### List Comprehension

Jamie Saxon

Introduction to Programming for Public Policy

October 10, 2016

#### One-Line Loops

- ▶ We've already talked about lists and loops.
- ▶ Sometimes it can be useful to combine them.
- ▶ This is called 'list comprehension.' At its simplest:

```
■ print([i for i in range(10)])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- ▶ So: the thing on the left is repeated for each entry in the iterable.
- ▶ This is a very concise way to create useful lists.

#### Using the Variables

▶ You can also do simple operations, for instance, get the squares

```
■ print([i*ifor i in range(10)])
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

▶ Or perform operations on each item:

```
■ words = 'The only people for me are ...'.split()
■ print([w.upper() for w in words])
['THE', 'ONLY', 'PEOPLE', 'FOR', 'ME', 'ARE', '...']
```

This is still 'somewhat artificial.'

## Example 1: Sudoku (Homework)

- ▶ In this week's homework, you'll permanently solve Sudoku.
- ▶ If your 'puzzle' is stored in an 81-digit list, puzzle, what are the values in row, column, or block 5?
- Rows are easiest:

[puzzle[
$$5*9+x$$
] for x in range(9)]

► Columns are pretty straightforward:

▶ Blocks don't lend themselves well to this:

### Example 1: Sudoku (Homework)

- ▶ In this week's homework, you'll permanently solve Sudoku.
- ▶ If your 'puzzle' is stored in an 81-digit list, puzzle, what are the values in row, column, or block 5?
- ► Rows are easiest:

[puzzle[
$$5*9+x$$
] for x in range(9)]

► Columns are pretty straightforward:

▶ Blocks don't lend themselves well to this:

#### Example 1: Sudoku (Homework)

- ▶ In this week's homework, you'll permanently solve Sudoku.
- ▶ If your 'puzzle' is stored in an 81-digit list, puzzle, what are the values in row, column, or block 5?
- ► Rows are easiest:

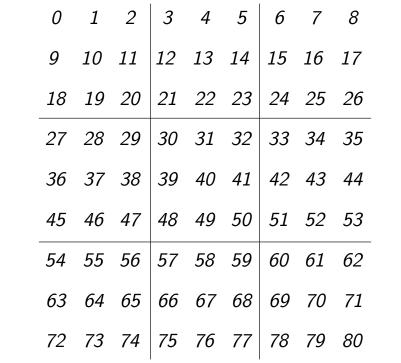
```
[puzzle[5*9+x] for x in range(9)]
```

► Columns are pretty straightforward:

```
[puzzle[5+9*x] for x in range(9)]
```

Blocks don't lend themselves well to this:

```
[puzzle[(x\%3) + (x//3)*9 + (b\%3) * 3 + (b//3)*27] for x in range(9)]
```



#### Selection

Using if, you can control selection into the lists...

```
\blacksquare num = [6, -5, -5, 10, 4, 8, -6, -5, 0, -6]
```

 $\blacksquare$  print([x for x in num if x > 0])

[6, 10, 4, 8]

▶ How could we take the intersection of two lists?

```
#!/usr/bin/env python
a = [1, 2, 3, 5, 6, 0, 3, 5]
b = [1, 3, 5, 7, 9, 11, 13]
l = [x for x in a if x in b]
print(1)
```

#### Additional 'Comprehensions'

We'll mainly see lists, but the syntax applies more broadly:

▶ Using curly brackets, {}, one can also create sets (~unique lists):

```
■ print({i//2 for i in range(10)})
0, 1, 2, 3, 4
```

▶ With parentheses, one gets a 'generator object' — lists for which each individual element is generatedd 'just in time' for you to use them:

```
■ print((i*i for i in range(10)))
<generator object <genexpr> at 0x102997a40>
```

► Generators can be very slick – feel free to learn about them!

# **Operations on Iterables**

- Anything you can put in a for loop is an iterable.
- ▶ There are a few useful functions on iterables, not specific to lists.
- ▶ max() returns the larget value.
- min() returns the smallest value.
- sum() returns the sum of all values.
- ▶ len() returns the number of elements.
- ▶ all() returns True if every element is True, and False otherwise.
- ▶ any() returns True if at least one element is True.

```
1 = [1, 0, 1.7, -1, 2.4]
print(len(1), max(1), sum(1) all(1), any(1))
```