CMSC201
Computer Science I for Majors

Lecture 10 – File I/O

Prof. Jeremy Dixon

Last Class We Covered

- Using **`while`** loops
  - Syntax
  - Using them for interactive loops
- Two different ways to mutate a list
  - **`append()`** and **`remove()`**
- Nested loops
- Two-dimensional lists (lists of lists)

Any Questions from Last Time?

Today's Objectives

- To learn about escape sequences
  - Why we need them
  - How to use them
- To be able to
  - Open a file
  - Read in its data
  - Close it after it's done
  - Manipulate the file object

Escape Sequences

“Misbehaving” `print()` Function

- There are times when the **`print()`** function doesn’t output exactly what we want

```
>>> print("I am 6 feet, 2 inches")
I am 6 feet, 2 inches
>>> print("I am 6'2"")
  File "<stdin>", line 1
    print("I am 6'2"")
                    ^
SyntaxError: EOL while scanning string literal
```

Special Characters

- Just like Python has special keywords…
  - **`for`**, **`int`**, **`True`**, etc.

- It also has special characters
  - single quote (`'`), double quote (`"`), etc.

Backslash: Escape Sequences

- The backslash character (\) is used to "*escape*" a special character in Python
  - Tells Python not to treat it as special

- The backslash character goes in front of the character we want to "escape"

```
>>> print("I am 6'2\"")
I am 6'2"
```

Using Escape Sequences

- There are three ways to solve the problem of printing out our height using quotes

```
>>> print("I am 6'2\"")
I am 6'2"
>>> print('I am 6\'2"')
I am 6'2"
>>> print("I am 6\'2\"")
I am 6'2"
```

Using Escape Sequences

- There are three ways to solve the problem of printing out our height using quotes

```
>>> print("I am 6'2\"")
I am 6'2"
>>> print('I am 6\'2"')
I am 6'2"
>>> print("I am 6\'2\"")
I am 6'2"
```

Common Escape Sequences

Escape Sequence	Purpose
`\'`	Print a single quote
`\"`	Print a double quote
`\\`	Print a backslash
`\t`	Print a tab
`\n`	Print a new line (“enter”)
`"""`	Allows multiple lines of text

`"""` is not really an escape sequence, but is useful for printing quotes

Escape Sequences Example

```
>>> tabby_cat = "\tI'm tabbed in."
>>> print(tabby_cat)
        I'm tabbed in.
```

\t adds a tab

```
>>> persian_cat = "I'm split\non a line."
>>> print(persian_cat)
I'm split
on a line.
```

\n adds a newline

```
>>> backslash_cat = "I'm \\ a \\ cat."
>>> print(backslash_cat)
I'm \ a \ cat.
```

\\ adds a single backslash

Escape Sequences Example

```
>>> fat_cat = """
... I'll do a list:
... \t* Cat food
... \t* Fishies
... \t* Catnip\n\t* Grass
... """
>>> print(fat_cat)

I'll do a list:
        * Cat food
        * Fishies
        * Catnip
        * Grass
```

Escape Sequences Example

```
>>> fat_cat = """
... I'll do a list:
... \t* Cat food
... \t* Fishies
... \t* Catnip\n\t* Grass
... """
>>> print(fat_cat)

I'll do a list:
        * Cat food
        * Fishies
        * Catnip
        * Grass
```

when using triple quotes ("""), the times you hit "enter" print as newlines

Escape Sequences Example

```
>>> fat_cat = """
... I'll do a list:
... \t* Cat food
... \t* Fishies
... \t* Catnip\n\t* Grass
... """
>>> print(fat_cat)

I'll do a list:
        * Cat food
        * Fishies
        * Catnip
        * Grass
```

\t puts in a tab

\n adds a newline

File Input/Output

Why Use Files?

- Until now, the Python programs you've been writing are pretty simple for input/output
  - Type input at the keyboard
  - Results (output) are displayed in the console
- This is fine for short and simple input…
  - But what if we want to average 50 numbers, and mess up when entering the 37th one?
  - Start all over???

What is File I/O?

- One solution is to read the information in from a file on your computer
  - You could even write information to a file
- This process is called *File I/O*
  - "I/O" stands for "input/output“
  - Python has built-in functions that make this easy

File I/O Example Usage

- "Read" in a file using a word processor
  - File opened
  - Contents read into memory (RAM)
  - File closed
  - IMPORTANT: Changes to the file are made to the copy stored in memory, not the original file on the disk

File I/O Example Usage

- "Write" a file using a word processor
  - (Saving a word processing file)
  - Original file on the disk is reopened in a mode that will allow writing
    - This actually erases the old contents!
  - Copy the version of the document stored in memory to the original file on disk
  - File is closed

File Processing

- In order to do interesting things with files, we need to be able to perform certain operations:
  - Associate an external file with a program object
    - Opening the file
  - Manipulate the file object
    - Reading from or writing to the file object
  - Close the file
    - Making sure the object and file match

Syntax: Opening a File

Syntax for **`open()`** Function

```
myFile = open(FILE_NAME [, ACCESS_MODE][, BUFFERING])
```

`FILE_NAME`

- This argument is a string the contains the name of the file you want to access
  - **`"input.txt"`**
  - **`"numbers.dat"`**
  - **`"roster.txt"`**

Syntax for `open()` Function

```
myFile = open(FILE_NAME [, ACCESS_MODE][, BUFFERING])
```

`ACCESS_MODE` (optional argument)

- This argument is a string that determines which of the modes the file is to be opened in
  - `"r"` (open for reading)
  - `"w"` (open for writing)
  - `"a"` (open for appending)

Syntax for `open()` Function

```
myFile = open(FILE_NAME [, ACCESS_MODE][, BUFFERING])
```

BUFFERING (optional argument)

- This argument is an integer that specifies to desired buffer size for the file
  - 0 (unbuffered)
  - 1 (line buffered)
  - >1 (buffer of approximately that size in bytes)

we won't be using buffering much (if at all) in this class

Examples of Using `open()`

- In general, we will use a command like:

```
myFile = open("FILENAME.txt")
```

- We will ignore the two optional arguments

```
myFile = open("scores.txt")
```

an example input file

```
scores.txt
2.5 8.1  7.6 3.2 3.2
3.0 11.6 6.5 2.7 12.4
8.0 8.0  8.0 8.0 7.5
```

File Processing: Reading

Reading Files

`name = open("filename")`

– opens the given file for reading, and returns a file object

`name.read()` – file's entire contents as a string

`name.readline()` – next line from file as a string

`name.readlines()` – file's contents as a list of lines

– the lines from a file object can also be read using a `for` loop

```
>>> f = open("hours.txt")
>>> f.read()
'123 Susan 12.5 8.1 7.6 3.2\n
456 Brad 4.0 11.6 6.5 2.7 12\n
789 Jenn 8.0 8.0 8.0 8.0 7.5\n'
```

Escape Sequences

File Input Template

- A template for reading files in Python:

```
name = open("filename")
for line in name:
    statements
```

strip() removes leading and trailing whitespace

```
>>> input = open("hours.txt")
>>> for line in input:
...     print(line.strip())   # strip() removes \n

123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

File Input Template

- Another way we can loop through a file is line by line using a **`for`** loop
- This reads the first 5 lines of a file:

```
infile = open(someFile, "r")
for i in range(5):
    line = infile.readline()
    print line[:-1]
```

Slicing is another way to strip out the newline characters at the end of each line

File Processing

- Another option is to ask the user for a file name to open
- First, prompt the user for a file name
- Open the file for reading

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

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

main()
```

Notice that we do not have a way to check if the file exists!

The file is read as one string and stored in the variable data

Exercise

- Write a program that goes through a file and reports the longest line in the file

Example Input File:

```
"carroll.txt"
        Beware the Jabberwock, my son,
        the jaws that bite, the claws that catch,
        Beware the JubJub bird and shun
        the frumious bandersnatch.
```

Example Output:

```
>>> longest.py
longest line = 42 characters
the jaws that bite, the claws that catch,
```

Exercise Solution

longest.py

```
def main():
    input = open("carroll.txt")
    longest = ""
    for line in input:
        if len(line) > len(longest):
            longest = line

    print("Longest line =", len(longest))
    print(longest)
main()
```

Splitting into Variables

- If you know the number of tokens, you can **`split()`** them directly into a sequence of variables

  `var1, var2, ..., varN = string.split()`

- May want to convert type of some tokens:

  `type(value)`

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

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 hours: 6.3 / day
Brad ID 456 worked 36.8 hours: 7.36 / day
Jenn ID 789 worked 39.5 hours: 7.9 / day
```

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()
```

From: https://courses.cs.washington.edu/courses/cse142/11au/python/06-files.ppt

File Processing (Writing)

Writing Files

```
name = open("filename", "w")
name = open("filename", "a")
```

- opens file for write (deletes previous contents), or
- opens file for append (new data goes after previous data)

`name.write(str)` – writes the given string to the file

`name.close()` – saves file once writing is done

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()

>>> open("output.txt").read()
'Hello, world!\nHow are you?'
```

From: https://courses.cs.washington.edu/courses/cse142/11au/python/06-files.ppt

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`**

File Processing

- When done with the file, it needs to be *closed*.
- Closing the file causes any outstanding operations and other bookkeeping for the file to be completed.
- In some cases, not properly closing a file could result in data loss.

File Processing

- 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

- Python can treat the file itself as a sequence of lines!

```
infile = open(someFile, "r")
for line in infile:
    # process the line here
infile.close()
```

Example Program: Batch Usernames

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

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)
```

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()`** method is used to convert the names into all lower case, in the event the names are mixed upper and lower case

Announcements

- (Pre) Lab 6 will be released Friday on Blackboard
- Homework 4 is out
  - Due by Tuesday (Oct 6th) at 8:59:59 PM
- Homework 1 re-grade and re-submit petitions must be made to your TA before Friday @ 3 PM
- Exam 1 will be on October 14th and 15th