Special Topics in Computer Science- CSC 4992

Files

Text Files

- A text file is logically a sequence of characters
- Basic operations are
  - input (read characters from the file)
  - output (write characters to the file)
- There are several flavors of each type of operation

File Input

- We want to bring text in from a file for processing

- Three steps:
  - Open the file for input
  - Read the text and save it in a variable
  - Process the text

Opening a File

```
<a variable> = open(<a file name>, <a flag>)
```

<a flag> can be

- **'r'** - used for input, to *read* from an existing file
- **'w'** - used for output, to *overwrite* an existing file
- **'a'** - used for output, to *append* to an existing file

Opening and Closing Files in Python

Method Name	Use	Explanation
open	`open(filename,'r')`	Open a file called `filename` and use it for reading. This will return a reference to a file object.
open	`open(filename,'w')`	Open a file called `filename` and use it for writing. This will also return a reference to a file object.
close	`filevariable.close()`	File use is complete.

```
>>>fileref = open("rainfall.txt","r")
>>>
>>>fileref.close()
>>>
```

Using Files for Large Data Sets (cont'd.)

- We can read such data from the file and fill our collections for later processing.
- In Python, we must open files before we can use them and close them when we are done with them.
- Python has functions that can be used to open and close files.
- Once a file has been opened it becomes a Python object just like all other data.
- When we are finished with the file, we can close it by using the close method.
- After the file is closed, any further attempts to use file reference will result in an error.

Read Lines of Text from a File

```
filename = input('Enter a file name: ')
myfile = open(filename, 'r')

for line in myfile:
    print(line)
```

The variable **`line`** picks up the next line of text on each pass through the loop

`line` will contain the newline character, so the echo will print extra blank lines

Read Lines of Text from a File

```
filename = input('Enter a file name: ')
myfile = open(filename, 'r')

for line in myfile:
    print(line[:-1])
```

Extract a substring up to but not including the last character (the newline)

This will produce an exact echo of the input file

Iterating Over Lines in a File Example

- Suppose we have a text file called rainfall.txt that contains the data below representing the total annual rainfall (in inches) for 25 towns in Iowa.

```
Akron 25.81
Albia 37.65
Algona 30.69
Allison 33.64
Alton 27.43
AmesW 34.07
AmesSE 33.95
Anamosa 35.33
Ankeny 33.38
Atlantic 34.77
Audubon 33.41
Beaconsfield 35.27
Bedford 36.35
BellePlaine 35.81
```

```
Bellevue 34.35
Blockton 36.28
Bloomfield 38.02
Boone 36.30
Brighton 33.59
Britt 31.54
Buckeye 33.66
BurlingtonKBUR 37.94
Burlington 36.94
Carroll 33.33
Cascade 33.48
```

Iterating Over Lines in a File (cont'd.)

- Although it would be possible to consider entering this data by hand each time it is used, you can imagine that such a task would be time-consuming and error-prone.
- In addition, it is likely that there could be data from many more towns.
- We can use this file as input in a program that will do some data processing.

Iterating Over Lines in a File (cont'd.)

- As the *for* loop iterates through each line of the file, the loop variable will contain the current line of the file as a string of characters.

```
for line in myFile:
    statement1
    statement2
    ...
```

```
rainfile = open("rainfall.txt","r")

for aline in rainfile:
    values = aline.split()
    print(values[0], "had",values[1],"inches of rain.")

rainfile.close()
```

```
Akron had 25.81 inches of rain.
Albia had 37.65 inches of rain.
Algona had 30.69 inches of rain.
Allison had 33.64 inches of rain.
Alton had 27.43 inches of rain.
AmesW had 34.07 inches of rain.
AmesSE had 33.95 inches of rain.
Anamosa had 35.33 inches of rain.
Ankeny had 33.38 inches of rain.
Atlantic had 34.77 inches of rain.
Audubon had 33.41 inches of rain.
```

```
Beaconsfield had 35.27 inches of rain.
Bedford had 36.35 inches of rain.
BellePlaine had 35.81 inches of rain.
Bellevue had 34.35 inches of rain.
Blockton had 36.28 inches of rain.
Bloomfield had 38.02 inches of rain.
Boone had 36.30 inches of rain.
Brighton had 33.59 inches of rain.
Britt had 31.54 inches of rain.
Buckeye had 33.66 inches of rain.
BurlingtonKBUR had 37.94 inches of rain.
Burlington had 36.94 inches of rain.
Carroll had 33.33 inches of rain.
Cascade had 33.48 inches of rain.
```

Alternative File-Reading Methods

- In addition to the *for loop*, *Python provides three methods to read data from the input file.*
- The *readline* method reads one line from the file and returns it as a string.
- The string returned by readline will contain the newline character at the end.
- This method returns the empty string when it reaches the end of the file.

Alternatively, Use **`readline`**

```
filename = input('Enter a file name: ')
myfile = open(filename, 'r')

while True:
    line = myfile.readline()
    if line == '':
        break
    print(line[:-1])
```

The **`readline`** method reads a line of text and returns it as a string, including the newline character

This method returns the empty string if the end of file is encountered

Alternative File-Reading Methods (cont'd.)

- The *readlines* method returns the contents of the entire file as a *list* of strings, where each item in the list represents one line of the file.
- It is also possible to read the entire file into a single string with *read*.

Example: Read Text from a File

```
filename = input('Enter a file name: ')

myfile = open(filename, 'r')

text = myfile.read()

print(text)
```

The file name must either be in the current directory or be a pathname to a file in a directory

Python raises an error if the file is not found

`text` refers to one big string

Method Name	Use	Explanation
`write`	`filevar.write(astring)`	Adds a string to the end of the file. `filevar` must refer to a file that has been opened for writing.
`read(n)`	`filevar.read()`	Reads and returns a string of `n` characters, or the entire file as a single string if `n` is not provided.
`readline(n)`	`filevar.readline()`	Returns the next line of the file with all text up to and including the newline character. If `n` is provided as a parameter, then only `n` characters will be returned if the line is longer than `n`.

Alternative File-Reading Methods (cont'd.)

- Note that we need to reopen the file before each read so that we start from the beginning.
- Each file has a marker that denotes the current read position in the file.
- Anytime one of the read methods is called, the marker is moved to the character immediately following the last character returned.
- In the case of *readline*, this moves the marker to the first character of the next line in the file.
- In the case of read or *readlines*, the marker is moved to the end of the file.

File Processing Example

- A text file can be thought of as a sequence of lines

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500To: source@collab.sakaiproject.orgFrom: stephen.marquard@uct.ac.zaSubject: [sakai] svn commit: r39772 - content/branches/Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772

File Processing

- A text file has newlines at the end of each line

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008\n
Return-Path: <postmaster@collab.sakaiproject.org>\n
Date: Sat, 5 Jan 2008 09:12:18 -0500\nTo:
source@collab.sakaiproject.org\nFrom: stephen.marquard@uct.ac.za\nSubject:
[sakai] svn commit: r39772 - content/branches/\nDetails:
http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n

Reading the *Whole* File

- We can read the whole file (newlines and all) into a single string.

```
fhand = open('mbox-short.txt', 'r')
inp = fhand.read()
print (len(inp)) #94626
print (inp[:20])  #From stephen.marquar
```

Searching Through a File

- We can put an if statement in our for loop to only print lines that meet some criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:') :
        print (line)
```

OOPS!

What are all these blank lines doing here?

From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu

...

OOPS!

What are all these blank lines doing here?

Each line from the file has a newline at the end.

The print statement adds a newline to each line.

From: stephen.marquard@uct.ac.za\n
\n
From: louis@media.berkeley.edu\n
\n
From: zqian@umich.edu\n
\n
From: rjlowe@iupui.edu\n
\n
...

Searching Through a File (fixed)

- We can strip the whitespace from the right hand side of the string using rstrip() from the string library
- The newline is considered "white space" and is stripped

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print (line)
```

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....

Skipping with continue

- We can conveniently skip a line by using the continue statement

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:') :
        continue
    print (line)
```

Using in to select lines

- We can look for a string anywhere in a line as our selection criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line :
        continue
    print (line)
```

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using –f
From: stephen.marquard@uct.ac.zaAuthor: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan 4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...

Prompt for File Name

```
fname = raw_input('Enter the file name:  ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print (“There were', count, 'subject lines in”, fname)
```

Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

Enter the file name: mbox-short.txt
There were 27 subject lines in mbox-short.txt

Bad File Names

```
fname = raw_input('Enter the file name:  ')
try:
    fhand = open(fname)
except:
    print 'File cannot be opened:', fname
    exit()
count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print 'There were', count, 'subject lines in', fname
```

Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

Enter the file name: na na boo boo
File cannot be opened: na na boo boo

Count the Words

```
filename = input('Enter a file name: ')
myfile = open(filename, 'r')
wordcount = 0

for line in myfile:
    wordcount += len(line.split())

print('The word count is', wordcount)
```

File Output

- We want to save data to a text file

- Four steps:
  - Open the file for output
  - Covert the data to strings, if necessary
  - Write the strings to the file
  - Close the file

Writing a File

- Let's think about another example of file processing: converting our file from data about rainfall in inches to data about rainfall in centimeters.
- This will require that we read the file contents as before, but, instead of printing a message, we will do some computation and then write the results back to another file.

Writing a File (cont'd.)

```
rainfile = open("rainfall.txt","r")
outfile = open("rainfallInCM.txt","w")

for aline in rainfile:
    values = aline.split()

    inches = float(values[1])
    cm = 2.54 * inches

    outfile.write(values[0]+" "+str(cm)+"\n")

rainfile.close()
outfile.close()
```

Writing a File (cont'd.)

- The write statement does all of the work to create a new line in the output file.
- Note that it can add only a single string to the file each time it is used.
- For this reason, we need to use string concatenation to build up the line piece by piece.
- The entire line is completed by adding a newline character.

Example: Write Text to a File

```
filename = input('Enter a file name: ')

myfile = open(filename, 'w')

myfile.write('Two lines\nof text')

myfile.close()
```

If the file already exists, it is overwritten; otherwise, it is created in the current directory or path

The **`write`** method expects a string as an argument

Failure to close the file can result in losing data

Example: Write Integers to a File

```
filename = input('Enter a file name: ')

myfile = open(filename, 'w')

for i in range(1, 11):
    myfile.write(str(i) + '\n')

myfile.close()
```

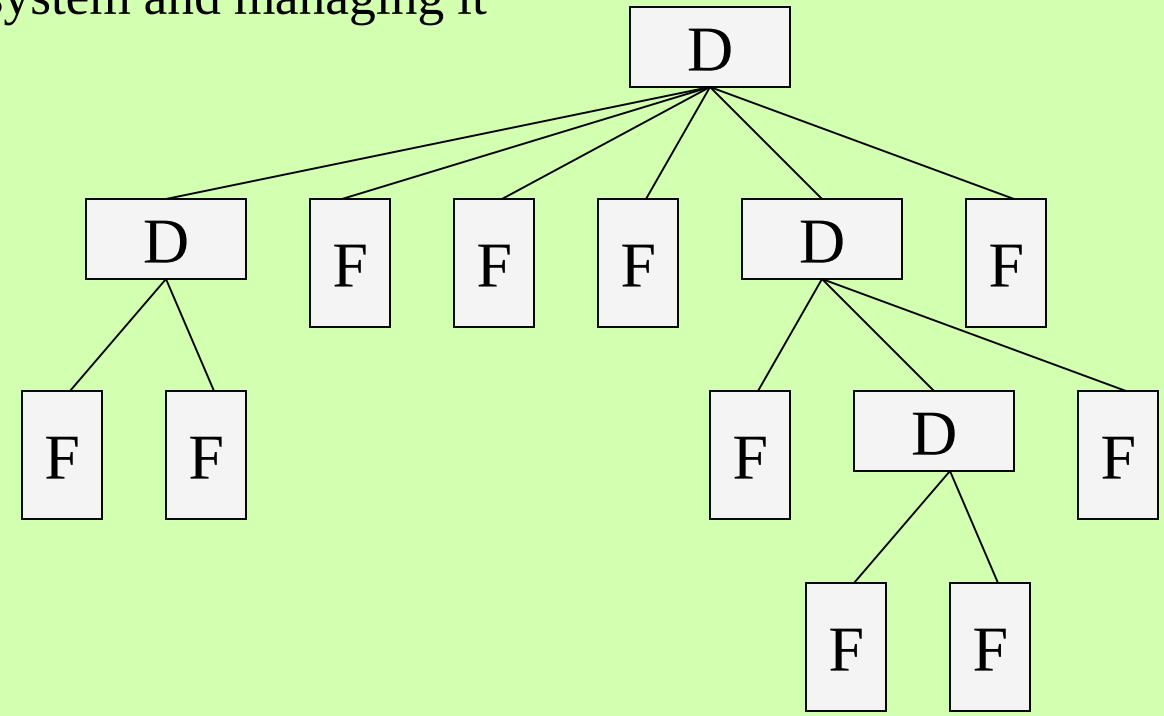
`write` can be called 0 or more times

Data values must be converted to strings before output

Separators, such as newlines or spaces, must be explicitly written as well

Managing Directories

- Directories are organized in a tree-like structure
- Python's **`os`** module includes many functions for navigating through a directory system and managing it



`os` Functions

Function	What It Does
`os.getcwd()`	Returns the current working directory (string)
`os.chdir(path)`	Attaches to the directory specified by path
`os.listdir(path)`	Returns a list of the directory's contents
`os.remove(name)`	Removes a file at path
`os.rename(old, new)`	Resets the old path to the new one
`os.removedirs(path)`	Removes the directory (and all subdirectories) at path

`os.path` Functions

Function	What It Does
`os.path.exists(path)`	Returns **`True`** if **`path`** exists or **`False`** otherwise.
`os.path.isdir(path)`	Returns **`True`** if **`path`** names a directory or **`False`** otherwise.
`os.path.isfile(path)`	Returns **`True`** if **`path`** names a file or **`False`** otherwise.
`os.path.getsize(path)`	Returns the size of the object named by **`path`** in bytes.

Example: Does the File Exist?

```
import os.path

filename = input('Enter a file name: ')

if not os.path.exists(filename):
    print('Error: the file not not exist!')
else:
    myfile = open(filename, 'r')
    print(myfile.read())
    myfile.close()
```

String Formatting

- Converting variables to strings and concatenating these strings together can be a tedious process.
- Python provides us with a better alternative: formatted strings.
- For example:

  print (values [0] , "had" , values [1] ,"inches of rain.")

  Using a formatted string, we write the previous statement as

  print("%s had %d inches of rain" % (values [0] ,values[1]))

String Formatting (cont'd.)

- The % operator is a string operator called the *format operator*.
- The left side of the expression holds the template or format string, and the right side holds a collection of values that will be substituted into the format string.
- The number of values in the collection on the right side corresponds with the number of % characters in the format string.
- Values are taken-in order-from the collection and inserted into the format string.

String Formatting (cont'd.)

Character	Output Format
d,i	Integer or long integer
u	Unsigned integer
f	Floating point as m.ddddd
e	Floating point as m.dddddde+/-xx
E	Floating point as m.dddddE+/-xx
g	Use %e for exponents less than −4 or greater than +5, otherwise use %f
c	Single character
s	String, or any python data object that can be converted to a string by using the str function.
%	Insert a literal % character

String Formatting (cont'd.)

Modifier	Example	Description
number	%20d	Put the value in a field width of 20
-	%-20d	Put the value in a field 20 characters wide, left-justified
+	%+20d	Put the value in a field 20 characters wide, right-justified
0	%020d	Put the value in a field 20 characters wide, fill in with leading zeros.
.	%20.2f	Put the value in a field 20 characters wide with 2 characters to the right of the decimal point.
(name)	%(name)d	Get the value from the supplied dictionary using name as the key.

String Formatting (cont'd.)

```
>>> a = 10
>>> b = 'apple'
>>> print("The %s costs %d cents" % (b,a))
The apple costs 10 cents
>>> myStr = "The %+15s costs %4.1d cents" % (b,a)
>>> myStr
'The           apple costs   10 cents'
>>> myStr = "The %+15s costs %6.1f cents" % (b,a)
>>> myStr
'The           apple costs   10.0 cents'
>>> myDict = { 'name':'apple', 'cost':10, 'price':15}
>>> print("The %(name)s costs %(price)5.1f cents" % myDict)
The apple costs  15.0 cents
```