CMPT 120
Intro to CS & Programming I

WEEK 11 (Mar. 24-28)

— *Jérémie O. Lumbroso* —

Lecture 26:
More File Input and Output

http://www.sfu.ca/~jlumbros/Courses/CMPT120/

Recap of last lecture

PREVIOUSLY, ON CMPT 120…

Recap on Files

- We can read/write files with Python
- First: `f = open(filename, mode)` where
  - `filename` contains the name of the file that will be opened in the folder given by `os.getcwd()` (note: it is also possible to give a complete path)
  - `mode` is a string, for the moment either `"r"` if we to read from the file or `"w"` to write to it
- `f.read()`: read whole content of file into string
- `f.write(s)`: write string `s` to file
- For the moment, we have not done anything very complex on a file (only one liners)

Other Useful Functions

- `s.strip()` removes all whitespace at the beginning and end of the string `s`
- `s.split("-")` creates a list where the string `s` is split at the character `"-"`
- **Newline characters:** `"\n"` and `"\r"` or both (which are considered whitespace)

Current Working Directory (1)

- The `os` module contains functions to manipulate and navigate the file system
- We used `os.getcwd()` to get current working directory (where Python expects to find files)
- In IDLE, it can be changed by **saving** a file and **restarting** the Python shell (or **running** a module with F5)
- **Consequence:** make sure the folder does not change while you are working

AND NOW…

Current Working Directory (2)

- It is possible to change the directory using `os.chdir(folder)` where `folder` is a path

```
>>> import os
>>> os.getcwd()
/Users/jlumbroso/Documents
>>> os.chdir("/Users/jlumbroso")
>>> os.getcwd()
/Users/jlumbroso
```

- **Important:** in Windows, paths use the **backslash** `"\"`, and Python always doubles this character, for ex. `"C:\\Python27"` is the path `"C:\Python27"`

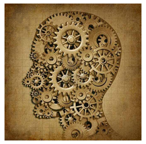
Read Lines

- We worked with files with only one line
- If the file has many lines, use `f.readlines()`
- It reads all lines of a file and puts them in a **list**

```
>>> f = open("file1.txt")
>>> lines = f.readlines()
>>> lines
['Avery Baird\n', 'Gay Brower\n',
'Felecia Binkley\n', 'Clarice Cothran\n',
'Jarvis Deaton\n', 'Rocco Fite\n'...
```

- Can **iterate** over each line, like we do with lists

Clean Up Input



- As before, `s.strip()` will remove `"\n"` character at the end of each name
- Clean all names, iterate over list and **modify** each element

```
f = open("file1.txt")
lines = f.readlines()
for i in range(len(lines)):
    # iterate over position because
    # elements need to be modified
    lines[i] = lines[i].strip()
```

A	Done
B	No computer

- (If we iterate over elements, `for line in lines`, we cannot modify them.)

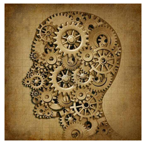
Compare two/three files

TASK 1: COUNTING MALES

Our (First) Task

- The file `"file1.txt"` contains a list of people
  - Each line: `<first name> <last name>\n`
  - We want to count the number of **females** and the number of **males**
- How?
- Two other files (http://deron.meranda.us/data/)
  - `"popular-female-first.txt"` contains a list of popular female first names in the US in 1990s
  - `"popular-male-first.txt"`, male first names
- All first names of `"file1.txt"` in these lists

Task 1: Plan of Action



A. Prepare the reference lists

- Read names from "`popular-female-first.txt`" using `readlines()` and put it variable `fnames`
- Clean up: apply `s.strip()` and `s.lower()`
- Do the same with the male names, put in `mnames`

A Done

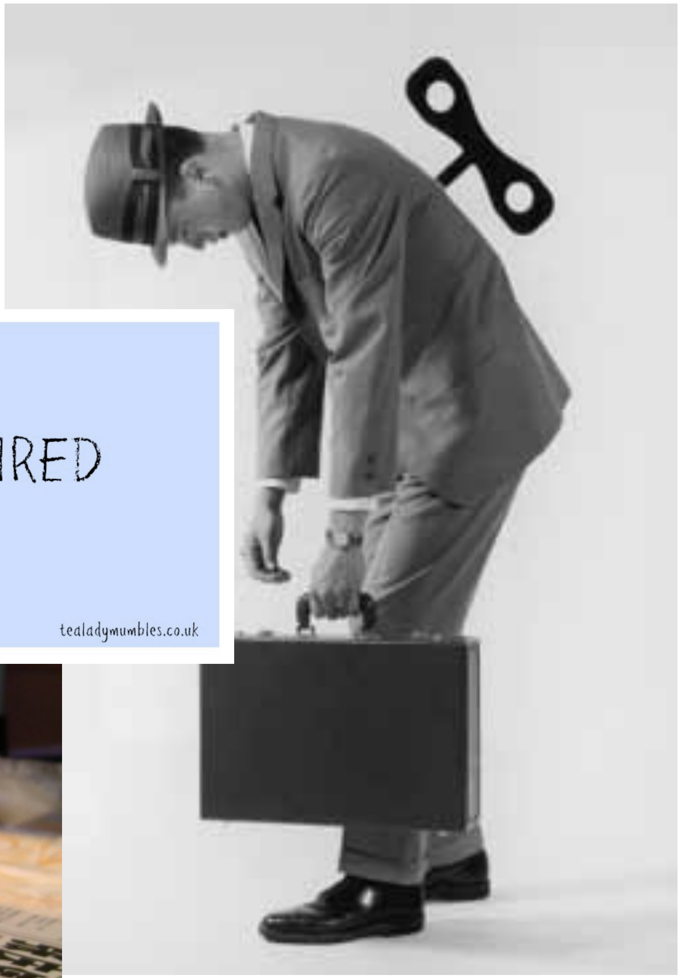
B Help!

C No computer

B. Process "`file1.txt`"

- Read all lines into variable lines
- Clean up: apply `s.strip()` and `s.lower()`, then `s.split()` to sep. first and last name, and only keep first
- Check if name is in `fnames` or `mnames`, and increment counter

```
L = [ "abc", "edf" ]
if "abc" in L:
   counter = counter + 1
   print "that element is in the list"
```



I'm Oh So....TIRED
tealadymumbles.co.uk

Sidenode: Why do we Lowercase?

- The `"popular-...-first.txt"` files:

```
AARON
ABEL
ABRAHAM
…
```

- The `"file1.txt"` file:

```
Avery Baird
Gay Brower
Felecia Binkley
Clarice Cothran
Jarvis Deaton
…
```

- Since
  - we are going to compare items from these files together
  - the files have different capitalization
- It makes sense to convert everything to the same capitalization (here, lowercase – but it could have been anything else)

Task 1: Prepare Reference Lists

- Because we are doing three times the same thing (loading a list of names from a file), it is perhaps a good idea to create a function, rather than writing the same code thrice

http://goo.gl/CNQwkt

```
def load_names(filename)
  f = open(filename)
  lines = f.readlines()
  for i in range(len(lines)):
    # Clean up the line (strip, and lowercase)
    lines[i] = lines[i].strip().lower()
    # In case there is more than one word on
    # the line, we always take the first only
    lines[i] = lines[i].split()[0]
  return lines
```

Copy/download this function

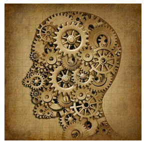
- Then, part **A.** is simply

```
fnames = load_names("popular-female-first.txt")
mnames = load_names("popular-male-first.txt")
```

A Done

B No computer

Task 1: Process "file1.txt"



B. Process "file1.txt"

- Read all lines into variable lines
- Clean up: apply `s.strip()` and `s.lower()`, then `s.split()` to sep. first and last name, and only keep first
- Check if name is in `fnames` or `mnames`, and increment counter **counting** the number of female and male names

```
L = [ "abc", "edf" ]
if "abc" in L:
  counter = counter + 1
  print "element is in the list"
```

Can you do this now that you have the function `load_names` (which does the first two steps for you)?

HOW MANY MALE NAMES? (vote with iClicker)

Task 1: Solution

```
fnames = load_names("popular-female-first.txt")
mnames = load_names("popular-male-first.txt")

lines = load_names("file1")
males = 0
females = 0

for name in lines:
  if name in fnames:
    females = females + 1
  if name in mnames:
    males = males + 1

print "There are", males, "male names."
```

- And the answer is… 60.

Detecting very similar programs…

TASK 2: CATCHING COPIES

A Little Bit of Background

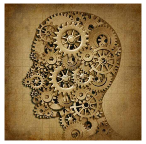
- The list of names is actually a list of students
- Students of Fonda Boise, a CS instructor, who gave a programming assignment
- She does not tolerate students copying each other's work, so she has a detection program
- The detection program
  - compares all **pairs** of submissions
  - measures the **similarity** (which percent of lines of code are the same)
  - measures the proportion of **dissimilar lines** that are neither `print` or `raw_input` statements

"file3.txt"

```
Kenneth Badillo,Armand Britton,0.86,0.68,0.38,0.49,0.25
Kenneth Badillo,Octavio Bunting,0.85,0.56,0.36,0.34,0.34
Kenneth Badillo,Tracie Cade,0.76,0.56,0.22,0.45,0.33
Kenneth Badillo,Timothy Fair,0.72,0.53,0.43,0.34,0.39
Kenneth Badillo,Susan Blanchette,0.88,0.70,0.24,0.21,0.38
Kenneth Badillo,Archie Archer,0.81,0.65,0.46,0.21,0.48...
```

- 7 columns, separated by comma "," (**CSV file**)
  - two first columns are names of students
  - five last columns are scores
    - similarity score of first student, then second student
    - dissimilarity measure of first, then second student
- Need to detect lines where: similarity is **high** and dissimilarity is **low**

Task 2: Catching Duplicates

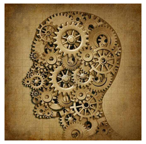


```
Kenneth Badillo,Armand Britton,0.86,0.68,0.38,0.49,0.25
Kenneth Badillo,Octavio Bunting,0.85,0.56,0.36,0.34,0.34
Kenneth Badillo,Tracie Cade,0.76,0.56,0.22,0.45,0.33
```

- Read lines from `"file3.txt"`
- Strip of white space
- Split according to the `","`
- Convert 5 last entries with `float(...)` function
- Consider that the similarity (`scoreS`) is **`max`** of the first two floats (i.e., `0.85`,`0.56`)
- Consider that the dissimilarity (`scoreD`) is **`min`** of the next two floats (i.e., `0.22`,`0.45`)
- Print line if `scoreS > 0.55` and `scoreD < 0.12`
- **Vote with iClickers: HOW MANY CHEATERS?**

Pacing and Understanding

How well did you understand today?



- **A** Too easy, this lecture is way below my abilities
- **B** Everything went at a good pace, and I am fine
- **C** Too fast, but I will catch up on my own
- **D** Too fast, and I need you to slow down
- **E** I really do not think I can handle this