List Comprehensions

A list comprehension provides a compact way of mapping a list into another list by applying a function to each of the elements of the list.

```
a_list = [1, 9, 8, 4]
print ([elem * 2 for elem in a_list] )          #①
#[2, 18, 16, 8]

print (a_list)                                   #②
#[1, 9, 8, 4]

a_list = [elem * 2 for elem in a_list]           #③
print (a_list)
#[2, 18, 16, 8]
```

① To make sense of this, look at it from right to left. `a_list` is the list you're mapping. The Python interpreter loops through `a_list` one element at a time, temporarily assigning the value of each element to the variable `elem`. Python then applies the function `elem * 2` and appends that result to the returned list.

② A list comprehension creates a new list; it does not change the original list.

③ It is safe to assign the result of a list comprehension to the variable that you're mapping. Python constructs the new list in memory, and when the list comprehension is complete, it assigns the result to the original variable.

You can use any Python expression in a list comprehension.

You can use any Python expression in a list comprehension, including the functions in the os module for manipulating files and directories.

```
import os, glob
print (glob.glob('*.xml') )                              #①
#['feed.xml']


print ([os.path.realpath(f) for f in glob.glob('*.xml')])  #②
#['/usercode/feed.xml']
```

① This returns a list of all the `.xml` files in the current working directory.

② This list comprehension takes that list of `.xml` files and transforms it into a list of full pathnames.

List comprehensions can also filter items, producing a result that can be smaller than the original list.

```
import os, glob
print ([f for f in glob.glob('*.py') if os.stat(f).st_size < 6000])  #①
#['customserializer.py', 'romantest3.py', '__ed_file.py', 'romantest1.py',
# 'humansize.py', 'pickleversion.py', 'plural6.py', 'stdout.py',
# 'fibonacci2.py', 'roman8.py', 'alphametics.py', 'roman3.py', 'oneline.py',
#'roman1.py', 'roman2.py', 'roman10.py', 'roman9.py', 'romantest2.py']
```

① To filter a list, you can include an if clause at the end of the list comprehension. The expression after the `if` keyword will be evaluated for each item in the list. If the expression evaluates to `True`, the item will be included in the output. This list comprehension looks at the list of all `.py` files in the current directory, and the `if` expression filters that list by testing whether the size of each file is smaller than 6000 bytes. There are 18 such files, so the list comprehension returns a list of 18 filenames.

All the examples of list comprehensions so far have featured simple expressions — multiply a number by a constant, call a single function, or simply return the original list item (after filtering). But there’s no limit to how complex a list comprehension can be.

```
import os, glob
print ([(os.stat(f).st_size, os.path.realpath(f)) for f in glob.glob('*.xml')] )  #①
#[(2796, '/usercode/feed.xml')]
```

```
import humansize
print ([(humansize.approximate_size(os.stat(f).st_size), f) for f in glob.glob('*.xml')])  #②
#[('2.7 KiB', 'feed.xml')]
```

① This list comprehension finds all the `.xml` files in the current working directory, gets the size of each file (by calling the `os.stat()` function), and constructs a tuple of the file size and the absolute path of each file (by calling the `os.path.realpath()` function).

② This comprehension builds on the previous one to call the approximate_size() function with the file size of each `.xml` file.