CMSC201
Computer Science I for Majors

Lecture 11 – File I/O (Continued)

Prof. Katherine Gibson

Last Class We Covered

- Escape sequences
  - Uses a backslash (\)
- File I/O
  - Input/Output
  - How to open a file
    - For reading or writing
  - How to read lines from a file

Any Questions from Last Time?

Today's Objectives

- To review how to open and read from a file
- To learn how to use the `split()` function
  - To break a string into tokens
  - And to learn the `join()` function
- To get more practice with File I/O
- To cover the different ways to write to a file
- To learn how to close a file

Review from Last Class

Using **open()**

- Which of these are valid uses of **open()**?

```
1.myFile   = open(12, "r")
2.fileObj  = open("HELLO.txt")
3.writeTo  = open(fileName, "w")
4."file"   = open("test.dat")
5.theFile  = open("file.dat", "a")
```

Using **open()**

- Which of these are valid uses of **open()**?

not a valid string

```
✗1. myFile   = open(12, "r")
✓2. fileObj  = open("HELLO.txt")
✓3. writeTo  = open(fileName, "w")
✗4. "file"   = open("test.dat", "R")
✓5. theFile  = open("fil
```

not a valid filename

uppercase “**R**” is not a valid access mode

Three Ways to Read a File

- Write the code that will perform each of these actions using a file object called **`aFile`**

1. Read the whole file in as one big long string
2. Read the first line of the file
3. Read the file in as a list of strings (each is one line)

Three Ways to Read a File

- Write the code that will perform each of these actions using a file object called **`aFile`**

1. Read the whole file in as one big long string

```
bigString  = aFile.read()
```

2. Read the first line of the file

```
firstLine  = aFile.readline()
```

3. Read the file in as a list of strings (each is one line)

```
stringList = aFile.readlines()
```

Whitespace

- There are two ways we know of to remove whitespace from a string

- Slicing can be used to remove just the newline at the end of a line that we have read in from a file:

```
myLineWithoutNewline = myLine[:-1]
```

- The **`strip()`** function removes all leading and trailing whitespace (tabs, spaces, newlines) from a string

```
withoutWhitespace = myLine.strip()
```

Using `for` Loops to Read in Files

- Remember, **`for`** loops are great for iterating!

- With a list, the **`for`** loop iterates over…
  - Each element of the list (in order)
- Using a **`range()`**, the **`for`** loop iterates over…
  - Each number generated by the range (in order)
- And with a file, the **`for`** loop iterates over…
  - Each line of the file (in order)

String Splitting

String Splitting

- We can break a string into individual pieces
  - That you can then loop over!

- The function is called **`split()`**, and it has two ways it can be used:
  - Break the string up by its whitespace
  - Break the string up by a specific character

Splitting by Whitespace

- Calling **`split()`** with no arguments will remove all of the whitespace in a string
  - Even the “inside” whitespace

```
>>> line = "hello world this is my song\n"
>>> line.split()
['hello', 'world', 'this', 'is', 'my', 'song']

>>> whiteCat = "\t\nI   love\t\t\nwhitespace\n  "
>>> whiteCat.split()
['I', 'love', 'whitespace']
```

Splitting by Specific Character

- Calling **`split()`** with a string in it, we can remove a specific character (or more than one)

these character(s) are called the delimiter

```
>>> commas = "once,twice,thrice"
>>> commas.split(",")
['once', 'twice', 'thrice']

>>> double = "hello how ill are all of your llamas?"
>>> double.split("ll")
['he', 'o how i', ' are a', ' of your ', 'amas?']
```

Splitting by Specific Character

- Calling **`split()`** with a string in it, we can remove a specific character (or more than one)

```
>>> commas = "once,twice,thrice"
>>> commas.split(",")
['once', 'twice', 'thrice']
```

these character(s) are called the delimiter

```
>>> double = "hello how ill are all of your llamas?"
>>> double.split("ll")
['he', 'o how i', ' are a', ' of your ', 'amas?']
```

notice that it didn't remove the whitespace

Practice: Splitting

- Use **`split()`** to solve the following problems

- Split this string on all of its whitespace:

```
daft = "around the \nworld"
```

- Split this string on the double t’s (**`tt`**):

```
doubleT = "nutty otters making lattes"
```

Practice: Splitting

- Use **`split()`** to solve the following problems

- Split this string on all of its whitespace:

```
daft = "around the \nworld"
daft.split()
```

- Split this string on the double t's (**`tt`**):

```
doubleT = "nutty otters making lattes"
doubleT.split("tt")
```

Looping over Split Strings

- Splitting a string creates a list of smaller strings
- Using a **`for`** loop with a split string, we can iterate over each word (or token) in the string
- Syntax:

```
for piece in myString.split():
    # do something with each piece
```

Example: Looping over Split Strings

```
>>> double = "hello how ill are all of your llamas?"
>>> for token in double.split("ll"):
...     print("y" + token + "y")
...
yhey
yo how iy
y are ay
y of your y
yamas?y
```

append a "y" to the front and end of each list element, then print

remember, **`double.split("ll")`** makes the list **`['he', 'o how i', ' are a', ' of your ', 'amas?']`**

String Joining

Joining Strings

- We can also join a list of strings back together!
  - The syntax is very different from **`split()`**
  - And it only works on a list of <u>strings</u>

$$\underset{\downarrow}{\texttt{"X"}}\underbrace{\texttt{.join}}_{\boxed{\text{function name}}}\texttt{(}\underbrace{\texttt{LIST\_OF\_STRINGS}}_{\boxed{\text{the list of strings we want to join together}}}\texttt{)}$$

$$\boxed{\text{the delimiter (what we will use to join the strings)}}$$

Example: Joining Strings

```
>>> names = ['Alice', 'Bob', 'Candi', 'Dave', 'Eve']
>>> "_".join(names)
'Alice_Bob_Candi_Dave_Eve'
```

- We can also use more than one character as our delimiter if we want

```
>>> " <3 ".join(names)
'Alice <3 Bob <3 Candi <3 Dave <3 Eve'
```

Splitting into Variables

Known (Formatted) Input

- Known input means that we know how the data inside a file will be formatted (laid out)
- For example, in workerHours.txt, we have:
  - The employee ID number
  - The employee's name
  - The hours worked over five days

workerHours.txt

```
123 Suzy 9.5 8.1 7.6 3.1 3.2
456 Brad 7.0 9.6 6.5 4.9 8.8
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

Splitting into Variables

- If we know what the input will look like, we can **`split()`** them directly into different variables

$$\underbrace{\texttt{var1, var2, var3}}_{\text{all of the variables we want to split the string into}} \texttt{ = } \underbrace{\texttt{threePartString}}_{\text{the string whose input we know, and are splitting on}}\texttt{.split()}$$

we can have as many different variables as we want

Example: Splitting into Variables

```
>>> s = "Jessica 31 647.28"
>>> name, age, money = s.split()
>>> name
'Jessica'
>>> int(age)
31
>>> float(money)
647.28
```

we may want to convert some of them to something that's not a string

Writing to Files

Opening a File for Writing

- Use **`open()`** just like we do for reading
  - Provide the filename <u>and the access mode</u>

```
fileObj = open("output.txt", "w")
```

- Opens the file for writing
- Wipes the contents!

```
fileObj = open("myNotes.txt", "a")
```

- Opens the file for appending
- Writes new data to the end of the file

Writing to a File

- Once a file has been opened, we can write to it

```
myFile.write( "hello world!" )
```

- We can also use a string variable in **`write()`**

```
myFile.write( writeString )
```

Word of Caution

- Write can only take one string at a time!

Why don’t these work?
the first is multiple strings
the second is an int, not a string

- These won’t work:

```
fileObj.write("hello", "my", "name")
fileObj.write(17)
```

Why does this work?
concatenation creates one string

- But this will:

```
fileObj.write("hello" + " my " + "name")
```

Closing a File

- Once we are done with our file, we close it
  - We do this for all files – ones that we opened for writing, reading, and appending!

```
myFileObject.close()
```

- Properly closing the file is important – why?
  - It ensures that the file is saved correctly

Exercise: Writing to a File

- Remember our grocery list program?
- At the end of our program, the user has added all of their items to the list **`grocery_list`**

- Write the contents of **`grocery_list`** to a file
  - Don't forget to open and close the file!

Solution: Writing to a File

```
# code above this populates grocery_list

# open file for writing
gFile = open("groceries.txt", "w")

for g in grocery_list:
    # print each item, plus a newline
    gFile.write(g + "\n")

# close file
gFile.close()
```

Writing to a File: Newlines

- Why did we need a newline in our example?

- Without it, our file looks like this:

```
durianscoconutlimecoke
```

- But with it, each item is on a separate line:

```
durians
coconut
lime
coke
```

Batch Programs

Batch Programs

- Batch mode processing is where program input and output are done entirely with files

- The program is <u>not</u> designed to be interactive

Practice

Exercise

- Suppose we have this **`hours.txt`** data:

```
123 Suzy 9.5 8.1 7.6 3.1 3.2
456 Brad 7.0 9.6 6.5 4.9 8.8
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

- Compute each worker's total hours and hours/day
  - Assume each worker works exactly five days
  - Sample output:

```
Suzy ID 123 worked 31.4 hou
Brad ID 456 worked 36.8 hou
Jenn ID 789 worked 39.5 hou
```

`workerHours.txt`

```
123 Suzy 9.5 8.1 7.6 3.1 3.2
456 Brad 7.0 9.6 6.5 4.9 8.8
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

Exercise Answer

```
def main():
   input = open("hours.txt")
   for line in input:
      id, name, mon, tue, wed, thu, fri = line.split()

      # cumulative sum of this employee's hours
      hours = float(mon) + float(tue) + float(wed) + \
                 float(thu) + float(fri)

      print(name, "ID", id, "worked", \
            hours, "hours: ", hours/5, "/ day")
main()
```

Exercise

- Write code to read a file of gas prices in USA and Belgium:

```
8.20    3.81    3/21/11
8.08    3.84    3/28/11
8.38    3.92    4/4/11
…
```

- Output the average gas price for each country to an output file named **`gasout.txt`**

Exercise: Batch Usernames

- Let’s create usernames for a computer system where the first and last names come from an input file
  - A username is the first letter of their first name, and the first 7 letters of their last name (lowercase)
- Get the input and output files from the user

Example Program: Batch Usernames

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

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

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

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

[continued...]
```

Example Program: Batch Usernames

```
[...continued]

    # process each line of the input file
    for line in infile:
        # get the first and last names from line
        first, last = line.split()
        # create a username
        uname = (first[0]+last[:7]).lower()
        # write it to the output file
        print(uname, file=outfile)

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

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

Announcements

- We will be doing an in-class worksheet next time
  - Bring pencils and paper (or your notebook)
- Homework 4 is out
  - Due by Tuesday (Oct 6th) at 8:59:59 PM
- Midterm is next week – Oct 14th and 15th
  - You **must** bring your UMBC ID with you to the exam! We won't accept your test without it.