Files, os module and Comprehensions

Files

What is a file?

- A file is a collection of data that is stored on secondary storage like a disk or a thumb drive
- accessing a file means establishing a connection between the file and the program and moving data between the two

Two types of files

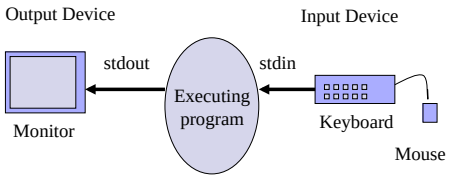
Files come in two general types:

- text files. Files where control characters such as “/n” are translated. This are generally human readable
- binary files. All the information is taken directly without translattion. Not readable and contains non-readable info.

File Objects or stream

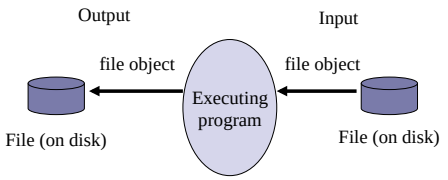
- When opening a file, you create a file object or file stream that is a connection between the file information on disk and the program.
- The stream contains a “buffer” of the information from the file, and provides the information to the program

Streams



Streams are objects with names such as stdin, stdout, stderr. They connect a device to a program and temporarily store data. They buffer it.

File Streams



We create a file stream as a connection to a file, a pipe between the file and the program. We can name this stream and use it to get file contents and, in general, manipulate the file.

Buffering

- Reading from a disk is very slow. Thus the computer will read a lot of data from a file in the hopes that, if you need the data in the future, it will be "buffered" in the file object.
- This means that the file object contains a **copy** of information from the file called a **cache** (pronounced "cash")

Making a file object

```
myFile = open("myFile.txt", "r")
```

`myFile` is the file object. It contains the buffer of information. The `open` function creates the connection between the disk file and the file object. The first quoted string is the file name on disk, the second is the mode to open it (here, `"r"` means to read)

Where is the disk file?

When opened, the name of the file can come in one of two forms:

- "file.txt" assumes the file name is file.txt and it is located in the current program directory
- "c:\bill\file.txt" is the fully qualified file name and includes the directory information

Different modes

- "r" is to read as a text file
- "w" is to write as a text file. Wipes the contents of the file if there is any
- "a" is append, adds to the end of a file
- "b" is a modifier, indicating a binary file. No character translation is done.
- "+" is a modifier, indicating both read and write. With "r", file must exists. With "w", makes or truncates the file, with "a" appends to the file

Text files use strings

- If you are interacting with text files (which is all we will do for this semester), remember that everything is a string
  - everything read is a string
  - if you write to a file, you can only write a string

Getting file contents

Once you have a file object:

- fileObject.read()
  - Reads the entire contents of the file as a string and returns it. It can take an optional argument integer to limit the read to N bytes, that is fileObject.read(N)
- fileObject.readline()
  - Delivers the next line as a string.

more file contents

- fileObject.readLines()
  - Returns a single list of all the lines from the file
- for line in fileObject:
  - iterator to go through the lines of a file

closing

When done, you close the file. Closing is important because the information in the fileObject buffer is "flushed" out of the buffer and into the file on disk, making sure that no information is lost.

- fileObject.close()

shortcut for reading a file

```
for line in file("fileToRead.txt"):
    print line
```

- file is automatically opened (by file( ))
- file is automatically closed at the end of the for loop
- defaults are read and text

writing

Once opened, you can write to a file (if the mode is appropriate):

- fileObject.write(s) Writes the string s to the file
- fileObject.writelines(list) write a list of strings (one at a time) to the file

Catching errors

- Your program should behave gracefully if the file can't be opened
- In later chapters we will describe "exception", but for now we will just assume that you can get the file.

Example, simpleFiles

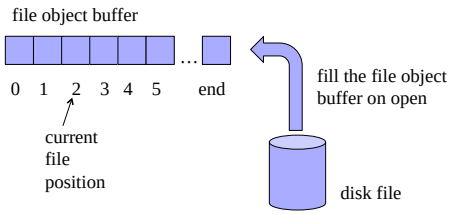
The current file position

- Every file maintains a “current file position”.
- It is the current position in the file, and indicates what the file will read next
- Is set by the mode table above

Remember the file object buffer

- When the disk file is opened, the contents of the file are copied into the buffer of the file object
- Think of the file object as a very big list, where every index is one of the pieces of information of the file
- The current position is the present index in that list

current file position



the tell() method

- The tell() method tells you the current file position
- The positions are in bytes (think characters for ASCII) from the beginning of the file

fd.tell() => 42L

the seek() method

- the seek() method updates the current file position to where you like (in bytes offset from the beginning of the file)
- fd.seek(0) # to the beginning of the file
- fd.seek(100) # 100 bytes from beginning

counting bytes is a pain

- counting bytes is a pain in the a$$
- seek has an optional argument set:
  - 0: count from the beginning
  - 1: count for the current file position
  - 2: count from the end (backwards)

example

```
fd = open('file.txt','r')
lst = fd.read()   # all contents in lst
                  # current position at end
fd.seek(0,0)      # current position to start
line = fd.readline() # read first line
fd.seek(-100,2)      # 100 bytes from end
line = fd.readline() # read last line (maybe)
```


every read moves current forward

- every read/readline/readlines moves the current pos forward
- when you hit the end, every read will just yield '', since you are at the end
- you need to seek to the beginning to start again (or close and open, seek is easier)

read(size=1)

- you can use the read() method to read just one byte at a time, and in combination with seek move around the file and "look for things". Once current is set, you can begin reading again

file seek example

os module

What is the os module

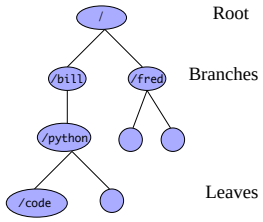
- The os module in Python is an interface between the operating system and the Python language.
- As such, it has many sub-functionalities dealing with various aspects.
- We will look mostly at the file related stuff

What is a directory/folder?

- Whether in Windows, Linux or on OS X, all OS's maintain a directory structure.
- A directory is a container of files/ otherDirectories
- These directories are arranged in a hierarchy or tree

Odd kind of tree

- odd kind of tree in CS
- it has a root node, with branch nodes, ends in leaf nodes
- it is upside down
- the directory structure is a tree



Directory tree

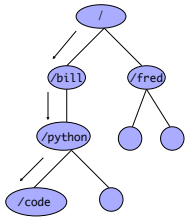
- Directories can be organized in a hierarchy, with the root directory and subsequent branch and leaf directories
- Each directory can hold files and 'point to' parent and children directories

A file path is a path through the tree

- A path to a file is a path through the hierarchy to the node that contains a file
- /bill/python/code/myCode.py
  - path is from the root node /, to the bill directory, to the python directory, to the code directory where the file myCode.py resides

the / in a path

- think of / as an operator, showing something is a directory
- follow the path, the leaf is either a directory or file



a path String

- a valid path string for python is a string which indicates a valid path in the directory structure
- Thus '/Users/bill/python/code.py' is a valid path string

different 'paths' for different os

- It turns out that each OS has its own way of specifying a path
  - C:\bill\python\myFile.py
  - /Users/bill/python/myFile.py
- Nicely, python knows that and translates to the appropriate OS
- Do it the second way, good for CSE 232

Two special directory names

- The directory name '.' is shortcut for the name of the current directory you are in as you traverse the directory tree
- The directory name '..' is a shortcut for the name of the parent directory of the current directory you are in

Some os commands

- os.getcwd(): Returns the full path of the current working directory
- os.chdir(pathString): Change the current directory to the path provided
- os.listdir(pathString): Return a list of the files and directories in the path (including '.')

Some more os commands

- os.rename(sourcePathStr, destPathStr): Renames a file or directory
- os.mkdir(pathStr): make a new directory. So os.mkdir('/Users/bill/python/new') creates the directory new under the directory python.
- os.remove(pathStr). Removes the file
- os.rmdir(pathStr). Removes the directory,but the directory must be empty

os.stat

Gives info on a file. Here are a few.

- statInfo = os.stat(pathStr)
  - statInfo.st_size (size in bytes)
  - statInfo.st_atime (last time of access)
  - statInfo.st_mtime (last modification time)
  - statInfo.st_uid (user id of the owner)

the walk function

- os.walk(pathStr): Starts at the directory in pathStr. It yields three values:
  - dirName, name of the current directory
  - dirList, list of subdirectories in the directory
  - files, list of files in the directory
- If you iterate through, walk will visit every directory in the tree. Default is topDown

glob package

- crazy word in CS to represent some simple pattern matching.
- Say you want to list all the files in a directory that end with .mp3
- Use the wildcard character '*', meaning match all characters, therefore '*.mp3'
- glob.glob("*.mp3")

file Search

os.path module

allows you to gather some info on a path's existence

- os.path.isfile(pathStr): is this a path to an existing file (T/F)
- os.path.isdir(pathStr): is this a path to an existing directory (T/F)
- os.path.exists(pathStr): the path (either as a file or directory) exists (T/F)

os.path names

for p= '/Users/bill/python/myFile.py'

- os.path.basename(p): returns 'myFile.py'
- os.path.dirname(p): returns '/Users/bill/python'
- os.path.split(p): returns ['Users/bill/python','myFile.py']
- os.path.splitext(p): returns ('/Users/bill/python/myFile', '.py')
- os.path.join(os.path.split(p)[0],'other.py'): returns '/Users/bill/python/other.py'

a more complex file search

What this does

- Prompts for:
  - root directory to start from
  - suffix of files you care about (such as .py)
  - a string you are looking for in a file
- It walks the directory from the root down. If a file has the right suffix, it looks to see if the file contains the string
- it remembers the files that had the string and the directories of those files.

reprompts the user if bad dir

- uses os.path.isdir(p) to test of path p is a valid dir
- if not, reprompts till the user gives a good one

split and join

- uses splitext to get the suffix of each file
- uses join to create a new path, combining the dirname and the filename

file read

- fileDescriptor.read() reads the entire contents of the file in as a string
- test to see if the target string is in the 'file as a string' variable
- Expensive, opens and closes a lot of files!

List Comprehensions

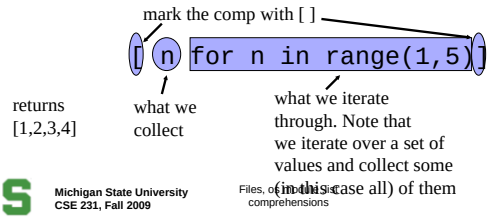
Lists are a big deal!

- The use of lists in Python is a major part of its power
- Lists are very useful and can be used to accomplish many tasks
- Therefore Python provides some pretty powerful support to make common list tasks easier

Constructing lists

One way is a “list comprehension”

```
[n for n in range(1,5)]
```



modifying what we collect

```
[ n**2 for n in range(1,6)]
```

returns [1,4,9,16,25]. Note that we can only change the values we are iterating over, in this case n

multiple collects

```
[x+y for x in range(1,4) for y in range (1,4)]
```

It is as if we had done the following:

```
myList = [ ]
for x in range (1,4):
  for y in range (1,4):
     myList.append(x+y)
```

returns [2,3,4,3,4,5,4,5,6]

modifying what gets collected

```
[c for c in “Hi There Mom” if c.isupper()]
```

The “if” part of the comprehensive controls which of the iterated values is collected at the end. Only those values which make the if part true will be collected

returns [‘H’,’T’,’M’]

Example, simpleComprehension