CSc 110, Spring 2018

Lecture 37: List Comprehensions

Exercise

- Write a program that allows a user to ask the distance between two friends.
  - If person 1 and person 2 are friends then they are distance 0
  - If person 2 is friends with a friend of person 2 they are distance 1
- What structure is appropriate for this problem?

List comprehensions

```
[expression for element in list]
```

- A compact syntax that can replace loops that alter lists
  - Applies the expression to each element in the list
  - You can have 0 or more for or if statements

List comprehensions

```
vec = [2, 4, 6]
result = [3 * x for x in vec]
print(result)                    # [6, 12, 18]

result2 = [3 * x for x in vec if x > 3]
print(result2)                   # [12, 18]

result3 = [3 * x for x in vec if x < 2]
print(result3)                   # []
```

Notice the contents of `vec` do not change

List comprehensions

More than one element can be generated from each element in the original list

```
vec = [2, 4, 6]
result = [[x, x ** 2] for x in vec]
print(result)                  # [[2, 4], [4, 16], [6, 36]]

result2 = [(x, x ** 2) for x in vec]
print(result2)                 # [(2, 4), (4, 16), (6, 36)]
```

Exercise

- Given a list a words in any casing, create a new list containing the words with the first letter capitalized and the rest lowercase.

```
[word[0].upper() + word[1:].lower() for word in words]
```

List comprehensions

An if statement can be added to the end to allow selecting only certain elements of the original list

[expression for element in list if condition]

```
vec = [2, 3, 4, 5, 6]
result = [x for x in vec if x % 2 == 0]
print(result)                  # [2, 4, 6]

result2 = [x ** 2 for x in vec if x % 2 == 0 and x < 5]
print(result2)                 # [4, 16]
```

Exercise

- Create a list with all words from an original `text` list that are over 3 letters long

```
[word for word in text if len(word) > 3]
```

Exercise

- Count occurrences of "money" in an email text
  - We counted word occurrences earlier this semester using loops
  - Word counts can help us do things like identify spam emails

```
len([1 for word in email if word == 'money'])
```

Exercise

- Extend the solution to the last problem to count occurrences of any word that occurs in a list called `spam_words`

```
len([1 for word in email if word in spam_words])
```

Exercise

- Create a list that's contents simulates a series of 10 coin tosses (generate a 1 to represent heads, 0 for tails)

```
[randint(0, 1) for i in range(0, 10)]
```

Nested List Comprehension

- You can write a list comprehension to go over a list of lists

```
matrix = [[0,1,2,3], [4,5,6,7], [8,9,10,11]]

flattened = [i for row in matrix for i in row]

# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Set Comprehension

- Set comprehensions work just like list comprehensions except that they are surrounded by {}

```
vec = [2, 4, 6]
result = {3 * x for x in vec}
print(result)                    # {6, 12, 18}

vec2 = [2, 4, 6, 2, 2, 4, 3]
result2 = {3 * x for x in vec2}
print(result2)                   # {6, 12, 18, 9}
```

Dictionary Comprehension

- Dictionary comprehensions work similarly to list and set comprehensions except that they are surrounded by {} and generate key, value pairs

```
original = {'two' : 2, 'four' : 4, 'six' : 6}

{value: key for key, value in original.items()}
```

What does this comprehension do?