print i
        sleep(1)

    if roll() == 1:
        print "Bomb didn't explode"
        return
   raise RuntimeError
```

Write a `try ... except` statement in which the `bomb` function is called and, if the exception is raised, print a message saying the bomb has exploded. Test your function.

When an exception is raised, some additional information can be provided. Thus the following code, to replace the `raise` statement above, supplies a message (a string) about what sort of bomb has exploded:

```
    message = choose(['jelly', 'gelignite', 'fruit', 'TNT', 'atom',
                      'bath']) + ' bomb'
    raise RuntimeError (message)
```

The details about the exception can be found in the `except` statement like this:

```
except  RuntimeError, details:
    print 'The', details, 'exploded'
```

Modify your code to print the details about the bomb.

4. One way of defining the factorial function $n!$ is as the product of the natural numbers from 1 to n:

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times \ldots \times n$$

The factorial of 0 is defined to be 1 (*i.e.,* $0! = 1$). The factorial function is undefined for negative integers and for all non-integer values.

Write a function that uses a `for` loop to calculate the factorial of a given natural number. Make it rase a `ValueError` exception when it receives an inappropriate argument (ie, not a positive integer). Provide an informative message about the error and test your function.

You can test whether a variable is an integer with the built-in function `isinstance()`. For example,

```
>>> isinstance(1, int)
True
>>> isinstance(1.2, int)
False
```

Write a test function for `factorial` that uses `assert` to test that `factorial` computes the right result for a few values (check the edge cases) and also tests that the correct exceptions are raised.

5. The `urllib` module allows you to open URLs (universal resource locators – web addresses) rather like files. So you can read the contents of the webpage `http://empslocal.ex.ac.uk/studyres/ECM1408/secret.html` with code like:

```
import urllib

url = 'http://empslocal.ex.ac.uk/studyres/ECM1408/secret.html'
connection = urllib.urlopen(url)
for line in connection.fp:
    print line.strip()
```

Try this and read the (not very) secret message.

In order to follow the instructions in the secret message, you will find the following helpful. The string method `str.split(sep)` splits the string `str` into strings separated by the string `sep`. So for example:

```
>>> s = 'Mary,Bob,John'
>>> s.split(',')
['Mary', 'Bob', 'John']
>>> s = "Red white and blue"
>>> s.split()
['Red', 'white', 'and', 'blue']
```

(With no arguments the default separator is whitespace.)

You may also find the `upper()` method useful. This converts returns an uppercase version of the string, for example:

```
>>> s = 'mixeD CaSe'
>>> t = s.upper()
>>> print t
MIXED CASE
>>>
```