# 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

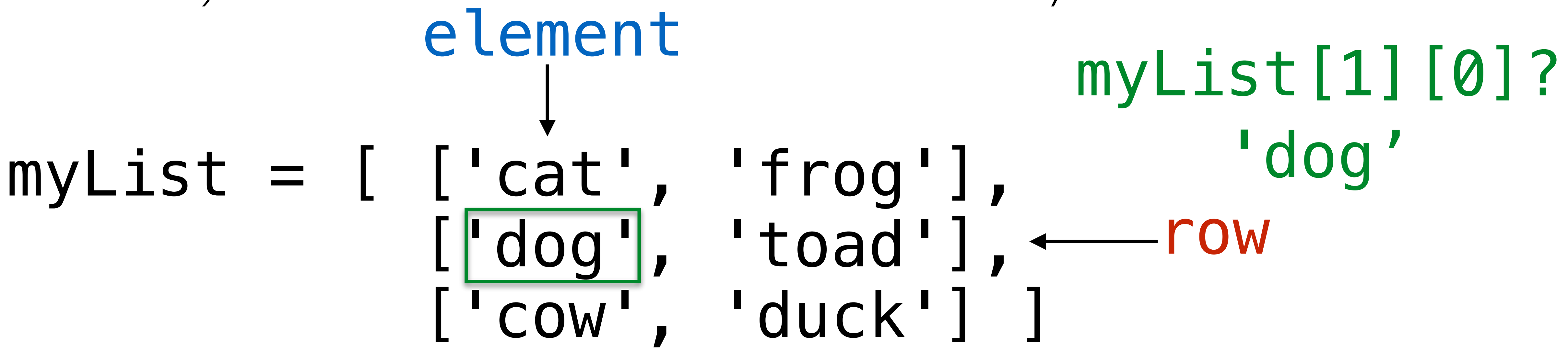
## Nested Lists

# 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.



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

# Lists and Data Types

- Python is a 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 <u>lists</u> of strings** versus **list of strings**:

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

myList[1][0] is 'dog'     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', 'Scarllet O Hara']
>>> characters
[['Elizabeth Bennet', 'Fitzwilliam Darcy'],
 ['Harry Potter', 'Ron Weasley'],
 ['Frodo Baggins', 'Samwise Gamgee'],
 ['Julius Ceasar', 'Brutus'],
 'Rhett Butler',
 'Scarllet 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

# Tabular Data: Oscars 2024

| MOVIE | 20 Days in Mariupol | American Fiction | American Symphony | Ana |
|---|---|---|---|---|
| BEST PICTURE | American Fiction | Anatomy of a Fall | Barbie | The |
| BEST ACTOR | Maestro - Bradley Cooper | Rustin - Colman Domingo | The Holdovers - Paul Giamatti | Opp |
| BEST SUPPORTING ACTOR | American Fiction - Sterling K. Brown | Killers of the Flower Moon - Robert De Niro | Oppenheimer - Robert Downey Jr. | Barb |
| BEST ACTRESS | Nyad - Annette Bening | Killers of the Flower Moon - Lily Gladstone | Anatomy of a Fall - Sandra Huller | Mae |
| BEST SUPPORTING ACTRESS | Oppenheimer - Emily Blunt | The Color Purple - Danielle Brooks | Barbie - America Ferrera | Nya |
| BEST DIRECTOR | Anatomy of a Fall - Justin Triet | Killers of the Flower Moon - Martin Scorsese | Oppenheimer - Christopher Nolan | Poo |
| BEST INTERNATIONAL FEATURE FILM | Io Capitano (Italy) | Perfect Days (Japan) | Society of the Snow (Spain) | The |
| BEST ANIMATED FEATURE FILM | The Boy and the Heron | Elemental | Nimona | Rob |
| BEST PRODUCTION DESIGN | Barbie | Killers of the Flower Moon | Napoleon | Opp |
| BEST CINEMATOGRAPHY | El Conde | Killers of the Flower Moon | Maestro | Opp |
| BEST COSTUME DESIGN | Barbie | Killers of the Flower Moon | Napoleon | Opp |
| BEST DOCUMENTARY | Bobi Wine: The People's President | The Eternal Memory | Four Daughters | To K |
| BEST DOCUMENTARY SHORT | The ABCs of Book Banning | The Barber of Little Rock | Island in Between | The |
| BEST FILM EDITING | Anatomy of a Fall | The Holdovers | Killers of the Flower Moon | Opp |
| BEST MAKEUP & HAIR STYLING | Golda | Maestro | Oppenheimer | Poo |
| BEST ORIGINAL SCORE | American Fiction | Indiana Jones and the Dial of Destiny | Killers of the Flower Moon | Opp |
| BEST ORIGINAL SONG | Flamin' Hot - "The Fire Inside" | Barbie - "I'm Just Ken" | American Symphony - "It Never Went Away" | Kille |
| BEST ANIMATED SHORT | Letter to a Pig | Ninety-Five Senses | Our Uniform | Pac |
| BEST LIVE ACTION SHORT | The After | Invincible | Knight of Fortune | Red |
| BEST SOUND | The Creator | Maestro | Mission: Impossible - Dead Reckoning Part One | Opp |
| BEST VISUAL EFFECTS | The Creator | Godzilla Minus One | Guardians of the Galaxy Vol. 3 | Miss |
| BEST ADAPTED SCEENPLAY | American Fiction | Barbie | Oppenheimer | Poo |
| BEST ORIGINAL SCREENPLAY | Anatomy of a Fall | The Holdovers | Maestro | May |

# 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, lets imagine we are able to store the data as follows:
  - Entire table: **list of lists** `oscar_data`
  - `0`th row of the table: list at index `0`
  - `1`st row of the table: list at index `1`
  - ....
  - `i`th row of the table: list at index `i`

# Extracting Movie Data

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

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

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

```
>>> nominations = oscar_data[1:]
```

Give me the 0th element
(single list)

Give the entire list of lists
excluding the 0th list

# 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, lets 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)`

## Helper Function: `count_nominations`

```
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`

```
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?

  ```
  % 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

```python
# foo.py
# test the role of __name__ variable
print("__name__ is set to", __name__)
```

Running foo.py as a **script**

```
shikhasingh@Shikhas-iMac cs134 % python3 foo.py
__name__ is set to __main__

shikhasingh@Shikhas-iMac cs134 % python3
Python 3.10.0 (v3.10.0:b494f5935c, Oct  4 2021,
14:59:20) [Clang 12.0.5 (clang-1205.0.22.11)] on
darwin
Type "help", "copyright", "credits" or "license"
for more information.
>>> import foo
__name__ is set to foo
```

Importing it as a **module**

## 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