Python List Comprehensions

by Kevin Luo

Kevin Luo

Advanced Writing

Prof Ethan Whittet

Spring 2016

Unit 3 Final Draft

Context Memo:

I'm writing an introduction to list comprehensions in python. Python is a common interpreted language used in scientific research and web development. The language features many different ways to perform similar tasks, and experienced programmers will pick between the many tools available to them based on readability and efficiency needs. List comprehensions specifically are a concise way to perform many common list operations.

Organizations that handle code often have require their members to write code in a particular, consistent style. These requirements are usually in place in order to make code more readable and cohesive. Despite the existence of general style guidelines published by governing bodies, many organizations will internally publish their own guidelines. Python is a language in particular that allows for many different syntax structures that perform the same action. The memo below is intended to be distributed to new programmers in the SMILE lab to introduce them to particular style conventions.

I wrote the instructions for intermediate python programmers. Readers familiar with mathematical syntax or the map and filter lambda functions in python notation will pick up on the syntax fairly quickly. I begin with the comparison, in hopes of allowing them to skip reading large portions of the document that would not be useful to them. Since the motivation for using list comprehensions is their succinctness and readability, I put a few comparisons to functionally identical code written in other ways. Once I give the reader an intuitive feel for how list comprehensions function, by way of some common use cases, I define the syntax explicitly and introduce a few more complex examples. For most of the examples, I provide some commentary on some appropriate stylistic decisions according the the SMILE lab conventions. As a continuation, I then introduce a few similar syntax forms in python for further reading.

Smile Lab Style Guidelines: List Comprehensions

Python list comprehensions are a concise way to compose and edit lists and other iterables that are similar to mathematical notation. For the most part, they replace the built in map and filter functions. In usage in the SMILE lab, list comprehensions should be used whenever they are more concise than other syntax structures.

Consider extracting the even numbers from the following list.

```
nums = [1, 2, 3, 4, 5, 6]
```

The following expressions are equivalent. Since the list comprehension contains the fewest characters, it is preferred.

Using a list comprehension:

```
return = [x for x in nums if x%2 == 0]
```

Using the filter function:

```
return filter(lambda x: x%2 == 0, nums)
```

Using a for loop:

```
evens = []
    for i in range(len(nums)):
        if nums[i]%2 == 0:
            evens.append(nums[i])
return evens
```

A list comprehension consists of a variable that represents the items of the newly composed list, a *for* statement, and then some number of *for* and *if* statements.

Note that list comprehension does not allow access to the index of the elements, so in some cases, it cannot replace explicit iteration through a list. In these cases, it is appropriate to use a loop structure. The following code is appropiately written with a loop structure, since it cannot be written with a list comprehension.

```
maxInd = 0
for i in range(len(nums)):
        if nums[i] > nums[maxInd]:
            maxInd = i;
return maxInd
```

The Cartesian product of two lists can be written as follows:

```
[(x, y) for x in list1 for y in list2]
```

Note how **(x, y)** replaces the functionality of python's built in *map* function, and how the dual for statements have the same effect as a nested loop

List comprehensions can include internal functions, and can operate on lists of other entities. The code below finds all seats in seats for which the function isVacant() returns true.

```
[x for x in seats if x.isVacant()]
```

Note that the **if x.isVacant()** clause replaces the functionality of python's built in *filter* function

List comprehensions can be composed of other list comprehensions. The following code transposes a matrix. In more complex cases like these, it is also appropirate provide a short inline comment describing the function of the list comprehension.

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

// transposes matrix

[[row[i] for row in matrix] for i in range(4)]
```

Similar syntax is also valid for *sets* and *dicts* in python using {} braces instead of [] braces. Please refer to the official python documentation for details on their usage.