List Comprehensions

MARCH 25TH, 2013

Examples to Get Us Started

-
  ```
  [x**2 for x in range(10)]
  [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
  ```

-
  ```
  [str(x)+str(x) for x in range(10)]
  ['00', '11', '22', '33', '44', '55', '66', '77', '88', '99']
  ```

-
  ```
  [str(x)+str(x) for x in range(10) if x%2 == 0]
  ['00', '22', '44', '66', '88']
  ```

These are all *list comprehensions*

- They provide a flexible, fast, and compact way of creating new lists from old lists.
- Anything you can do using `map` and `filter`, you can do using the list comprehension. More on this later.
- List comprehensions provide a more compact alternative to explicitly using `for`-loops.
- See Section 5.1.4 (on *List Comprehensions*) from Python v2.7.3 documentation.

List Comprehension: Basic Syntax

[*expr* for *x* in list]

Notes:

- for and in are Python keywords, used just as in for-loops.
- *x* is a variable that takes on values of elements in list, in order.
- *expr* is Python expression, typically involving the variable *x*.
- The expression [*expr* for *x* in list] evaluates to a list made up of the different values that *expr* takes on for different x.
- This is similar to the “set builder” notation used in math:

  {x*y | x and y are even}.

List Comprehensions: Syntax with `if`-clause

[*expr* for *x* in *list* if *bool-expr*]

Notes:

- *bool-expr* is a boolean expression involving *x*.
- The overall expression evaluates to a list of values of *expr* evaluated for all values of *x* in *list* satisfying the *bool-expr*.
- **Example:** `[str(x)+str(x) for x in range(10) if x%2 == 0]` evaluates to `['00', '22', '44', '66', '88']`

Examples

- Generating lists of lists.

```
[range(x) for x in range(1, 5)]
```

Evaluates to: [[0], [0, 1], [0, 1, 2], [0, 1, 2, 3]]

- All numbers in the range 0..49 containing the digit “7”.

```
[x for x in range(50) if "7" in str(x)]
```

Evaluates to: [7, 17, 27, 37, 47]

List Comprehensions and map and filter

- map(f, list) can be written as the list comprehension [f(x) for x in list].

- filter(P, list) can be written as the list comprehension [x for x in list if P(x)].

- map requires a function f, filter requires a (boolean) function P. List comprehensions can often manage with expressions.

Nested List Comprehensions

Example:

```
[x*y for x in range(3) for y in range(3)]
[0, 0, 0, 0, 1, 2, 0, 2, 4]
```

Notes:

- As in nested loops, for every iteration of the first loop (the for-x loop), all iterations of the second loop (the for-y loop) are executed.

Example: Generating Perfect Squares

```
[x for x in range(100) for y in range(x) if y*y == x]
[4, 9, 16, 25, 36, 49, 64, 81]
```

Notes:

- Those x and y values (from their respective lists) that satisfy the condition $y^2 = x$, are generated.
- Thus all x values generated in this manner are perfect squares.

Example: Generating Composites

```
composites = [x for y in range(2, 10) for x in range(2*y, 100, y)]
```

Notes:

- For each y = 2, 3,…, 9, the variable x takes on values that are multiples of y.
- For y = 2, the variable x takes on values 4, 6, 8,…, 98.
- For y = 3, the variable x takes on values 6, 9, 12,…, 99.
- Thus the values of x generated in this manner are (strict) multiples of 2, 3, 4,…, 9.
- This covers all composites in the range 2..99.

Example: Generating Prime Numbers

```
primes = [x for x in range(2, 100) if x not in composites]
```

Notes:

- Primes in the range 2..99 can be obtained by taking the complement of the generated composites.

Example: Flattening Lists

```
>>> nestedList = [range(x) for x in range(1, 4)]
>>> nestedList
>>> [[0], [0, 1], [0, 1, 2]]
>>> [y for x in nestedList for y in x]
>>> [0, 0, 1, 0, 1, 2]
```

Example: Transposing a Matrix

```
>>> mat = [[3, 0, 1],
           [2, 1, 7],
           [1, 3, 9]]

>>> [ [mat[i][j] for i in range(len(mat))] for j in range(len(mat))]
>>> [[3, 2, 1], [0, 1, 3], [1, 7, 9]]
```

Notes:

- The expression, which is the first element of the list comprehension, itself happens to be a list comprehension.
- Therefore, each element of the constructed list, is a list itself.

Warning!

- The danger with list comprehensions is that your code may become hard to understand, especially with nested list comprehensions.

- If by using a list comprehension, you are making your code hard to understand, then it is time to desist.