CS 112: Introduction to Programming:

File IO

File Processing

- Data can be read from and written to a file in addition to standard input (keyboard) and standard output (screen)
  - Create a file reference and designate the input/output (I/O) mode:
    - "r" → read mode
    - "w" → write mode
  - Use this file reference to process the lines of data into and out of the file using file I/O functions
  - Close the file (for reading/writing)

File Processing Sequence

1. Open the file
2. Read from the file
3. Close the file

In some cases, not properly closing a file could result in data loss.

File Processing

- Working with text files in Python
  - Associate a file with a reference variable using the open function
    <filevar> = open(<name>, <mode>)
  - **Name** is a string with the actual file name on the disk.
  - **Mode** is either 'r' or 'w' depending on whether we are reading or writing the file. 'a' for appending to an existing file. ('a' will also create a non-existent file)
  - ***infile = open("numbers.dat", "r")***

Reading Files

- <filevar>.***read()*** – returns the entire file as a string
- <filevar>.***read(x)*** – returns the next x characters from the file **(file position is remembererd)**
- <filevar>.***readline()*** – returns the next line of the file. All text up to ***and including*** the next newline character
- <filevar>.***readline(x)*** – returns the next x characters of the line. All text up to ***and including*** the next newline character
- <filevar>.***readlines()*** – returns a list of all lines in the file. Each list item is a single line including the newline characters.

File Processing: read

```
# printfile.py
#    Prints a file to the screen.

def main():
   fname = raw_input("Enter filename: ")
   infile = open(fname,'r')
   data = infile.read()
   print data

main()
```

- First, prompt the user for a file name
- Open the file for reading through the variable infile
- The file is read as one string and stored in the variable data

File Processing : readline

- readline can be used to read the next line from a file, including the trailing newline character

```
infile = open(someFile, 'r')
for i in range(5):
        line = infile.readline() # Read a single line
        print line[:-1] # Slice off the newline
```

- This reads the first 5 lines of a file
- Slicing is used to strip out the newline characters at the ends of the lines

File processing: readline(x)

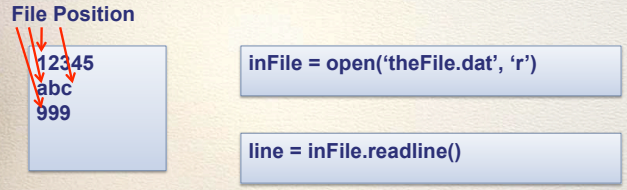
readlineExample.py – /Users/dfleck/Documents/gmuwebsite/classes/cs112/spring09/samplecode/file_i...

```
# Example file to show how readline(x) and read(x) work

def main():
    inputFile = open('input.dat', 'r')
    outStr = ''
    inStr = 'x'

    while inStr != '':
        inStr = inputFile.readline(10) # Reads the numbers
        outStr += inStr
        inStr = inputFile.read(1) # Reads the newline

    print '%s' %(outStr)

main()
```

Ln: 15 Col: 6

```
>>>
1234567890123456789012345678901234567890123456789012345678901234567890
>>>
```

File Position



inFile = open('theFile.dat', 'r')

line = inFile.readline()

line = inFile.read(2)
(pointing to newline)

line = inFile.read(1)
(pointing to newline)

File Processing: readlines

- Another way to loop through the contents of a file is to read it in with readlines and then loop through the resulting list.

```
infile = open(someFile, 'r')
for line in infile.readlines():
    # Line processing here
infile.close()
```


File Processing: easiest way!

- Python treats the file itself as a sequence of lines!
-
```
infile = open(someFile, 'r')
for line in infile:
    # process the line here
infile.close()
```


File Processing: writing

- Two basic functions for writing data in text file mode:
- ***write(x)*** – writes te string x to text file
- ***writelines(x)*** – writes strings in list x to text file

File Processing: writing

- Opening a file for writing prepares the file to receive data
- If you open an existing file for writing, you wipe out the file's contents. If the named file does not exist, a new one is created.
- Outfile = open("mydata.out", 'w')
- <filevar>.write(<string>)

Warning: If you open an existing file for writing you DELETE EXISTING CONTENT of the file!!

File Processing : Writing

```
outfile = open("example.out", 'w')
count = 1
outfile.write("This is the first line\n")
count = count + 1
outfile.write("This is line number %d" % (count))
outfile.close()
```

- If you want to output something that is not a string you need to convert it first. Using the string formatting operators are an easy way to do this.

```
This is the first line
This is line number 2
```


Example Program: Batch Usernames

- *Batch* mode processing is where program input and output are done through files (the program is not designed to be interactive)
- Let's create usernames for a computer system where the first and last names come from an input file.

Example Program: Batch Usernames

```
# userfile.py
#    Program to create a file of usernames in batch mode.

import string

def main():
    print "This program creates a file of usernames from a"
    print "file of names."

    # get the file names
    infileName = raw_input("What file are the names in? ")
    outfileName = raw_input("What file should the usernames go in? ")

    # open the files
    infile = open(infileName, 'r')
    outfile = open(outfileName, 'w')
```

Example Program: Batch Usernames

```
# process each line of the input file
  for line in infile:
     # get the first and last names from line
     last, first = string.split(line, ",") # Split the names on comma
     # create a username
     uname = string.lower(first[0]+last[:7])
     # write it to the output file
     outfile.write(uname+'\n')

  # close both files
  infile.close()
  outfile.close()

  print "Usernames have been written to", outfileName
```


Example Program: Batch Usernames

- Things to note:
  - It's not unusual for programs to have multiple files open for reading and writing at the same time.
  - The lower function is used to convert the names into all lower case, in the event the names are mixed upper and lower case.
  - We need to concatenate '\n' to our output to the file, otherwise the user names would be all run together on one line.

Basic File I/O Modes

- Three basic modes
  - **"r" – read from file**
    - If file doesn't exist → error
  - **"w" – write to file**
    - If file doesn't exist → file created
    - If file does exist → content overwritten
  - **"a" – append to file**
    - If file doesn't exist → file created
    - If file does exist → new data appended

Basic File I/O Modes

- Three mixed modes (read/write)
  - **"r+" – read from and write to file**
    - If file doesn't exist → error
  - **"w+" – write to and read from file**
    - If file doesn't exist → file created
    - If file does exist → content overwritten
  - **"a+" – append to and read from file**
    - If file doesn't exist → file created
    - If file does exist → new data appended

File Seek and tell

- Lots of other File methods you can use:
  - http://docs.python.org/library/stdtypes.html#bltin-file-objects
- <filevar>.tell() – returns the current position in the file
- <filevar>.seek(offset [, whence]) -- set the file's current position
  - whence = 0, set absolute position starting from beginning of file
  - whence = 1, set relative position starting at current file position
  - whence = 2, set absolute position from end of the file
- Lets see an example: fileSeekAndTell.py

Where to find information

Remember:

- Language reference – describes specific syntax about Python. Use this if you are writing a Python interpreter (rarely used)
- Library reference – describes modules, built-in functions, data types, etc… used frequently!