File I/O in Python

CS 8: Introduction to Computer Science
Lecture #11

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

Midterm #2 is next week on Thursday 5/18!

Tutoring/Review Session Available!

- TWO of them again!
- Friday, **5/12** at **1:00 PM & 2:30 PM** in **PHELPS 3525**

Next class (Tuesday, 5/16) will be used as a **review** for the midterm

Homework assignment is due TUESDAY, 5/23

Highly recommend that you work on it as prep for the midterm!

Example Programs

FYI and ICYDK(?!):

All the example programs we use in class

<u>are ALSO on our website!</u>

Stressing a point:
`return x` vs. `print(x)`

- ***Not the same thing!***
  - Python interpreter (IDLE) just makes it seem that way!

- If a function **returns** a result, that result can be used later – for printing or whatever else
  - e.g., `func1(value)` returns an integer, so we can things like:

```
newResult = func1(5) + 92
```

- If a function **prints** a result – that's ALL it does!
  - e.g., `func2()` prints, but doesn't return anything, so:

```
result = func2()
```

… stuff gets printed here

If you did this instead:

`print(result)`*, you'd get nothing…*

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

Files

- First, **open** file, and state purpose – **r**ead or **w**rite

```
ThatImportantFile = open('mydata.txt', 'r')
MyPrettyOutputs   = open('myresults.txt', 'w')
```

- Often process text files as a sequence of lines

```
for line in inputFile: # process each line as a string
    outputFile.write(line[:5] + '\n')
```

- Best to **close** the files when you're done

```
inputFile.close()
outputFile.close()
```

DEMO!
Let's try it!

More Ways To Read A File

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

DEMO!
Let's try it!

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…*

- 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

- Need a properly-formatted Uniform Resource Locator string – then you can open the remote file:

```
import urllib.request
urlName = "http://www.cs.ucsb.edu"
file = urllib.request.urlopen(urlName)
```

Reading a file over the Internet

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

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

oneLine = file.readline()      # okay
allLines = file.readlines()    # okay
allText = file.read()          # okay
```

See **getWebpage.py**

Formatting Output Lines 1

- Applies whether for output to display or to file

- All file I/O is considered strings, so you may have to do some **conversions** if you want to do numerical calculations
  - e.g.

```
num = int(infile.readline( ))
# you want to use the input you read as an integer
doubleNum = num * 2
```

Formatting Output Lines 2

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

- See **table 5.3** in textbook

- You can:
  - Make right- or left-justified formats
  - Put in leading zeros
  - Define how many decimal points you want to show
  - *And other things…*

YOUR TO-DOs

- We're still going thru **Chapter 5**, so read ahead
- Keep working on **Project1** (due **Friday 5/12**)

</LECTURE>