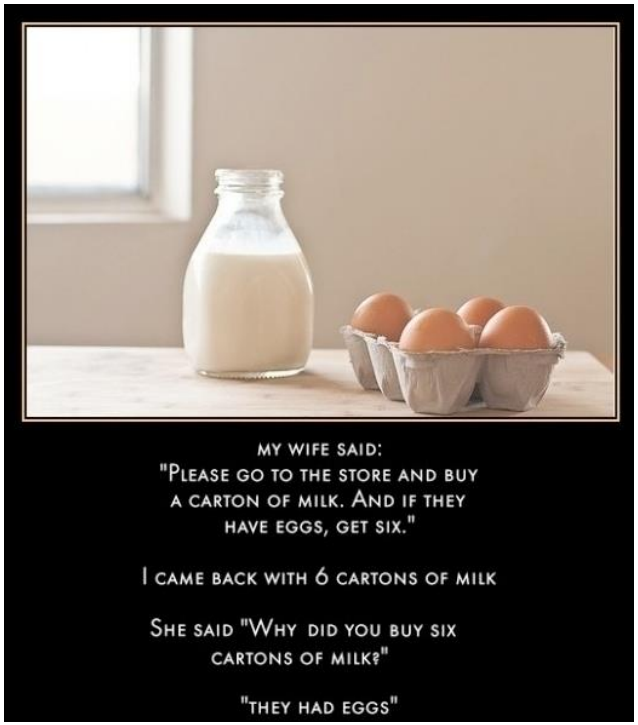
CSc 110, Spring 2017

Lecture 18: Line-Based File Input

Adapted from slides by Marty Stepp and Stuart Reges



Programming feel like that?

IMDb movies problem

- Consider the following Internet Movie Database (IMDb) data:

```
1 9.1 196376 The Shawshank Redemption (1994)
2 9.0 139085 The Godfather: Part II (1974)
3 8.8 81507 Casablanca (1942)
```

- Write a program that displays any movies containing a phrase:

```
Search word? part

Rank    Votes   Rating  Title
2       139085  9.0     The Godfather: Part II (1974)
40      129172  8.5     The Departed (2006)
95      20401   8.2     The Apartment (1960)
192     30587   8.0     Spartacus (1960)
4 matches
```

Note: the file is `imdb.txt`

Pseudocode

```
1 9.1 196376 The Shawshank Redemption (1994
2 9.0 139085 The Godfather: Part II (1974)
3 8.8 81507 Casablanca (1942)
```

ask the user for a search word
open the IMDb data file
create a list of the files contents
if search word is in the list of files contents
print the header for the output
set matches counter
for each line in the list
if the search word is in the line
increment the matches counter
print the line in the proper format
print the number of matches

```
Rank    Votes   Rating  Title
  2     139085  9.0     The Godfather: …
```

Functions for imdb

Start with these:

`get_phrase()`

- use `input()` as usual
- returns a string

`search_list(line_list, search_word)`

- searches `line_list` to find `search_word`
- if finds `search_word`, returns the line
- if no match found, returns an empty string

We want to put in debugging prints statements

We want to *know* the structure of `imdb.txt`

(Let's take a look at it before writing the code.)

Remaining functions for imdb

`search_line(line, search_word)`

- searches string `line` to find `search_word`
- if finds `search_word`, returns the line
- if no match found, returns an empty string

Remaining functions for imdb

`display(line)`

- `line` is a string

A string in `line` looks like this:

```
'2 9.0 139085 The Godfather: Part II (1974)'
```

The structure of the output is:

Rank	Votes	Rating	Title
2	139085	9.0	The Godfather: Part II (1974)

What does the method `split()` do?

-creates a list from a string

-by default uses whitespace (thiink spaces) to delimit the elements

Helpful strategies

- Use print statements for debugging
- Make your input file small to start with
- Know the structure of your input file
- Review the methods and functions for lists and strings

IMDb main

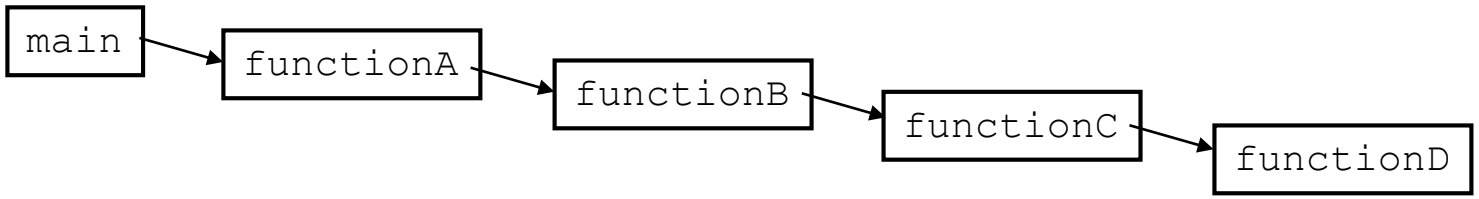
```
# Displays IMDB's Top 250 movies that match a search string.

def main():
    search_word = get_phrase()
    file = open("imdb.txt")
    line_list = file.readlines()
    line = search_list(line_list, search_word)

    if (len(line) > 0):
        print("Rank\tVotes\tRating\tTitle")
        matches = 0
        for a_line in line_list:
            ans = search_line(a_line, search_word)
            if (len(ans) > 0):
                matches = matches + 1
                display(a_line)
        print(str(matches) + " matches.")

# Asks the user for their search word and returns it.
def get_phrase():
    search_word = input("Search word: ")
    search_word = search_word.lower()
    print()
    return search_word
...
```

IMDb functions

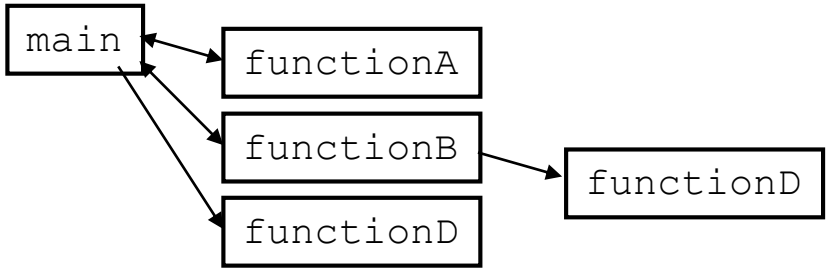
```
    ...

# Searches a list of lines for a line that match the search word.
def search_list(line_list, search_word):
    for line in line_list:
        line_lower = line.lower()     # case-insensitive match
        if (search_word in line_lower):
            return line
    return ""   # not found

# Looks for the search word in a single line.
def search_line(line, search_word):
    line_lower = line.lower()         # case-insensitive match
    if (search_word in line_lower):
        return line
    return ""   # not found


# displays the line in the proper format on the screen.
def display(line):
    parts = line.split()
    rank = parts[0]
    rating = parts[1]
    votes = parts[2]
    title = ""
    for i in range(3, len(parts)):
        title  = title + parts[i] + " "    # the rest of the line
    print(rank + "\t" + votes + "\t" + rating + "\t" + title)
```

"Chaining"

- `main` should be a concise summary of your program.
  - It is bad if each function calls the next in a nested structure (we call this *chaining*):



- A better structure has `main` make most of the calls.
  - Functions must return values to `main` to be passed on later.



File output

Output to files

- Open a file in write or append mode
  - 'w' - write mode – replaces everything in the file
  - 'a' – append mode – adds to the bottom of the file preserving what is already in it

```
name = open("filename", "w")      # write
name = open("filename", "a")      # append
```

Output to files

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

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

Example:

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

text = open("output.txt").read()
  # Hello, world!\nHow are you?
```

Removing short words

- Write a program that reads a file, writes the file contents , and then creates a new file containing only the words greater than 3 characters.

Removing short words

```
def main():
    file = open("poem.txt")
    text = file.read()

    print(text)
    outfile = open("nosmallwords.txt", "w")
    text = text.split()
    for word in text:
       if len(word) > 3:
            print(word)
            outfile.write(word)
    outfile.close()
main()
```