CSC148H Week 3

Dan Zingaro

January 20, 2017

List Comprehensions

```
[<expression> for <variable> in <iterable>]
```

- The `<variable>` steps through the `<iterable>` (like a for-loop)
- For each value of `<variable>`, `<expression>` is evaluated and added to a running list

```
>>> lst = [1, 2, 3, 4, 5]
>>> [element + 5 for element in lst]
[6, 7, 8, 9, 10]
```

Exercise: List Comprehensions

Suppose we have a list of lines that each end with a newline. Create a new list of the same elements but without the newlines.

zip

```
zip(iterable1, iterable2,...)
```

- Combine the first elements of each iterable, second elements of each iterable, etc.

```
>>> list(zip([1, 2], [3, 4]))
[(1, 3), (2, 4)]
>>> for pair in zip([1, 2], [3, 4]):
...   print(pair)
...
(1, 3)
(2, 4)
```

Using Comprehensions with zip

Here, `zip` creates a list of tuples, and the comprehension multiplies corresponding elements.

```
>>> x = [1, 2]
>>> y = [3, 4]
>>> [a * b for a, b in zip(x, y)]
[3, 8]
```

Extended List Comprehensions

```
[<expression> for <variable1> in <iterable1>
  for <variable2> in <iterable2> ...]
```

A nested loop over the iterables.

```
>>> [(x, y) for x in range(2) for y in range(3)]
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

Extended List Comprehensions...

```
[<expression> for <variable> in <iterable> if <condition>]
```

`<variable>` runs through `<iterable>`, but `<expression>` is evaluated iff `<condition>` is True.

```
>>> lst = ['a', 'list', 'of', 'strs']
>>> short = [s for s in lst if len(s) <= 2]
>>> short
['a', 'of']
```

Exercise: Extended List Comprehensions

What does this do?

```
[(x, y) for x in range(5) if x % 2 == 0
        for y in range(5) if y % 2 == 1]
```

Understanding List Comprehensions

```
[(x, y) for x in range(5) if x % 2 == 0
        for y in range(5) if y % 2 == 1]
```

It helps to rewrite it using explicit for-loops:

```
lst = []
for x in range(5):
  if x % 2 == 0:
    for y in range(5):
      if y % 2 == 1:
        lst.append((x, y))
```

Exercise: List Comprehensions

Say we have a matrix (a list of lists).
Use list comprehensions to

- Extract the first column
- Multiply all elements by a scalar
- Multiply all elements pairwise with another matrix

filter

`filter` is a **functional tool**. It takes a function as a parameter.
`filter(function, iterable)`
Call `function` on each element of `iterable` and return the elements that cause the function to return True.

```
>>> def is_even(x):
...     return x % 2 == 0
...
>>> list(filter(is_even, [1, 2, 3, 4, 5]))
[2, 4]
```

any

`any(iterable)`
Returns True iff at least one element in `iterable` is True.

```
>>> any([False, False])
False
# assume is_even is defined as before
>>> any([is_even(x) for x in range(4)])
True
```

if/else ternary expression

```
if b: # if-statement
  result = x
else:
  result = y
```

is like the **expression**

```
result = x if b else y
```

```
>>> lst = [1, 2, 3, 11, 12, 13]
>>> ['even' if e % 2 == 0 else 'odd' for e in lst]
['odd', 'even', 'odd', 'odd', 'even', 'odd']
```