File I/O in Python
Formats for Outputs

CS 8: Introduction to Computer Science, Winter 2018
Lecture #12

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- Homework #7 is **DUE on MONDAY (3/12)**
- Lab #5 will be out soon and due **Wed 3/7**
- Remaining on the calendar… ***This supersedes anything on the syllabus***

DATE	TOPIC	ASSIGNED	DUE
Mon. 3/5	File I/O ; Formats for Outputs	Hw #7 Lab #5	Hw #6 Lab #5
Wed. 3/7	Digital Images ; While-Loops		
Mon. 3/13	Recursive Functions	Hw #8 Lab #6	Hw #7, Hw #8 Lab #6 Proj #2
Wed. 3/15	Review for the Final Exam		

Lecture Outline

Chapter 5

- File I/O in Python
  - How to…
  - Demos
- Reading file over the Internet
- Formats for outputs

Starting Chapter 5

Files

- Mostly handle like any sequential data type
- A sequence of characters if a text file,
  or a sequence of bytes if a binary file
- Can you name some file *types* that are textual?
  Or binary?

Why Use Files?

4 Good Reasons:

- Files allow you to store data permanently and conveniently!
- Data output to a file lasts after the program ends
  - You can usually view them without the need of a Python program
- An input file can be used over and over
  - No typing of data again and again for testing
- Files allow you to deal with larger data sets

Input and Output in Computers

- Recall input and output are 2 of the main components of a computer

- There are different types of I/O
  - What we call “standard output” is usually the screen
  - What we call “standard input” is usually the keyboard
  - But there ARE other ways to get I/O
    - Like using files to write to (output) or to read from (input)

File I/O: Examples of How To

Example of READING from a file

```
filename = input
("What is the name of the file to
open? ")

InFile = open(filename, 'r')

count = 0
for line in InFile:
    count += 1
    print(line)
print("There are ", count, " lines
in the file ", filename)

InFile.close()
```

Example of WRITING to a file

```
filename = input
("What is the name of the file to
open? ")

OutFile = open(filename, 'w')

for n in range(10):
    myFile.write('Number ', n)

OutFile.close()
```

Read File

Example of READING from a file

```
filename = input
("What is the name of the file to
open? ")

InFile = open(filename, 'r')

count = 0
for line in InFile:
    count += 1
    print(line)
print("There are ", count, " lines
in the file ", filename)

InFile.close()
```

open() *function, using the* ***'r'*** *option means that we want to READ this file. Note that* ***filename*** *is a string.*

This is what we're doing to the lines that we read from the file. Note that the use of the ***print()*** *function here means that the output goes to* ***"standard output"*** *(i.e. your screen)*

Always ***close()*** *the file after opening it!*

Alternative instruction: `InFile = open(filename, 'r', encoding='utf-8')`

Write File

Example of WRITING to a file

```
filename = input
("What is the name of the file to
open? ")

OutFile = open(filename, 'w')

for n in range(10):
    myFile.write('Number ', n)

OutFile.close()
```

open() *function, using the* ***'w'*** *option means that we want to WRITE to this file. Note that* ***filename*** *is a string.*

This is the data that we're creating to put into the file. Note that the use of the ***write()*** *function here means that the output goes to* ***"file output"*** *(not "standard output")*

Always ***close()*** *the file after opening it!*

More Ways To Read A File

- Already saw: `for line in file` – to process each line as a separate string (including `'\n'` at ends of each line)
- To get just a *single* line (as a **string**): `file.readline()`
  - Do it again to get the next line, and so on
- Also can get a **list** of all lines (again, as **strings**) by using `file.readlines()`
  - This includes `'\n'` at ends of each line
  - Note readline**s** vs readline

DEMO!
Let's try it!

More Ways To **Read** A File

- You can also just `file.read()` – to get ***all*** of the file's text **as a single string**

- Note: use `open()` again if want to go back to the beginning of a file and read from start

Summary of Different Ways to Read

file.readline()	Reads ONE line (as a string)
file.readlines()	Reads ALL lines (as a list of strings)
file.read()	Reads ALL lines (as one giant string)

Demonstration

- **Given**: An input file with information on rainfall (in inches) for various geographical locations. Looks like this:

```
Akron 25.81
Albia 37.65 …etc…
```

- **You have to**: Create an output file that reads each line and outputs:

```
Akron had 25.81 inches of rain.
Albia had 37.65 inches of rain.
```

…etc…

See **rainfall.py** and **rainfall_advanced.py**

Reading a File Over the Internet

- Instead of reading a file that's in the same directory (i.e. *local*) , you can read a file that's on the Internet (i.e. *remote*)
- Need a properly-formatted **U**niform **R**esource **L**ocator (URL) string – then you can open the remote file:

```
import urllib.request
urlName = 'http://www.cs.ucsb.edu'
file = urllib.request.urlopen(urlName)
urlcontent = file.read().decode('utf-8')
```

Reading a File Over The Internet

- Now treat it *almost* like any file open for reading:

```
for line in file: # not okay to do – it is not iterable

oneLine = file.readline()    # use this to read ONE line
allLines = file.readlines()  # use this to read ALL lines
allText = file.read()        # use this to read ALL lines
```

See **getWebpage.py**

Formatting Output Lines 1

- Applies whether for output to “standard” (i.e. display) or to a file

- All file I/O are considered strings ***BY DEFAULT***, so you may have to do some **conversions** if you want them to be used as integers or float (e.g. to be able to do numerical calculations)
  - Example:

```
num = int( infile.readline( ) )
# the file I’m reading has lines of just integers
# I want to ensure that these inputs will be used as int
doubleNum = num * 2
```

Formatting Output Lines2

- You can use a Python **template** called formatted strings
  - Uses an operator **%**
  - In the context of its use this IS NOT THE MODULO OPERATOR!

- For example, instead of:

```
print("My friend", FriendName, "is", age, "years old")
```

you can do:

```
print("My friend %s is %d years old" % (FriendName, age))
```

*%s = a **string** goes here*

*%d = an **integer** goes here*

Why Do This?

- Because we can easily use **format modifiers**

```
print("My friend %s is %5d years old" % (FriendName, age))
```

- This example puts the value of variable age in a field width of 5 spaces

```
My friend John is    33 years old
```

5 spaces

Format Modifiers: More Examples

See Tables 5.2 & 5.3 in Textbook

```
>>> print("These %d bottles of beer on the wall" % 99)
'These 99 bottles of beer on the wall'

>>> n = 4
>>> cost = 0.5
>>> print ("%d items, each for $%.2f" % (n, cost))
'4 items, each for $0.50'

>>> name = "Mr. Bojangles"
>>> print ("Hello, %s, how are you?!" % (name))
```

%s - String (or any object with a string representation, like numbers)

%d - Integers

%f - Floating point numbers

%.<number of digits>f - Floating point numbers with a fixed amount of digits to the right of the dot.

New Way: Using `.format`

Similar ideas, different syntax:

```
>>> "{1} has ${0}".format(42,'Joe')
'Joe has $42.00’

>>> "{1} has ${0:.2f}".format(42,'Joe')
'Joe has $42.00'

>>> "{0} is {age} years old".format('Edward',age=20)
'Edward is 20 years old'
```

New Way: Using `.format`

- *Keyword* arguments are handy,
  - Especially if there are lots of arguments

- More information can be found at:

http://www.python-course.eu/python3_formatted_output.php

(but, alas, not in our text book)

THIS WEBSITE MATERIAL IS REQUIRED READING

YOUR TO-DOs

- Next: **Chapter 6**

- Start working on HW # 7, Lab # 5
- Keep working on **Project2** (due **Friday 3/16**)

</LECTURE>