Reminders

• Assignments
  • Assign 5 due
  • Assign 6 live

• Spring 2022 UTA applications open
  • See Ed announcement
Key instructions

• Input ✓
• Output ✓
• Assignments* ✓
• Math/Logic ✓
• Conditionals ✓
• Repetition ✓

*not listed in book
Python Data Types

• int, float, bool ✔
• Collections
  • Strings ✔
  • Lists ✔
  • Tuples ✔
  • Sets ✔
  • Dictionaries ✔
PFTD

• Exceptions
• Recommender
  • Recommendations big picture
  • Assignment big picture
  • Simple recommendation example
  • Actual recommendation assignment
KISS Principle

• Think of the non-computing context for any word/terms
• KISS model
  • Work smarter, not harder!!
• “Good programmers are simply good designers.”
  • -Dr. Washington
• Design first and always!
• Importance of reusability
• USE PyCharm/PythonTutor IF YOU HAVE QUESTIONS!
People to Know: Frieda McAlear

- BS (Vesalius College-Brussels)
- Master of Research in Geograph (Queen Mary University of London)
- Senior Research Associate
  - Kapor Center
- Examines: 1) the barriers facing youth of color in STEM, (2) their coping strategies, and (3) programmatic interventions and resources to reduce barriers to STEM attainment.
- Co-founder-M4SJ (Mapping for Social Justice)
- Native Alaskan (Inupiaq)
Python exceptions

• What should you do if you prompt user for a number and they enter "one"
  • Test to see if it has digits?

• Exceptions make your program robust.

• Use exceptions with try: and except:
General syntax

```python
try:
    # code block that may cause the error
except errorName:
    # code that should happen if error occurs
```
Handling Exceptions

• What happens: \( x = \text{int}("123abc") \)

```python
d=['This is a test', 12, 'string', 'Blue Devils. ']

st=input("Choose 1:")
val=int(st)
if 0<=val and val<len(d):
    print(d[val])
```

• PyCharm example
Recommendation Systems: Yelp

• Are all users created equal?
• Weighting reviews

• What is a recommendation?
Recommender Systems: Amazon

• How does Amazon create recommendations?
Recommendation Systems: Netflix

• Netflix offered a prize in 2009
  • Beat their system? Win $1M
  • http://nyti.ms/sPvR
Compsci 101 Recommender

- Doesn't work at the scale of these systems, uses publicly accessible data, but ...
  - Movie data, food data, book data

- Make recommendations
  - Based on ratings, how many stars there are
  - Based on weighting ratings by users like you!

- Collaborative Filtering: math, stats, compsci
### Simple Example

- Rate restaurants on a scale of -5 to 5
- What restaurant should I choose to go to?
  - How do I decide?
- What do the ratings say? Let’s take the average!

<table>
<thead>
<tr>
<th>Tandoor</th>
<th>Fornio</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

-3 3 3 5 1 -1
Calculating Averages

<table>
<thead>
<tr>
<th>Tandoor</th>
<th>Fornio</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>-3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

- What is the average rating of each restaurant?
- Tandoor: \((1 + -3)/2 = -1.00\)
  - Don’t count rating if not rated
- Il Fornio: \((3 + 1 + 3)/3 = 2.33\)
- Where to eat? Who has the highest average?
  - McDonalds and The Loop \((8/2=4.00)\)
Python Specification

• Items: list of strings (header in table shown)

```python
items = ['Divinity Cafe', 'Farm Stead', 'Il Forno', 'Loop Pizza Grill', 'McDonalds', 'Panda Express', 'Tandoor', 'The Commons', 'The Skillet']
```

• Dictionaries: Key-value pairs are name: ratings (string: int list)

```python
ratings = 
{
  "Sarah Lee": [3, 3, 3, 3, 0, -3, 5, 0, -3],
  "Melanie": [5, 0, 3, 0, 1, 3, 3, 3, 1],
  "J J": [0, 1, 0, -1, 1, 1, 3, 0, 1],
  "Sly one": [5, 0, 1, 3, 0, 0, 3, 3, 3],
  "Sung-Hoon": [0, -1, -1, 5, 1, 3, -3, 1, -3],
  "Nana Grace": [5, 0, 3, -5, -1, 0, 1, 3, 0],
  "Harry": [5, 3, 0, -1, -3, -5, 0, 5, 1],
  "Wei": [1, 1, 0, 3, -1, 0, 5, 3, 0]
}
```
Python Specification

• Items: list of strings (header in table shown)

```python
items = ['DivinityCafe', 'FarmStead', 'IlForno', 'LoopPizzaGrill',
         'McDonalds', 'PandaExpress', 'Tandoor', 'TheCommons',
         'TheSkillet']
```

• Values in dictionary are ratings: int list
  • `len(ratings[i]) == len(items)`

```python
ratings = {
    'Sarah Lee': [3, 3, 3, 3, 0, -3, 5, 0, -3],
    'Melanie': [5, 0, 3, 0, 1, 3, 3, 3, 1],
    'J J': [0, 1, 0, -1, 1, 1, 3, 0, 1],
    'Sly one': [5, 0, 1, 3, 0, 0, 3, 3, 3],
    'Sung–Hoon': [0, -1, -1, 5, 1, 3, -3, 1, -3],
    'Nana Grace': [5, 0, 3, -5, -1, 0, 1, 3, 0],
    'Harry': [5, 3, 0, -1, -3, -5, 0, 5, 1],
    'Wei': [1, 1, 0, 3, -1, 0, 5, 3, 0]
}
```
def averages(items, ratings):

Input: items -- list of restaurants/strings
Input: ratings -- dictionary of name to ratings

  key: string, “Melanie”
  value: list of ints, \([1, 0, -1, \ldots, 1]\)
    . parallel list to list of restaurants (items)
      . k^{th} rating maps to k^{th} restaurant

Output: recommendations

  List of tuples (name, avg rating) or (str, float)
  . Sort by rating from high to low
Activity 1:
Drawbacks of Averaging

• Are all user’s ratings the same to me?
  • Weight/value ratings of people most similar to me

• Collaborative Filtering
  • How do we determine who is similar to/"near” me?

• Mathematically: treat ratings as vectors in an N-dimensional space, N = # of items that are rated
  • a.k.a. weight has higher value → closer to me
Determining "closeness"

- Calculate a number measuring closeness to me (higher number → closer)
  - I’m also a rater, "me" is parameter to function

- Function:
  - `similarities("rodger", ratings)`

- Return `[("rater1", #), ("rater2", #), ...]`
  - List of tuples based on closeness to me
  - sorted high-to-low by similarity
What's close? Dot Product

  - For [3,4,2] and [2,1,7]
    - $3 \times 2 + 4 \times 1 + 2 \times 7 = 6 + 4 + 14 = 24$

- How close am I to each rater?
- What happens if the ratings are
  - Same sign? Me: 3, -2 Other: 2, -5
  - Different signs? Me: -4 Other: 5
  - One is zero? Me: 0 Other: 4
- What does it mean when # is…
  - Big? Small? Negative?
LOOK AT VIDEO Writing similarities

- Given dictionary, return list of tuples
  
  ```python
def similarities(name, ratings):
    return [('name0', #), ...(nameN', #)]
  ```

- What is the # here?
  - Dot product of two lists
  - One list is fixed (name)
  - Other list varies (loop)

- Think: How many tuples are returned?
Collaborative Filtering

• Once we know raters "near" me? Weight them!
  • How many raters to consider? 1? 10?
  • Suppose Fran is \[2, 4, 0, 1, 3, 2\]
• What is Sam’s similarity to Fran?

\[
\begin{align*}
2 \times 0 + 4 \times 3 + 0 \times 5 + 1 \times 0 + 3 \times (-3) + 2 \times 5 &= 13
\end{align*}
\]

Sam’s ratings are \[0, 3, 5, 0, -3, 5\] * 13
Sam weighted: \[0, 39, 65, 0, -39, 65\]

<table>
<thead>
<tr>
<th></th>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</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>
Collaborative Filtering

• Once we know raters "near" me? Weight them!
  • How many raters to consider? 1? 10?
  • Suppose Fran is \([2, 4, 0, 1, 3, 2]\)

• What is Sam’s similarity to Fran?
  • \[2 \times 0 + 4 \times 3 + 0 \times 5 + 1 \times 0 + 3 \times -3 + 2 \times 5 = 13\]
  • Sam’s ratings \([0, 3, 5, 0, -3, 5]\) * 13
  • Sam’s weighted: \([0, 39, 65, 0, -39, 65]\)

<table>
<thead>
<tr>
<th></th>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</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>
What is Chris’s similarity and weights?

• Suppose Fran is \[2, 4, 0, 1, 3, 2\]
• Chris’s similarity is:

\[2 \times 1 + 4 \times 1 + 0 \times 0 + 1 \times 3 + 3 \times 0 + 2 \times (-3) = 3\]

Chris’s weighted ratings:

\[3 \times [1, 1, 0, 3, 0, -3] = [3, 3, 0, 9, 0, -9]\]
What is Chris’s similarity and weights?

- Suppose Fran is \([2, 4, 0, 1, 3, 2]\)
- Chris’s similarity is:
  \[2 \times 1 + 4 \times 1 + 0 \times 0 + 1 \times 3 + 3 \times 0 + 2 \times (-3) = 3\]
- Chris’ weighted ratings:
  \[3 \times [1, 1, 0, 3, 0, -3]\]
  \[[3, 3, 0, 9, 0, -9]\]

<table>
<thead>
<tr>
<th></th>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</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>
Steps for Recommendations

• Start with you, a rater/user and all the ratings
  • Get similarity "weights" for users: dot product
• Calculate new weighted ratings for all users
  • \([\text{weight} \times r \text{ for } r \text{ in ratings}]\)
• Based on these new ratings, find average
  • Using weighted & original average function
  • Don't use zero-ratings
• Check recommendations by … (not required)
  • Things I like are recommended? If so, look at things I haven't tried!
Recommendations

- Get new weighted averages for each eatery
  - Then find the best eatery I've never been to

```python
def recommendations(name, items, ratings, numUsers):
    return [('eatery0', #), ...('eateryN', #)]
```

Fran gets a recommendation (considering numUsers raters)

```python
rc = recommendations("Fran", items, ratings, 3)
#use this to provide evals to Fran
```
Similarities Summarized

• How do we get weighted ratings?

<table>
<thead>
<tr>
<th></th>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</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>
<tr>
<td>Fran</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

```python
def similarities(name, ratings):
    return [('name', #), ...('name', #)]
```

weights = similarities("Fran", ratings)
Making Recommendations

• How do we get weighted ratings? Call average?

```
weights = similarities("Fran", ratings)
weights = #slice based on numUsers
weightedRatings = {}. # new dictionary
for person, weight in weights:
    weightedRatings[?] = ?
```
Calculating Weighted Average

<table>
<thead>
<tr>
<th></th>
<th>Tandoor</th>
<th>IlForno</th>
<th>McDon</th>
<th>Loop</th>
<th>Panda</th>
<th>Twin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>0</td>
<td>39</td>
<td>65</td>
<td>0</td>
<td>-39</td>
<td>65</td>
</tr>
<tr>
<td>Chris</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>-9</td>
</tr>
<tr>
<td>Nat</td>
<td>-36</td>
<td>36</td>
<td>36</td>
<td>60</td>
<td>12</td>
<td>-12</td>
</tr>
<tr>
<td>Total</td>
<td>-36</td>
<td>75</td>
<td>101</td>
<td>60</td>
<td>-27</td>
<td>53</td>
</tr>
<tr>
<td>Avg</td>
<td>-36</td>
<td>37.5</td>
<td>50.5</td>
<td>60</td>
<td>-13.5</td>
<td>26.5</td>
</tr>
</tbody>
</table>

recommendations("Fran",items,ratings,2)

- Make recommendation for Fran? Best? Worst?
- Fran should eat at Loop! Even though only using Nat’s rating
  - No recommendation from Sam, so only 1 recommendation for Loop
- But? Fran has been to Loop! Gave it a 1, … McDonalds!!!! ??
Activity 2:
Implement functions in this order:

**RecommenderEngine**
1. averages(…)
2. similaries(…)
3. recommendations(…)

**RecommenderMaker**
1. makerecs(…)

**MovieReader**
1. getdata(…)

**BookReader**
1. getdata(…)

Can be implemented before Recommender stuff or after:

**RecommenderEngine** before RecommenderMaker and use TestRecommender
Function Call Ordering

• Some_Reader_Module.getdata(…)
• RecommenderMaker.makerecs(…)
  • RecommenderEngine.recommendations(…)
    • RecommenderEngine.similarities(…)
    • RecommenderEngine.averages(…)

Start with inner most call and work outwards

Test on your computer and on Gradescope as you go!
Reminders

• Work smarter, not harder
• Design first
• Get smaller parts working, then build on it
• Try to identify where you are stuck
  • Identify resources to help solve problem
• Leverage your design and PythonTutor to understand program flow of control
  • http://pythontutor.com