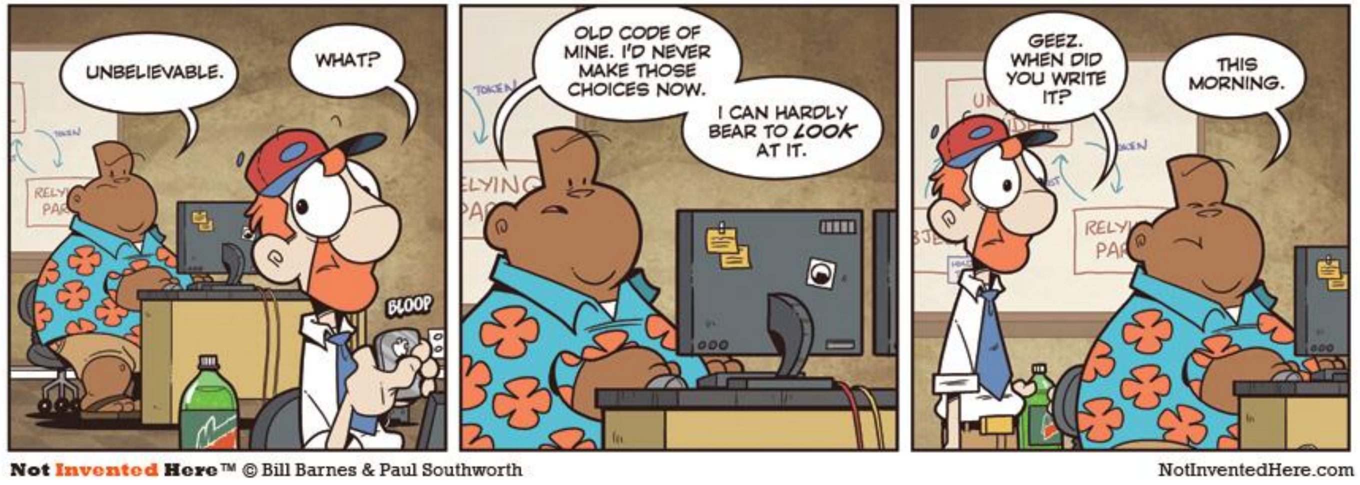
CSc 110, Autumn 2017

Lecture 20: Line-Based File Input

Adapted from slides by Marty Stepp and Stuart Reges


UNBELIEVABLE.
WHAT?
BLOOP
OLD CODE OF MINE. I'D NEVER MAKE THOSE CHOICES NOW.
I CAN HARDLY BEAR TO LOOK AT IT.
GEEZ. WHEN DID YOU WRITE IT?
THIS MORNING.
Not Invented Here™ © Bill Barnes & Paul Southworth
NotInventedHere.com

Hours question

- Given a file `hours.txt` with the following contents:

```
123 Clark 12.5 8.1 7.6 3.2
456 Tate 4.0 11.6 6.5 2.7 12
789 Zach 8.0 8.0 8.0 8.0 7.5
```

- Consider the task of computing hours worked by each person:

```
Clark (ID#123) worked 31.4 hours (7.85 hours/day)
Tate (ID#456) worked 36.8 hours (7.36 hours/day)
Zach (ID#789) worked 39.5 hours (7.90 hours/day)
```

Line-based file processing

- Instead of using `read()` use `readlines()` to read the file
- Then use `split()` on each line

```
file = open("<filename>")
lines = file.readlines()
For line in lines:
    parts = line.split()
    <process the parts of the line>
```

Hours answer

```
# Processes an employee input file and outputs each employee's hours.
def main():
    file = open("hours.txt")
    lines = file.readlines()
    for line in lines:
        process_employee(line)

def process_employee(line):
    parts = line.split()
    id = parts[0]           # e.g. 456
    name = parts[1]         # e.g. "Greg"
    sum = 0
    count = 0
    for i in range(2, len(parts)):
        sum += float(parts[i])
        count += 1
    average = sum / count
    print(name + " (ID#" + id + ") worked " +
        str(sum) + " hours (" + str(average) + " hours/day)")
```

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:

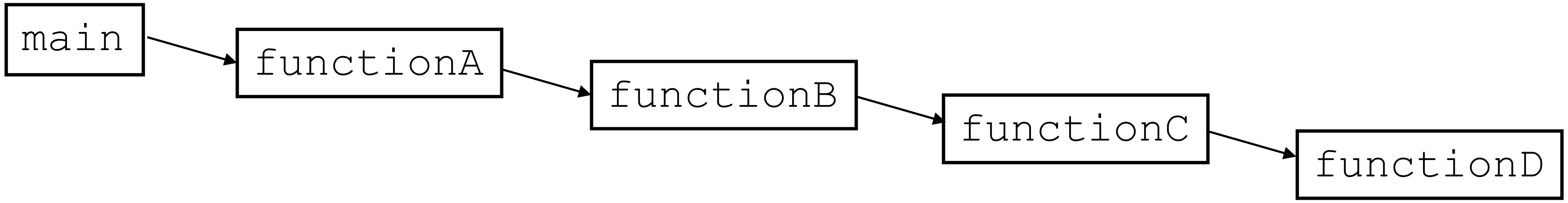
```
Search word? part

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.
```

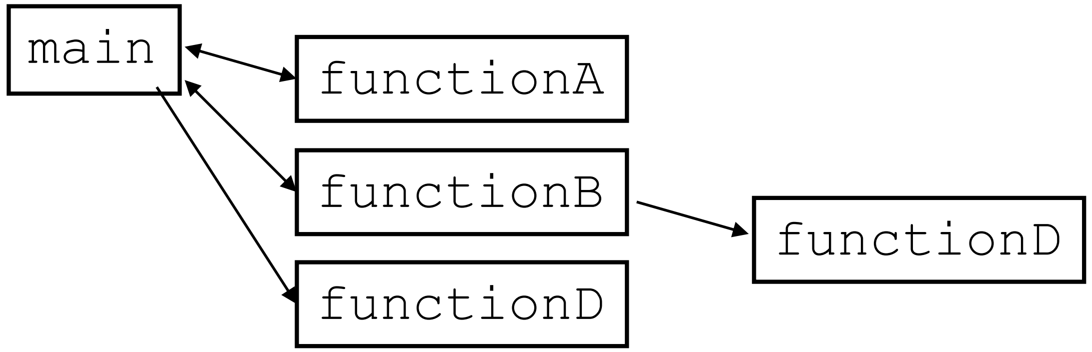
  - Is this a token or line-based problem?

"Chaining"

- `main` should be a concise summary of your program.
  - It is bad if each function calls the next without ever returning (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.



Bad IMDb "chained" code 1

```
# Displays IMDB's Top 250 movies that match a search string.
def main():
    get_word()

# Asks the user for their search word and returns it.
def get_word():
    search_word = input("Search word: ")
    search_word = search_word.lower()
    print()
    file = open("imdb.txt")
    search(file, search_word)

# Breaks apart each line, looking for lines that match the search word.
def search(file, search_word):
    matches = 0
    for line in file:
        line_lower = line.lower()      # case-insensitive match
        if (search_word in line_lower):
            matches += 1
            print("Rank\tVotes\tRating\tTitle")
            display(line)
```

Bad IMDb "chained" code 2

```
# 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 += parts[i] + " "    # the rest of the line
    print(rank + "\t" + votes + "\t" + rating + "\t" + title)
```

Better IMDb answer 1

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

def main():
    search_word = get_word()
    file = open("imdb.txt")
    line = search(file, search_word)

    if (len(line) > 0):
        print("Rank\tVotes\tRating\tTitle")
        matches = 0
        while (len(line) > 0):
            display(line)
            line = search(file, search_word)
            matches += 1
        print(str(matches) + " matches.")

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

Better IMDb answer 2

```
    ...

# Breaks apart each line, looking for lines that match the search word.
def search(file, search_word):
    for line in file:
        line_lower = line.lower()     # case-insensitive match
        if (search_word in line):
            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 += parts[i] + " "    # the rest of the line
    print(rank + "\t" + votes + "\t" + rating + "\t" + title)
```