CS 134 Lecture 9: Nested Lists
Announcements & Logistics

- **HW 4** due Monday at 10 pm
- **Lab 4** will be released today
  - Prelab will be posted but is not due at the start of lab
  - We will review the code for the prelab together at the start of lab
- **Lab 2 graded feedback**
  - Let us know if you questions
  - **Comments** and coding style: comments (start with #) are an important part of documenting your code
  - **Comments** vs **docstrings**: docstrings document the function interface (input parameters, expected return), comments document the function body (logic used to implement the interface

Do You Have Any Questions?
Last Time

• Introduced nested for loops
  • Discussed how to trace the execution of loop
  • Use more examples of the *range* sequence type
• Reviewed the role of return statements in code
Today's Plan

- Introduce and use **nested lists**
- More examples of iteration:
  - Iterate over nested sequences and collect/filter useful statistics
- Module vs scripts
  - How to import and test functions
  - Role of the special if name is main code block
Nested Lists
Nested Lists

• Remember, any object can be an element of a list. This includes other lists!
• That is, we can have **lists of lists** (sometimes called a two-dimensional list)!
• Suppose we have a **list of lists of strings** called **myList**
Nested Lists

• Remember, any object can be an element of a list. This includes other lists!
• That is, we can have **lists of lists** (sometimes called a two-dimensional list)!
• Suppose we have a **list of lists of strings** called `myList`
• `word = myList[row][element]` (# `word` is a string)
  • `row` is index into “**outer**” list (identifies **which inner list** we want). In other words, defines the “row” you want.
  • `element` is index into “**inner**” list (identifies **which element** within the inner list). In other words, defines the “column” you want.

```python
myList = [ ['cat', 'frog'],
           ['dog', 'toad'],
           ['cow', 'duck'] ]
```

`myList[1][0] == 'dog'`
Lists and Data Types

- Python is a \textit{loosely typed} programming language
  - We don't explicitly declare data types of variables
  - But every value still has a data type!
  - It's important to make sure we pay attention to what a function expects, especially with lists and strings! (remember this in Lab 4)

- Lists of \underline{lists} of strings \textit{versus} list of strings:

```python
myList = [ ['cat', 'frog'], ['dog', 'toad'], ['cow', 'duck'] ]
myList[1][0] is 'dog'
myList = ['cat', 'frog', 'dog', 'toad', 'cow', 'duck']
myList[1][0] is 'f'
```
Sequence Operations

characters = [['Elizabeth Bennet', 'Fitzwilliam Darcy'],
              ['Harry Potter', 'Ron Weasley'],
              ['Frodo Baggins', 'Samwise Gamgee'],
              ['Julius Ceasar', 'Brutus']]

>>> len(characters)  # what is this?
4

>>> len(characters[0])  # what is this?
2

>>> characters += ['Rhett Butler', 'Scarlett O Hara']
>>> characters
[['Elizabeth Bennet', 'Fitzwilliam Darcy'],
 ['Harry Potter', 'Ron Weasley'],
 ['Frodo Baggins', 'Samwise Gamgee'],
 ['Julius Ceasar', 'Brutus'],
 'Rhett Butler',
 'Scarlett O Hara']

Be careful when concatenating lists of two different types
Looping Over Nested Lists

characters =
[['Elizabeth Bennet', 'Fitzwilliam Darcy', 'Charles Bingley'],
['Harry Potter', 'Ron Weasley', 'Hermoine Granger'],
['Frodo Baggins', 'Samwise Gamgee', 'Gandalf']]

for char_list in characters:
    print(char_list)
    for name in char_list:
        print(name)

Loops over the "outer lists"

Prints each inner list one by one

Prints each individual name one by one

Loops over the names in each "inner list"
Why Nested Lists?

• Nested Lists are useful to represent tabular data
  • Example: data stored in google sheets
• Each inner list is a row
• List of lists: collection of all rows
• Lets take an example of real data that we can store as list of lists
<table>
<thead>
<tr>
<th>MOVIE</th>
<th>20 Days in Mariupol</th>
<th>American Fiction</th>
<th>American Symphony</th>
<th>Analytical</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEST PICTURE</td>
<td>American Fiction</td>
<td>Anatomy of a Fall</td>
<td>Barbie</td>
<td>The Holdovers</td>
</tr>
<tr>
<td>BEST ACTOR</td>
<td>Maestro - Bradley Cooper</td>
<td>Rustin - Colman Domingo</td>
<td>The Holdovers - Paul Giamatti</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST ACTRESS</td>
<td>Nyad - Annette Bening</td>
<td>Killers of the Flower Moon - Lily Gladstone</td>
<td>Anatomy of a Fall - Sandra Huller</td>
<td>Macbeth</td>
</tr>
<tr>
<td>BEST SUPPORTING ACTRESS</td>
<td>Oppenheimer - Emily Blunt</td>
<td>The Color Purple - Danielle Brooks</td>
<td>Barbie - America Ferrera</td>
<td>Nyad</td>
</tr>
<tr>
<td>BEST DIRECTOR</td>
<td>Anatomy of a Fall - Justin Triet</td>
<td>Killers of the Flower Moon - Martin Scorsese</td>
<td>Oppenheimer - Christopher Nolan</td>
<td>Poor Women</td>
</tr>
<tr>
<td>BEST INTERNATIONAL FEATURE FILM</td>
<td>Io Capitano (Italy)</td>
<td>Perfect Days (Japan)</td>
<td>Society of the Snow (Spain)</td>
<td>The Holdovers</td>
</tr>
<tr>
<td>BEST ANIMATED FEATURE FILM</td>
<td>The Boy and the Heron</td>
<td>Elemental</td>
<td>Nimona</td>
<td>Bobbly</td>
</tr>
<tr>
<td>BEST PRODUCTION DESIGN</td>
<td>Barbie</td>
<td>Killers of the Flower Moon</td>
<td>Napoleon</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST CINEMATOGRAPHY</td>
<td>El Conde</td>
<td>Killers of the Flower Moon</td>
<td>Maestro</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST COSTUME DESIGN</td>
<td>Barbie</td>
<td>Killers of the Flower Moon</td>
<td>Napoleon</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST DOCUMENTARY</td>
<td>Bobi Wine: The People's President</td>
<td>The Eternal Memory</td>
<td>Four Daughters</td>
<td>To K</td>
</tr>
<tr>
<td>BEST DOCUMENTARY SHORT</td>
<td>The ABCs of Book Banning</td>
<td>The Barber of Little Rock</td>
<td>Island in Between</td>
<td>The Holdovers</td>
</tr>
<tr>
<td>BEST FILM EDITING</td>
<td>Anatomy of a Fall</td>
<td>The Holdovers</td>
<td>Killers of the Flower Moon</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST MAKEUP &amp; HAIR STYLING</td>
<td>Golda</td>
<td>Maestro</td>
<td>Oppenheimer</td>
<td>Poor Women</td>
</tr>
<tr>
<td>BEST ORIGINAL SCORE</td>
<td>American Fiction</td>
<td>Indiana Jones and the Dial of Destiny</td>
<td>Killers of the Flower Moon</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST ORIGINAL SONG</td>
<td>Flamin' Hot - &quot;The Fire Inside&quot;</td>
<td>Barbie - &quot;I'm Just Ken&quot;</td>
<td>American Symphony - &quot;It Never Went Away&quot;</td>
<td>Killers of the Flower Moon</td>
</tr>
<tr>
<td>BEST ANIMATED SHORT</td>
<td>Letter to a Pig</td>
<td>Ninety-Five Senses</td>
<td>Our Uniform</td>
<td>Pac-Man</td>
</tr>
<tr>
<td>BEST LIVE ACTION SHORT</td>
<td>The After</td>
<td>Invincible</td>
<td>Knight of Fortune</td>
<td>Red Riding Hood</td>
</tr>
<tr>
<td>BEST SOUND</td>
<td>The Creator</td>
<td>Maestro</td>
<td>Mission: Impossible - Dead Reckoning Part One</td>
<td>Oppenheimer</td>
</tr>
<tr>
<td>BEST VISUAL EFFECTS</td>
<td>The Creator</td>
<td>Godzilla Minus One</td>
<td>Guardians of the Galaxy Vol. 3</td>
<td>Mission: Impossible</td>
</tr>
<tr>
<td>BEST ADAPTED SCREENPLAY</td>
<td>American Fiction</td>
<td>Barbie</td>
<td>Oppenheimer</td>
<td>Poor Women</td>
</tr>
<tr>
<td>BEST ORIGINAL SCREENPLAY</td>
<td>Anatomy of a Fall</td>
<td>The Holdovers</td>
<td>Maestro</td>
<td>May The Fourth</td>
</tr>
</tbody>
</table>
Storing this Data

• We will defer some of the initial components:
  • How to write python code to read in the file
  • You will do this soon: in Lab 6
• For now, let's imagine we are able to store the data as follows:
  • Entire table: list of lists `oscar_data`
  • 0th row of the table: list at index 0
  • 1st row of the table: list at index 1
  • ...
  • ith row of the table: list at index i
Extracting Movie Data

- **Question.** How do we access the list of all movies?
  - It's the 0th line in the file → 0th list of our list of lists

```python
>>> movies = oscar_data[0]
```

- **Question.** How do we access the list of lists of all nominations?
  - It's the 0th line in the file → 0th list of our list of lists

```python
>>> nominations = oscar_data[1:]
```
Oscar 2024 Trivia

• Now that we have the data stored, we can find out use it to extract some useful information, e.g.
  • Finding out which movie(s) got the most nominations
    • `most_nominations(movie_list, nomination_list)`
  • Before we code, let’s figure out an algorithm for solving this problem
  • How do we solve this problem?
    • **Helper function**: count how many nominations a movie got
      • `count_nominations(movie, nomination_list)`
def count_nominations(movie, nomination_list):
    '''Function that takes two arguments: movie (str) and nomination_list (list of lists) and returns the count (int) of the number of times movie is nominated.'''

    # initialize accumulation variable
    count = 0

    # iterate over list of nominations
    for category in nomination_list:
        for nominee in category:
            # is the movie name a prefix of nomination?
            if is_prefix(movie, nominee):
                count += 1

    return count
**Exercise: most_nominations**

```python
def most_nominations(movie_list, nomination_list):
    '''Returns list of movies with most nominations'''
    most_so_far = 0  # keeps track of most # nominations
    most_list = []  # remember the movie names
    for movie in movie_list:
        num = count_nominations(movie, nomination_list)
        # found a movie with more nominations
        if num > most_so_far:
            most_so_far = num
            # remember the movie
            most_list = [movie]

    # what to do if there is a tie?
    elif num == most_so_far:
        # remember this movie as well
        most_list += [movie]

    return most_so_far
```
Modules vs Scripts
**Importing Functions vs Running as a Script**

- **Question.** If you only have function definitions in a file *funcs.py*, and run it as a script, what happens?

  ```python
  % python3 funcs.py
  ```

- For testing functions, we want to call /invoke them on various test cases, in Labs, we do this in a separate file called *runtests.py*.
  - To add function calls in *runtests.py*, we put them inside the guarded block `if __name__ == "__main__":`
  - The statements within this special guarded are only run when the file is run as a **script** but not when it is imported as a **module**
  - Let's see an example
# foo.py
# test the role of __name__ variable
print("__name__ is set to", __name__)
Takeaway: `if __name__ == "__main__"`

- If you want some statements (like test calls) to be run **ONLY when the file is run as a script**
  - Put them inside the guarded `if __name__ == "__main__"` block

- When we run our automatic tests on your functions we **import them** and this means name is NOT set to main
  - So nothing inside the guarded `if __name__ == "__main__"` block is executed

- This way your testing /debugging statements do not get in the way