List comprehensions
(and other shortcuts)

UW CSE 160
Spring 2018
Three Ways to Define a List

• Explicitly write out the whole thing:
  
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

• Write a loop to create it:
  
squares = []
  for i in range(11):
      squares.append(i * i)

• Write a **list comprehension**:
  
squares = [i * i for i in range(11)]

• A list comprehension is a concise description of a list
• A list comprehension is shorthand for a loop
Two ways to convert Centigrade to Fahrenheit

ctemps = [17.1, 22.3, 18.4, 19.1]

With a loop:

ftemps = []
for c in ctemps:
    f = celsius_to_farenheit(c)
    ftemps.append(f)

With a list comprehension:

ftemps = [celsius_to_farenheit(c) for c in ctemps]

The comprehension is usually shorter, more readable, and more efficient
Syntax of a comprehension

\[ ((x, y) \text{ for } x \text{ in } seq_1 \text{ for } y \text{ in } seq_2 \text{ if } \text{sim}(x, y) > \text{threshold}) \]

- **expression**
  - for clause (required)
  - assigns value to the variable \( x \)
  - for clauses
  - something that can be iterated

- zero or more
  - if clauses

- zero or more additional for clauses
Semantics of a comprehension

\[
[(x, y) \text{ for } x \text{ in } \text{seq1} \text{ for } y \text{ in } \text{seq2} \text{ if } \text{sim}(x, y) > \text{threshold}]\]

result = []
for x in seq1:
    for y in seq2:
        if sim(x, y) > threshold:
            result.append((x, y))

... use result ...
Types of comprehensions

List

\[
\begin{align*}
\left[ i \times 2 \text{ for } i \text{ in range}(3) \right]
\end{align*}
\]

Set

\{
\begin{align*}
i \times 2 \text{ for } i \text{ in range}(3)
\end{align*}
\}

Dictionary

\{
\begin{align*}
key: value \text{ for } item \text{ in sequence ...}
\end{align*}
\}

\{
\begin{align*}
i: i \times 2 \text{ for } i \text{ in range}(3)
\end{align*}
\}
Cubes of the first 10 natural numbers

Goal:
Produce: \([0, 1, 8, 27, 64, 125, 216, 343, 512, 729]\)

With a loop:

```python
cubes = []
for x in range(10):
    cubes.append(x ** 3)
```

With a list comprehension:

```python
cubes = [x ** 3 for x in range(10)]
```
Powers of 2: ( $2^0$ through $2^{10}$ )

Goal: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

powers = [2 ** i for i in range(11)]
Even elements of a list

**Goal:** Given an input list \( \text{nums} \), produce a list of the even numbers in \( \text{nums} \)

\[
\text{nums} = [3, 1, 4, 1, 5, 9, 2, 6, 5]
\]

\[
\Rightarrow [4, 2, 6]
\]

\[
\text{evens} = [x \text{ for } x \text{ in } \text{nums} \text{ if } x \% 2 == 0]
\]
**Dice Rolls**

**Goal:** A list of all possible dice rolls.

**With a loop:**
```
rolls = []
for r1 in range(1, 7):
    for r2 in range(1, 7):
        rolls.append( (r1, r2) )
```

**With a list comprehension:**
```
rolls = [ (r1, r2) for r1 in range(1, 7)  for r2 in range(1, 7) ]
```
All above-average 2-die rolls

Goal: Result list should be a list of 2-tuples:

\[(2, 6), (3, 5), (3, 6), (4, 4), (4, 5), (4, 6), (5, 3), (5, 4), (5, 5), (5, 6),
(6, 2), (6, 3), (6, 4), (6, 5), (6, 6)\]

\[(r1, r2) \text{ for } r1 \text{ in } [1, 2, 3, 4, 5, 6] \text{ for } r2 \text{ in } [1, 2, 3, 4, 5, 6] \text{ if } r1 + r2 > 7\]

\text{OR}

\[(r1, r2) \text{ for } r1 \text{ in } \text{range}(1, 7) \text{ for } r2 \text{ in } \text{range}(8-r1, 7)\]
Sum of above-average 2-die rolls

**Goal:** Result list should be a list of integers:

\[
[r_1 + r_2 \text{ for } r_1 \in [1, 2, 3, 4, 5, 6] \text{ for } r_2 \in [1, 2, 3, 4, 5, 6] \text{ if } r_1 + r_2 > 7]
\]

\[
= [8, 8, 9, 8, 9, 10, 8, 9, 10, 11, 8, 9, 10, 11, 12]
\]

**Remove Duplicates:** Use Set Comprehensions

\[
\{ r_1 + r_2 \text{ for } r_1 \in \text{range}(1, 7) \text{ for } r_2 \in \text{range}(1, 7) \text{ if } r_1 + r_2 > 7\}
\]

\[
= \text{set}([8, 9, 10, 11, 12])
\]
Making a Grid

**Goal:** A grid were each element is the sum of it's row # and column #.
(e.g. \([[0, 1, 2], [1, 2, 3]\])

With a loop:

```python
grid = []
for i in range(2):
    row = []
    for j in range(3):
        row.append(i + j)
    grid.append(row)
```

With a list comprehension:

```python
grid = [[i + j for j in range(3)] for i in range(2)]
```
A word of caution

List comprehensions are great, but they can get confusing. Err on the side of readability.

```python
nums = [n for n in range(100) if sum([int(j) for j in str(n)]) % 7 == 0]
```

```python
nums = []
for n in range(100):
    digit_sum = sum([int(j) for j in str(n)])
    if digit_sum % 7 == 0:
        nums.append(n)
```
A word of caution

List comprehensions are great, but they can get confusing. Err on the side of readability.

```python
def sum_digits(n):
    digit_list = [int(i) for i in str(n)]
    return sum(digit_list)

nums = [n for n in range(100) if sum_digits(n) % 7 == 0]
```
More shortcuts!
Enumerate a list

the_list = [10 ** i for i in range(10)]
for i in range(len(the_list)):
    print str(i) + ': ' + str(the_list[i])

Or:

for index, value in enumerate(the_list):
    print str(index) + ': ' + str(value)

Like dict.items()
Enumerate a list

Goal: add each element’s index itself

```python
the_list = range(10)
new_list = []
for i, v in enumerate(the_list):
    new_list.append(i + v)
```

With a list comprehension:

```python
the_list = range(10)
new_list = [ i + v for i, v in enumerate(the_list) ]
```
Ternary Assignment

A common pattern in python

```python
if x > threshold:
    flag = "Over"
else:
    flag = "Under"

Or

flag = "Under"
if x > threshold:
    flag = "Over"
```
A common pattern in python

```python
if x > threshold:
    flag = "Over"
else:
    flag = "Under"

With a ternary expression:

flag = "Over" if x > threshold else "Under"
```
Ternary Assignment

flag = "Over" if x > threshold else "Under"

- Only works for single expressions as results.
- Only works for if and else (no elif)
Ternary Assignment

Goal: A list of 'odd' or 'even' if that index is odd or even.

```python
the_list = []
for i in range(16):
    if i % 2 == 0:
        the_list.append('even')
    else:
        the_list.append('odd')
```

or

```python
the_list = []
for i in range(16):
    the_list.append('even' if i % 2 == 0 else 'odd')
```
Ternary Assignment

Goal: A list of 'odd' or 'even' if that index is odd or even.

```python
the_list = []
for i in range(16):
    if i % 2 == 0:
        the_list.append('even')
    else:
        the_list.append('odd')
```

Or with a list comprehension!

```python
the_list = ['even' if i % 2 == 0 else 'odd' for i in range(16)]
```
Get more practice

List Comprehensions:

\[
[(x, y) \text{ for } x \text{ in seq1 for } y \text{ in seq2 if } \sim(x, y) > \text{ threshold}]
\]

Enumerate:

for index, value in enumerate(seq):
    ...

Ternary If Statement:

flag = "Over" if x > threshold else "Under"