CompSci 101
Introduction to Computer Science

<table>
<thead>
<tr>
<th></th>
<th>ABP</th>
<th>BlueEx</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Nasher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

April 19, 2016

Prof. Rodger
Announcements

• Reading and RQ due next time
• Assign 8 due Thur., Assign9 next week
• APT 11 due on Tuesday
• APT Quiz 3 Sun-Tue next week

• Today:
  – Review Recursion
  – Regular Expressions
  – Assignment 8 Recommender
  – Last lab this week
Recursion Review

• Function calls a clone of itself
  – Smaller problem
  – Must be a way out of recursion
Example

def Mystery(num):
    if num > 0:
        return 1 + Mystery(num/2)
    else:
        return 2 + num

• Mystery(5) is $1 + 1 + 2 = 1 + 4 = 5$
• Mystery(2) is $1 + 1 + 2 = 1 + 3 = 4$
• Mystery(1) is $1 + 1 + 2 = 1 + 2 = 3$
• Mystery(0) is $2$
Recursion Review
Solving a problem by solving similar but smaller problems

**Question** - How many **rows** are there in this **classroom**?

**Similar but smaller question** - How many **rows** are there **until your row**?

Last row

- I don’t know, let me ask
- I don’t know, let me ask
- I don’t know, let me ask
- I don’t know, let me ask
- I don’t have anyone to ask. So I am in Row#1

**Row count** = 4+1 = 5

**Return Value** = 3+1 = 4

**Return Value** = 2+1 = 3

**Return Value** = 1+1 = 2

**Return value** = 1
Recursion to find ALL files in a folder

• A folder can have sub folders and files
• A file cannot have sub files

def visit(dirname):
    for inner in dirname:
        if isdir(inner):
            visit(inner)
        else:
            print name(inner), size(inner)
Revisit the APT Bagels Recursively

```python
filename: Bagels.py

def bagelCount(orders):
    """
    return number of bagels needed to fulfill the orders in integer list parameter orders
    """
```

1. orders = [1, 3, 5, 7]

   Returns: 16

   No order is for more than a dozen, return the total of all orders.

2. orders = [11, 22, 33, 44, 55]

   Returns: 175 since 11 + (22+1) + (33+2) + (44+3) + (55+4) = 175
def bagelCount(orders):
    if len(orders) > 0:
        return orders[0]/12 + orders[0] + bagelCount(orders[1:])
    else:
        return 0

def bagelCount(orders):
    if len(orders) > 0:
        return orders[-1]/12 + orders[-1] + bagelCount(orders[:-1])
    else:
        return 0

def bagelCount(orders):
    return orders[0] + orders[0]/12 + bagelCount(orders[1:])

def bagelCount(orders):
    if len(orders)>1:
        return orders[1] + orders[1]/12 + bagelCount(orders[2:])
    else:
        return bagelCount(orders[0])
What is Computer Science?

• … "it is the study of automating algorithmic processes that scale."

• If you need to find one email address on a webpage, you don't need computer science
  – If you need to scrape every email address, that number in the 10's to 100's, you could use help
How do you solve a problem like …

• How many words end in "aria"?
  – Start with "aria"? Contain "aria"?
  – Why would you care about this?

• Can you find ola@cs.duke.edu, susan.rodger@duke.edu, and andrew.douglas.hilton@gmail.com when searching through a webpage source?
  – What is the format of a "real" email address?
Examples of regex's at work

• What do aria$ and ^aria and aria share?
  – Answers to previous question

• What about the regex .+@.+
  – Turns out that . has special meaning in regex, so does +, so do many characters

• We'll use a module RegexDemo.py to check
  – Uses the re Python library
  – Details won't be tested, regex knowledge will
Regex expressions

- Regex parts combined in powerful ways
  - Each part of a regex "matches" text, can extract matches using programs and regex library
  - `^` is start of word/line, `$` is end

- Expressions that match single characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, a, 9 or ...</td>
<td>Any character matches itself</td>
</tr>
<tr>
<td>.</td>
<td>Matches any character</td>
</tr>
<tr>
<td>\w</td>
<td>Matches alphanumeric and _</td>
</tr>
<tr>
<td>\d</td>
<td>Matches digit</td>
</tr>
<tr>
<td>\s</td>
<td>Matches whitespace</td>
</tr>
</tbody>
</table>
Regex expressions

• Repeat and combine regex parts
  – * means 0 or more occurrences/repeats
  – + means 1 or more occurrences/repeats
  – ? Means (after * or +) to be non-greedy

• Expressions match more than one character

| [a-zAB]         | Brackets create character class |
| (regex)        | Tag or group a regex            |
| \1 or \2       | Matches previously grouped regex |
| {1} or {n}     | Repeat regex 1 or n times       |
Regex examples tried and explained

• Five letter words ending in p? Starts 'd'?
  – ^\w\w\w\wp$ but not ....p$

• Seven letter words, or seven ending with 'z'
  – Difference between ^\w\{7\}$ and ^\w\{7\}

• Words that start with a consonant:
  – ^[^aeiou] double meaning of ^
Regex examples tried and explained

• Five letter words ending in p? Starts 'd'?
  – ^\w\w\w\wp$ but not . . . . . p$

• Seven letter words, or seven ending with 'z'
  – Difference between ^\w{7}$ and ^\w{7}

• Start and end with the same two letters like sense and metronome, decipher this:
  – ^\((__\w\w)\).*\1$

• Start and end with three letters reversed, like despised and foolproof?
# Summary of Regular Expressions

<table>
<thead>
<tr>
<th>Regex</th>
<th>Purpose</th>
<th>Regex</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>any character</td>
<td>*</td>
<td>zero or more of previous regex</td>
</tr>
<tr>
<td>\w</td>
<td>any alphanumeric character (and _)</td>
<td>+</td>
<td>one or more of previous regex</td>
</tr>
<tr>
<td>\s</td>
<td>any whitespace character</td>
<td>*? or +?</td>
<td>non-greedy version of either * or +</td>
</tr>
<tr>
<td>\d</td>
<td>any digit character</td>
<td>()</td>
<td>tag/group a regular expression</td>
</tr>
<tr>
<td>[]</td>
<td>character class, e.g., [A-Z] or [aeiou]</td>
<td>\1, \2, ..</td>
<td>match numbered tagged/grouped regex</td>
</tr>
<tr>
<td>{n}</td>
<td>n occurrences of preceding regex</td>
<td>^</td>
<td>beginning of line/string</td>
</tr>
<tr>
<td>[^...]</td>
<td>not the characters in the class, e.g.,[^aeiou]</td>
<td>$</td>
<td>end of line/string</td>
</tr>
</tbody>
</table>
Regex Questions
bit.ly/101sp16-0419-2
Assignment 8
From User Rating to Recommendations

![Image of movies: Spectre, Martian, Southpaw, Everest, PitchPerfect 2]

<table>
<thead>
<tr>
<th></th>
<th>Spectre</th>
<th>Martian</th>
<th>Southpaw</th>
<th>Everest</th>
<th>PitchPerfect 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>-3</td>
<td>5</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

- **What should I choose to see?**
  - What does this depend on?
- **Who is most like me?**
  - How do we figure this out
Data For Recommender

• Users/Raters rate Items
  – We need to know the items
  – We need to know how users rate each item

• Which eatery has highest average rating?
  – Conceptually: average columns in table
  – How is data provided in this assignment?

<table>
<thead>
<tr>
<th></th>
<th>ABP</th>
<th>BlueEx</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Nasher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Data For Recommender

- itemlist are provided in a list of strings
  - Parsing data provides this list
- dictratings provided in dictionary
  - Key is user ID
  - Value is list of integer ratings

<table>
<thead>
<tr>
<th></th>
<th>ABP</th>
<th>BlueEx</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Nasher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Data For Recommender

• Given Parameters
  – itemlist: a list of strings
  – dictratings: dictionary of ID to ratings list

• Can you write
  – Average(itemlist, dictratings)

<table>
<thead>
<tr>
<th></th>
<th>ABP</th>
<th>BlueEx</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Nasher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>Chris</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>Nat</td>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Drawbacks of Item Averaging

• Are all ratings the same to me?
  – Shouldn't I value ratings of people "near" me as more meaningful than those "far" from me?

• Collaborative Filtering
  – How do we determine who is "near" me?

• Mathematically: treat ratings as vectors in an N-dimensional space, N = # ratings
  – Informally: assign numbers, higher the number, closer to me
Collaborative Filtering: Recommender

• First determine closeness of all users to me:
  – "Me" is a user-ID, parameter to function
  – Return list of (ID, closeness-#) tuples, sorted

• Use just the ratings of person closest to me
  – Is this a good idea?
  – What about the 10 closes people to me?

• What about weighting ratings
  – Closer to me, more weight given to rating
Collaborative Filtering

• For Chris: 12 \[1,1,0,3,0,-3] = - [12,12,0,36,0,-36]

• For Sam: [0,75,125,0,-75,125]
Adding lists of numbers

\[ [12, 12, 0, 36, 0,-36] \]
\[ [0, 75, 125, 0,-75,125] \]
\[ [-111,111,111,185,37, -37] \]
\[ ------------------------------- \]
\[ [-99, 198, 236, 221, -38, 52] \]

• Adding columns in lists of numbers
  – Using indexes 0, 1, 2, … sum elements of list
  – \( \text{sum}([\text{val}[i] \text{ for } \text{val} \text{ in d.values()}]) \)
Then divide by number of nonzeros

\[
\begin{bmatrix}
12, & 12, & 0, & 36, & 0, & -36 \\
0, & 75, & 125, & 0, & -75, & 125 \\
-111, & 111, & 111, & 185, & 37, & -37 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-99, & 198, & 236, & 221, & -38, & 52 \\
/2 & /3 & /2 & /2 & /2 & /3 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-49, & 66, & 118, & 110, & -19, & 17 \\
\end{bmatrix}
\]

Recommend 3\textsuperscript{rd} item
ReadFood modules: Food Format

- All Reader modules return a tuple of strings: itemlist and dictratings dictionary

Translated to:

```python
dict [('Will', [0, 0, 0, 1, 5, 1, 0]), ('Samantha', [-3, 3, 0, 3, 3, 5, 1]), ('Tracy', [-3, 3, 0, 5, 3, 3, -3]), ('Xiaobai', [-1, 5, 0, 1, -1, 0, 0]), ('George', [5, 5, -3, 1, 1, -3, 0])]
```
Follow 12-step process

• ReadFood first!
  – Read input and save it
  – Get list of restaurants – use that ordering! Set?
  – For each person
    • For each restaurant and its rating
      – Must find location of restaurant in itemlist
      – Then update appropriate counter
    – Print any structure you create to check it