April 19, 2016

Prof. Rodger

Announcements

• Reading and RQ due next time
• Assign 8 due Thur., Assign 9 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

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

• Mystery(5) is \(1 + \text{Mystery}(2) = 1 + 4 = 5\)
• Mystery(2) is \(1 + \text{Mystery}(1) = 1 + 3 = 4\)
• Mystery(1) is \(1 + \text{Mystery}(0) = 1 + 2 = 3\)
• 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

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

```python
def visit(dirname):
    for inner in dirname:
        if isdir(inner):
            visit(inner)
        else:
            print name(inner), size(inner)
```

Hierarchy in Folder Structure

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
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

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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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 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 ^

• 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[a-zA]</td>
<td>Brackets create character class</td>
</tr>
<tr>
<td>(regex)</td>
<td>Tag or group a regex</td>
</tr>
<tr>
<td>\1 or \2</td>
<td>Matches previously grouped regex</td>
</tr>
<tr>
<td>{1} or {n}</td>
<td>Repeat regex 1 or n times</td>
</tr>
</tbody>
</table>

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>+?</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, \3, ...</td>
<td>match numbered tagged/grouped regex</td>
</tr>
<tr>
<td>[^...]</td>
<td>not the characters in the class, e.g., [^aeiou]</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

<table>
<thead>
<tr>
<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>5</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<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>5</td>
<td>1</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>5</td>
<td>1</td>
<td>-1</td>
<td></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 \cdot [1, 1, 0, 3, 0, -3] = -[12, 12, 0, 36, 0, -36]
- For Sam: $[0, 75, 125, 0, -75, 125]$

<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>

Then divide by number of nonzeros

\[
\begin{align*}
[12, 12, & 0, 36, 0, -36] \\
[0, & 75, 125, 0, -75, 125] \\
[-111, 111, 111, 185, 37, -37] \\
```
-99, 198, 236, 221, -38, 52
```
\end{align*}
\]

Adding lists of numbers

- Adding columns in lists of numbers
  - Using indexes 0, 1, 2, … sum elements of list
  - \( \text{sum}([\text{val}[i] \text{ for val in d.values()}) \)

ReadFood modules: Food Format

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

<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>

<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>

Recommend 3rd item
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