**For-loop examples:**

**Example 1:** build a list of the squares of the integers from 0 to 9:

>>> squares = [] # first start with an empty list

>>> for x in range(10): # step over every element in the list of integers from 0 to 9

squares.append(x\*\*2) #finds the squares of integers & appends it to the list

>>> print(squares)

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

**Example 2:** build a list of the lengths of the names in a list:

>>> names = ['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach']

>>> lengths = []

>>> for x in names:

lengths.append(len(x))

>>> print(lengths)

[4, 3, 3, 5, 6, 7, 4]

**Example 3:** Looping over nested sequences. suppose we want a list of all possible pairs of drink and food from the lists ['water', 'tea', 'juice'] and ['ham', 'eggs', 'spam'], respectively:

>>> for drink in ['water', 'tea', 'juice']:

for food in ['idli', 'dosa', 'samosa']:

drink\_n\_food.append([drink, food])

>>> print(drink\_n\_food)

[['water', 'idli'], ['water', 'dosa'], ['water', 'samosa'], ['tea', 'idli'], ['tea', 'dosa'], ['tea', 'samosa'], ['juice', 'idli'], ['juice', 'dosa'], ['juice', 'samosa']]

**Example 4:** filtering a sequence according to some criterion or criteria.  
suppose we want a list of the squares of the integers from 0 to 9 where the square is greater than 5 and less than 50.

>>> conditional\_squares = []  
>>> for x in range(10):

square = x\*\*2

if square > 5 and square < 50:

conditional\_squares.append(square)

>>> print(conditional\_squares)

[9, 16, 25, 36, 49]

**Example 5:** Suppose we want to take a list of names and find only those starting with the letter B:

>>> names = ['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach']

>>> b\_names = []

>>> for name in names:

if name.startswith('B'):

b\_names.append(name)

>>> b\_names

['Bob', 'Barbara']

**Map, Lambda and Filter**

One way to achieve the same goals as in the above examples is to use some of Python’s tools from functional programming: **map()**, **filter()**, and **lambda()**.

**Map():**

**Example 1 using map():**

>>> def squares(n):

return n\*n

>>> squares\_map = map(squares, range(10))

>>> squares\_map

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

**Example 2 using map():**

>>> names = ['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach']  
>>> lengths = map(len, names)

>>> lengths

[4, 3, 3, 5, 6, 7, 4]

**Example 3 using map() and lambda():**

>>> drink = ['water', 'tea', 'juice']

>>> food = ['idli', 'dosa', 'samosa']

>>> food\_n\_drink\_map = map(lambda x,y: ([x,y]), drink, food )

>>> food\_n\_drink\_map

[['water', 'idli'], ['tea', 'dosa'], ['juice', 'samosa']]

**Lambda():**

>>> print(lambda x: x\*\*2)(5)

25

# Make a function of two arguments (x and y) that returns their product, then call the function with 3 and 4

>>> print(lambda x,y: x\*y)(3,4)

12

# Incrementing a value using lambda

>>> print(lambda x: x+1)(4)

5

**Filter:**Filter takes a function returning **True** or **False** and applies it to a sequence, returning a list of only those members of the sequence for which the function returned **True**.

**Using lambda with map and filter**

**Example 4:**

>>> squares = map(lambda x: x\*\*2, range(10))

>>> conditional\_squares = filter(lambda x: x>5 and x < 50, squares)

>>> conditional\_squares

[9, 16, 25, 36, 49]

**Example 5:**

**Using lambda with filter**

>>> print(filter(lambda x: x.startswith("B"), (['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach'])))

['Bob', 'Barbara']

**using lambda with map**

>>> print(map(lambda x: x.startswith("B"), (['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach'])))

[False, False, True, False, False, True, False]

**List Comprehensions**

The simplest form of a list comprehension is

**[** expression-involving-loop-variable **for** loop-variable **in** sequence **]**

This will step over every element of sequence, successively setting loop-variable equal to every element one at a time, and will then build up a list by evaluating expression-involving-loop-variable for each one. This eliminates the need to use lambda forms, and thus generally produces a much more readable code than using [**map()**](http://docs.python.org/dev/library/functions.html#map) and a more compact code than using a for-loop.

**Solving Example 1 using list comprehension:**

>>> squares = [x\*\*2 for x in range(10)]

>>> squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

**# Print the length of each word in the list of names:**

names = ['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach']

>>> print[len(name) for name in names]

[4, 3, 3, 5, 6, 7, 4]

**# Print the last letter of each name in the list of names:**

>>> print[name[-1] for name in names]

['e', 'y', 'b', 'd', 'e', 'a', 'h']

**# Print the reverse of each name in the list of names:**

>>> print[name[::-1] for name in names]

['ennA', 'ymA', 'boB', 'divaD', 'eirraC', 'arabraB', 'hcaZ']

**Note:**

complex expressions can be put in the slot for expression-involving-loop-variable. For example,

**# here we build up a list of names, first letters, and lengths for each name in the list:**

>>> print[[name, name[0], len(name)] for name in names]

[['Anne', 'A', 4], ['Amy', 'A', 3], ['Bob', 'B', 3], ['David', 'D', 5], ['Carrie', 'C', 6], ['Barbara', 'B', 7], ['Zach', 'Z', 4]]

**Nested list comprehensions**

List comprehensions can be nested, in which case they take on the following form:

**[** expression-involving-loop-variables **for** outer-loop-variable **in** outer-sequence **for** inner-loop-variable **in** inner-sequence **]**

This is equivalent to writing:

results = []

**for** outer\_loop\_variable **in** outer\_sequence:

**for** inner\_loop\_variable **in** inner\_sequence:

results.append( expression\_involving\_loop\_variables )

**Example 3 using nested list comprehension:**

>>> food\_n\_drink = [[drink,food] for drink in ['water', 'tea', 'juice'] for food in ['idli', 'dosa', 'samosa']]

>>> food\_n\_drink

[['water', 'idli'], ['water', 'dosa'], ['water', 'samosa'], ['tea', 'idli'], ['tea', 'dosa'], ['tea', 'samosa'], ['juice', 'idli'], ['juice', 'dosa'], ['juice', 'samosa']]

**Filtered list comprehensions**

The final form of list comprehension involves creating a list and filtering it similarly to using [**filter()**](http://docs.python.org/dev/library/functions.html#filter). The filtering form of list comprehension takes the following form:

**[** expression-involving-loop-variable **for** loop-variable **in** sequence **if** boolean-expression-involving-loop-variable **]**

**Solving Example 4 using filtered list comprehension:**

>>> conditional\_squares = [x\*\*2 for x in range(10) if x\*\*2>5 and x\*\*2<50]

>>> conditional\_squares

[9, 16, 25, 36, 49]

Note that the above is inefficient, however, since it has to calculate the square of x three separate times for each element in the loop. Thus, the following is an equivalent and more efficient approach:

>>> squares = [x\*\*2 for x in range(10)]

>>> conditional\_squares = [x for x in squares if x>5 and x<50]

>>> conditional\_squares

[9, 16, 25, 36, 49]

Finally, note that the foregoing can be written on a single line using a pair of list comprehensions as follows:

>>> conditional\_squares = [x for x in [x\*\*2 for x in range(10)] if x>5 and x<50]

>>> conditional\_squares

[9, 16, 25, 36, 49]

**Example 5:**

>>> b\_names = [name for name in names if name.startswith("B")]

>>> b\_names

['Bob', 'Barbara']

**Or**

>>> b\_names = [name for name in ['Anne', 'Amy', 'Bob', 'David', 'Carrie', 'Barbara', 'Zach'] if name.startswith("B")]

>>> b\_names

['Bob', 'Barbara']