Lecture IV
Files and Exceptions

- Files
  - File-like Objects
  - Opening, Reading, Writing and Seeking
  - Encoding Issues: Binary and Text
- Exceptions
  - Concept
  - Catching
  - Raising

python

Files-like Objects

- File access in Python is modeled after C++.
- Files are accessed through "file-like objects", which are very similar to streams in C++.
- Streams may be also used to access other resources, such as string buffers or network sockets.
- Due to Python's dynamic typing, most applications do not care what a stream is accessing.

Opening Files

- Opening a file in Python means creating a file object using the `open()` or `file()` functions.
- Examples:

  -
    ```
    f1 = open('C:/Music/Playlists/favourite.m3u')
    f3 = open('my_script.py', 'w')
    f2 = open('todo_list.txt', 'r')
    f4 = open('../../index.htm', 'a')
    f5 = file('downloads/book.txt', 'r+')
    ```
- Files are closed when a file object goes out of scope, but can also be closed manually.

Reading Files

- File-like objects have several functions for reading data:
  - `f1 = open('C:/Music/Playlists/favourite.m3u')`
    `print f.read()`
  - `f1 = open('C:/Music/Playlists/favourite.m3u')`
    `print f.read(10)`
  - `f1 = open('C:/Music/Playlists/favourite.m3u')`
    `print f.readlines()`
  - `f1 = open('C:/Music/Playlists/favourite.m3u')`
    `print f.readline()`

Iterating over Files

- One can iterate over the lines of a file-like objects in a for loop:

  -
    ```
    f = open('C:/Music/Playlists/favourite.m3u')
    i = 0
    for line in f:
        print '%-4d %s' % (i, line)
        i += 1
    ```

Writing to Files

- Writing to files is similar to reading and is done using the `write()` and `writelines()` function of the file object, or by using a `print` statement:

  -
    ```
    f = open('C:/Music/Playlists/favourite.m3u', 'w')
    f.write('Evanescence - My Immortal.mp3\n')
    f.writelines(['Deep Purple - Soldier of
      Fortune.mp3',
                  'Orthodox Celts – Fields of Athenry',
                  'Jonathan Coulton – Still Alive'])
      print >>f, 'Ralph McTell - Streets of London'
    ```

Seeking Files

- When reading files, one sometimes wants to skip to a particular point in the file. This can be done using the `seek()` function:

  -
    ```
    f = open('C:/Music/Playlists/favourite.m3u')
    print f.seek(5)
    print f.read(10)
    print f.seek(5)
    print f.read(10)
    print f.seek(5, 1)
    print f.read(10)
    ```

Exceptions

- Exceptions are a mechanism for handling exceptional occurrences, usually errors.
- When an error or exceptional situation occurs, an exception is "thrown" or "raised".
- When a piece of code might produce an error, it should be surrounded by a "try" block. If no errors occur, nothing happens. However, if an error does occur, the programmer can "catch" it and specify what to do about it.
- An uncaught exception results in a crash.

Catching Exceptions

- All errors in Python and its libraries result in thrown exceptions, so it is extremely important for a coder to know how to catch them, more so than knowing how to throw or raise them.
- Catching is done using a try block. The error-prone code goes in the try part and the error-handling code goes into the except block.
- An except block can either catch any exception, or a particular type, such as a divide-by-zero, or a missing-file error.

Catching Exceptions

- Examples:

  -
    ```
    try:
        f = open('a_file_that_might_not_exist.txt')
        print f.read()
    except:
        print 'Could not open file!'
    print 'Execution continued.'
    ```

Catching Exceptions

- Examples:

  -
    ```
    try:
        f = open('a_file_that_might_not_exist.txt')
        print f.read()
    except IOError:
        print 'Could not open file!'
    print 'Execution continued.'
    ```

Catching Exceptions

- Examples:

  -
    ```
    try:
        f = open('a_file_that_might_not_exist.txt')
        print f.read()
    except IOError, e:
        print e
    print 'Execution continued.'
    ```

Catching Exceptions

- Examples:

  -
    ```
    try:
        f = open('a_file_that_might_not_exist.txt')
    except IOError, e:
        print e
    else:
        print f.read()
    print 'Execution continued.'
    ```

Catching Exceptions

- Examples:

  -
    ```
    try:
        x = 5
        y = 0
        z = x / y
    except IOError, e:
        print e
    ```

Catching Exceptions

- Examples:
  - try:

```
try:
    x = 5
    y = 0
    z = x / y
except IOError, e:
    print 'IO:', e
except ZeroDivisionError, e:
    print 'Zero:', e
except:
    print 'Some other error:', e
```

python

Catching Exceptions

- Examples:
  -
    ```
    f = open('output.txt', 'w')
    try:
        s = f.read()
        x = some_function(s)
    except Exception, e:
        print e
    finally:
        f.close()
        print 'File closed.'
    ```

Catching Exceptions

- When an exception is thrown but not caught, it will "propagate up the call stack", or in other words, it will look for a try block in the function that called the current one, then the one that called that one, and so on.

Catching Exceptions

- Example:

  -
    ```
    def f():
        x = 5 / 0
    def g():
        f()
    def h():
        try:
            g()
        except IOError:
            print 'IO Error!'
    def j():
        try:
            h()
        except Exception, e:
            print 'Unknown Error:', e
    j()
    ```

python

Catching Exceptions

- Example:

  -
    ```
    def f():
        open('hello.txt')
    def g():
        f()
    def h():
        try:
            g()
        except IOError:
            print 'IO Error!'
    def j():
        try:
            h()
        except Exception, e:
            print 'Unknown Error:', e
    j()
    ```

Raising Exceptions

- When writing larger scripts or libraries, one often wants to raise their own exceptions. This is done using the raise keyword. Example:

  -
    ```
    def square_root(x):
      if x < 0:
        raise Exception('No square roots for
      negatives.')
      import math
      return math.sqrt(x)
    ```