Introductory Python: List Comprehensions

NYC Data Science Academy
Dr. Anthony Schultz

January 10, 2017

Project folder available at:
https://github.com/Trismeg/ListComprehensions

This is an introduction to list comprehensions in Python. List comprehensions use a concise syntax, which closely mimics mathematical notation, and provide a natural way of constructing lists. They are easy to read and allow lists to be constructed, on the fly, without requiring multiple lines of code. We will learn the following:

- the syntax of list comprehensions
- how to write list comprehensions
- example applications of list comprehensions

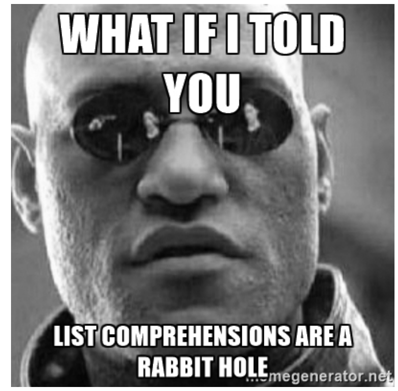


Figure 1: Using the search term "list comprehension" on Google can open up a secret programming challenge named "foobar". Pass the first three levels and get an interview with Google.

Mathematical Notation

The list L below is described using "set builder" notation.

$$L = \{\ \underbrace{\sqrt{x}}_{\text{output}}\ |\ \underbrace{x}_{\text{variable}}\ \in\ \underbrace{\mathbb{N}}_{\text{input set}}\ ,\ \underbrace{x^2 < 25}_{\text{condition}}\}$$

L is the set of $\sqrt{x}$ **for** x in the set of natural numbers $\mathbb{N}$ **if** $x^2 < 25$.

$$L = \{\ 0, \sqrt{1}, \sqrt{2}, \sqrt{3}, \sqrt{4}\}$$

x is an element of

$$x \in \mathbb{N}$$

the set of natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, 4...\}$$

In Python

We construct the set in Python, using list comprehension, as follows.

```
In [1]: [x**0.5 for x in range(100) if x**2<25]

Out[1]: [0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0]
```

This set can be constructed using a conditional statement inside a for loop.

```
In [2]: L=[]
        for x in range(100):
            if x**2<25:
                L.append(x**0.5)
        L

Out[2]: [0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0]
```

Figure 2: In computer science, ***syntactic sugar*** is syntax within a programming language that is designed to make things easier to read or to express. It makes the language "sweeter" for human use: things can be expressed more clearly, more concisely, or in an alternative style that some may prefer.

This set can also be constructed using **filter()**, **map()** and **lambda** magic.

```
In [3]: M=list(filter(lambda x: x**2 < 25, range(100)))
        L=list(map(lambda x: x**0.5, M))
        L

Out[3]: [0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0]
```

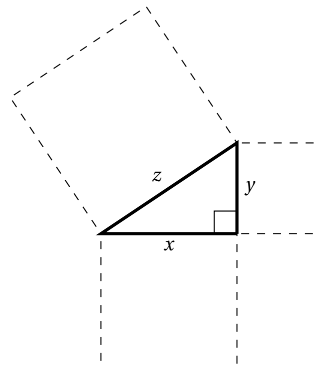
Figure 3: List comprehensions are a complete substitute for the lambda function as well as the functions map(), filter() and reduce().

Example Zero: Pythagorean Triples

This example shows how to construct Pythagorean triples. It uses three indices, x, y and z and filters for the condition $x^2 + y^2 = z^2$.

```
In [4]: P=[[x,y,z] for x in range(1,26) \
            for y in range(x,26) \
            for z in range(y,26) if x**2 + y**2 == z**2]
        P

Out[4]: [[3, 4, 5],
          [5, 12, 13],
          [6, 8, 10],
          [7, 24, 25],
          [8, 15, 17],
          [9, 12, 15],
          [12, 16, 20],
          [15, 20, 25]]
```



Pythagorean triples describe the three integer side lengths of a right triangle. The name is derived from the Pythagorean theorem.

$$x^2 + y^2 = z^2$$

Example One: 1-D Centroid Analysis

Here we use list comprehension to succinctly compute average position of a 1-D distribution. We define a list representing the position of each cell, X.

```
In [5]: mx=5
        X=[x/(mx-1) for x in range(mx)]
        X

Out[5]: [0.0, 0.25, 0.5, 0.75, 1.0]

In [6]: a=[4,5,0,0,1]
        a0=sum(a)
        aX=sum([X[i]*a[i] for i in range(mx)])
        aX/a0

Out[6]: 0.225
```

Given a list a the sum of the list is a_0.

$$a_0 = \int a(x)\, dx$$

The position weighted sum of the list is a_X.

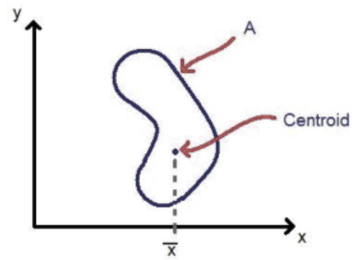
$$a_X = \int a(x)\, x\, dx$$

The center of mass, or centroid, of a is $\bar{a}_x$.

$$\bar{a}_x = \frac{a_X}{a_0}$$

Example Two: 2-D Centroid Analysis

In this example we apply centroid analysis in two dimensions. As in the previous example we construct the lists X and Y. These are now 2-D lists of lists.




```
In [7]: mx=5
        my=6
        X=[[x/(mx-1) for x in range(mx)] for y in range(my) ]
        X

Out[7]: [[0.0, 0.25, 0.5, 0.75, 1.0],
         [0.0, 0.25, 0.5, 0.75, 1.0],
         [0.0, 0.25, 0.5, 0.75, 1.0],
         [0.0, 0.25, 0.5, 0.75, 1.0],
         [0.0, 0.25, 0.5, 0.75, 1.0],
         [0.0, 0.25, 0.5, 0.75, 1.0]]

In [8]: Y=[[y/(my-1) for x in range(mx)] for y in range(my) ]
        Y

Out[8]: [[0.0, 0.0, 0.0, 0.0, 0.0],
         [0.2, 0.2, 0.2, 0.2, 0.2],
         [0.4, 0.4, 0.4, 0.4, 0.4],
         [0.6, 0.6, 0.6, 0.6, 0.6],
         [0.8, 0.8, 0.8, 0.8, 0.8],
         [1.0, 1.0, 1.0, 1.0, 1.0]]
```

Given a 2-D list A the sum of the list is A_0.

$$A_0 = \int\int A(x,y)\,dx\,dy$$

A_X is the x-position weighted sum of A.

$$A_X = \int\int A(x,y)\,x\,dx\,dy$$

A_Y is the y-position weighted sum of A.

$$A_Y = \int\int A(x,y)\,y\,dx\,dy$$

The 2-D centroid of A is $\left(\frac{A_X}{A_0}, \frac{A_Y}{A_0}\right)$.

```
def centroids(A):
    my=len(A)
    mx=len(A[0])
    A0=sum([sum([A[i][j] for i in range(my)]) for j in range(mx)])
    X=[[x/(mx-1) for x in range(mx)] for y in range(my) ]
    Y=[[y/(my-1) for x in range(mx)] for y in range(my) ]
    AX=sum([sum([A[i][j]*X[i][j] for i in range(my)]) for j in range(mx)])
    AY=sum([sum([A[i][j]*Y[i][j] for i in range(my)]) for j in range(mx)])
    Ax=AX/A0
    Ay=AY/A0
    return (Ax,Ay)
```

```
In [9]: centroids(P)

Out[9]: (0.6062091503267973, 0.6041083099906629)
```

Example Three: Natural Language Processing

```
In [10]: text = 'it was the best of times it was the worst of times \
                 yo it is nice to see you and nice to say yo yo to you \
                 say you say me say it together and say yo \
                 say yo say yo yo to me please oh please yo'
         wordlist=text.split()
         histo={}
         for i in wordlist:
             if i in histo:
                 histo[i] += 1
             else:
                 histo[i] = 1
         histo

Out[10]: {'and': 2,
          'best': 1,
          'is': 1,
          'it': 4,
          'me': 2,
          'nice': 2,
          'of': 2,
          'oh': 1,
          'please': 2,
          'say': 7,
          'see': 1,
          'the': 2,
          'times': 2,
          'to': 4,
          'together': 1,
          'was': 2,
          'worst': 1,
          'yo': 8,
          'you': 3}

In [11]: dSorted=sorted([(histo[i],i) for i in histo if histo[i] > 1])
         top=[(r[1],r[0]) for r in reversed(dSorted)]
         top

Out[11]: [('yo', 8),
          ('say', 7),
          ('to', 4),
          ('it', 4),
          ('you', 3),
          ('was', 2),
          ('times', 2),
          ('the', 2),
          ('please', 2),
          ('of', 2),
          ('nice', 2),
          ('me', 2),
          ('and', 2)]
```

In this final example we use list comprehension to process a word frequency histogram. The dictionary **histo** is constructed to record the frequency of each word in the example text.

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval. In practice, the Bag-of-words model is mainly used as a tool of feature generation. After transforming the text into a "bag of words", we can calculate various measures to characterize the text. The most common type of characteristics, or features calculated from the Bag-of-words model is term frequency, namely, the number of times a term appears in the text.

We process **histo** first by creating a list of tuples for repeated words. Restricting the list to repeated words means we filter for frequencies greater than 1. The tuples have the frequency first and the word second. This ordering allows the list to be sorted. In the second step we reverse the order of the sort, to descending, and the ordering of the tuple.