DS2000
Fall 2021
Handout: Nested Lists

In Python, a `list` can contain another `list` as one of its elements. We can use this feature to build out complicated, "nested" structures. There's only one thing you need to remember about nested lists:

```
Rows first, columns second!
```

Keep this in the back of your mind anytime you're accessing a specific element, iterating through a nested loop, or using a shortcut to create one.

Here's how I would make a nested list with 4 rows and 3 columns (aka, a list made of 4 lists with 3 elements each).

```
nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

Like our regular list, here we have multiple values stored in the same variable. Each value has an index -- except now the index will be two numbers instead of one (row first, column second).

Nested list in memory:

	0	**1**	**2**
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

We use two sets of square braces to access individual elements in this nested list, and we refer to the row first, and column second.

```
print(nested[0][0])     # prints 1
print(nested[1][1])     # prints 5
print(nested[3][2])     # prints 12
print(nested[2][3])     # doesn't exist, IndexError
```

Iterating Over a Nested List
Every element of a nested list is itself a list. That has a few potentially-surprising consequences:

```
print(nested[0])             # prints [1, 2, 3]

len(nested)                  # length of nested is 4 (num rows)
```

```
for item in nested:
      print(item)       # prints [1, 2, 3].
                        # Then [4, 5, 6].
                        # Then [7, 8, 9].
                        # Then [10, 11, 12]
```

If you want to access every element in your nested list, you use a nested loop. The outer loop controls the current row, and the inner loop controls the current column (because *row first, column second*).

Ex: Print every element
Just like with a single list, we can iterate over element individually and simply print them out. By convention, we use *i* as the variable for an outer loop (rows) and *j* as the variable for an inner loop (columns). Here's an example where we print every element in a nested list, by exhausting the first row first, then the second row, then the third, and then the fourth.

```
for i in range(4):
        for j in range(3):
            print(nested[i][j])

# this prints 1 2 3 4 5 6 7 8 9 10 11 12
```

We can improve on this example by getting rid of those hard-coded values 4 and 3. The outer loop (*i*) iterates over the rows, and there are *len(nested)* rows. The inner loop (*j*) iterates over the columns, and there are *len(nested[i])* columns in row *i.* Therefore, we can make the above code more flexible like this:

```
for i in range(len(nested)):
        for j in range(len(nested[i])):
            print(nested[i][j])

# this ALSO prints 1 2 3 4 5 6 7 8 9 10 11 12
```

Ex: Find the maximum in a column
Suppose we have a nested list like this:

```
grades = [['hw1', 91], ['hw2', 90], ['hw3', 98]]
```

And we want to know the highest grade you've gotten on a homework. We want to ignore the first element in each row, because it's a string. We can iterate over each row and just look at the number part, which is at position 1 within each row:

```
mx = -1
for i in range(len(grades)):
        if grades[i][1] > mx:
           mx = grades[i][1]

# at the end, the value 98 will be stored in mx
```

Ex: Report the averages of all the rows

Because each nested[i] it itself a list, we can consider each row on its own. Suppose we have a nested list of move-review stars (one row represents one movie's star ratings from a bunch of reviewers):

```
movies = [[1, 5, 4], [4, 4, 2], [5, 3, 4]]
```

Each row represents one movie, and we want to know each movie's average rating. We can iterate over each row, treat each row like its own standalone list, and then report the average of those values. We begin each *j* loop by setting a running total to zero. We sum all the values in a single row (that's the *j* loop), and then once that row is exhausted, calculated the average and report it out.

```
for i in range(len(movies)):
        total = 0
        for j in range(len(movies[i])):
            total += movies[i][j]
        avg = total / len(movies[i])
        print(avg)

# this prints 3.33, 3.33, 4.0
```