List Comprehension: list processing without loops

In Python, the list processing is usually done with an alternative mechanism of the language that shortens the code and becomes more efficient: the list comprehension. This guide shows an overview to their use.

Part 1. Simple Comprehension

List comprehension allows to create lists on the fly, without the use of loops and/or conditionals. Instead, a combination of loops and conditionals condexed sintax is used.

There are several methods to créate lists on the fly. The more simple case has the following sintax:

```
[<expression> for <element> in <iterable>]
```

where:

- *<`expression`> is a contant or variable o any expresión in general,*
- *<`element`> is a list comprehension local variable built and destroyed automatically,*
- *<`iterable`> is an iterable: a range, a list, a str, etc.*

In all the cases, a new list is created ready for its use.

Example:

```
a=[i for i in range(6)]
```

creates the list [0,1,2,3,4,5] stored in variable a. Yes, we know is easier without list comprehension using a=list(range(6)).

Example:

```
b=[1/(x+1)**0.5 for x in range(100)]
```

creates a list (stored in b) with the inverse of the square roots of teh number from 1 to 100. Recall the code needed in case of using loops:

```
b=[]
for x in range(100):
    b.append(1/(x+1)**0.5)
```

Example:

```
c=[(1+1j)**n for n in range(5)]
```

creates a list with the first five Powers of the complex number `1+1j`.

Example:

```
product=sum([a*b for a,b in zip(v1,v2)])
```

is the scalar producto of vectors v1 and v2. Note that zip(v1, v2) creates an iterable that includes a pairo f elements (one from v1 and another form v2).

Example:

```
outer=[[a*b for a in v1] for b in v2]
```

is the outer product of two vectors nesting two list comprehension.

Example:

```
smaller=[min(a,b) for a,b in zip(v1,v2)]
```

stores in smaller a list with the smaller of two elements, one from each list. Again, zip is used.

Example:

```
init=[s[0] for s in ["esto","es","una","prueba"]]
```

stores in init a list with the initial carácter from each str.

Example:

```
import math
#. . .
factorial=round(exp(sum([log(i) for i in range(1,n+1)])))
```

computes the factorial of a number n. In this case, it is computed as the inverse of the logarithm (exp) of the addition of the logarithms.

Example:

```
lista_1=[1,2,3,4]
lista_2=[10,20,30,40]
lista_3=[100,200,300,400]
suma=[a+b+c for a,b,c in zip(lista_1,lista_2,lista_3)]
```

sums the three lists.

Example:

```
lista=[50,-22,10,60,30,-18]
[print("number:",x) for x in lista]
[print("number",i+1,":",lista[i]) for i in range(len(lista))]
[lista.pop() for i in range(len(lista))]
```

This code snippet prints the list twice and then erases all the elements.

The list comprehension returns lists but they are not worth storing; function print or method pop are used instead. The last case returns the inverse of the original list (emptying this latter).

Example:

Given a matrix as matriz=[[1,2,3,4],[5,6,7,8],[9,10,11,12]], the sum of all its elements would be:

```
suma=sum([sum(matriz[i]) for i in range(len(matriz))])
```

Using list comprehensions with data from a file

Data from and to a file can be easily processed using list comprehension. Remember that when Reading a file,the typical process is to store an str per file's line. So imagine numbers is the list of str we have after reading a file; then the following code switch it to a list of floats:

```
numbers=["123\n","68.5\n","-4.1\n","9.9\n"]
numbers=[float(x) for x in numeros]
```

The list comprehension generates the list of floats from a list of strs, storing it in numbers.

Part 2. Conditional comprehension.

Conditionals can be introduced in the list comprehension expression to only include those elements that hold a certain condition. The syntax is:

```
[<expressionA> for <elemento> in <iterable> if <boolean-
expression>]
```

Example:

```
lista=[50,-22,10,60,30,-18]
negatives=[x for x in lista if x<0]
```

only the negative values are considered to generate the list in the second expresión.

The equivalent for-based code would be:

```
negatives=[]
for x in lista:
    if x<0:
        negatives.append(x)
```

Example:

```
n_negatives=sum([1 for x in lista if x<0])
```

counts the number of negative numbers in a list.

Example:

What about getting ride off the negative numbers?

```
[lista.remove(x) for x in lista if x<0]
[lista.remove(x) for x in list(lista) if x<0]
[lista.remove(x) for x in lista[:] if x<0]
[lista.remove(x) for x in lista+[] if x<0]
```

Each of these expressions produces a negative numbers free list lista, we do not need to store the outcome of the list comprehension as long as we are using remove, hence, modifying the list. The three latter creates a copy of the list in case we would like to develop further to generate a new list.

Example:

```
lista=["hola",12,6.7,[],True,1.5,-1]
numeros=[x for x in lista if type(x)==int or type(x)==float]
```

creates a list including the numerical ítems only.

Example:

```
[print(n) for n in range(1,1000) if n==sum([i for i in
range(1,n) if n%i==0])]
```

Shows the perfect numbers from 1 to 1000.

Part 3. The complete conditional comprehension.

We can use if-else conditionals as well; in this case, the syntax is:

```
[<expression1> if <boolean-expression> else <expression2> for
<element> in <iterable>]
```

where

- *<expression1> is the value to insert within the new list whenever boolean-expression holds,*
- *<expression2> is the value to insert within the new list otherwise.*

Example:

```
no_negatives=[x if x>=0 else 0 for x in lista]
```

creates a list with a 0 wherever a negative number appears in lista.

Example:

```
my_str="This is a test"
no_vowels="".join([c for c in my_str if c not in
"AEIOUaeiouÁÉÍÓÚáéíóúü"])
```

get ride off vowels.

Example:

```
no_vocales="".join([c if c not in "AEIOUaeiouÁÉÍÓÚáéíóúü" else
"_" for c in my_str])
```

includes an underscore char ('_') instead of a vowel in the new generated string.

Example:

We can nest conditionals within a list comprehension. Imagine we want to obtain a list with the type of data for a list, but ignoring non numerical values marking them as 'not-your-business':

```
lista=["hi",12,6.7,[],True,1.5,-1]
types=["int" if type(x)==int else "float" if type(x)==float else
"not-your-business" for x in lista]
```

Example:

```
mid-value = [x if y>x>z or y<x<z else y if x>y>z or x<y<z else z
for x,y,z in zip(p,q,r)]
```

creates a list with the value in the middle among three values, one per an independent list.

Part 4. Nesting list comprehensions.

We need nesting whenever we need to anylize more than one dimension, e.g., matrices.

Example:

Let m be a matrix, create a copy flipping the sign of the values. Indexes i and j refers to rows and columns respectively.

```
m=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
n=[[-row[j] for j in range(len(row))] for row in m]
```

Example:

Matrix transposes can be easily computed generating a new row for column… nevertheless, this code asumes the matrix is well formed (all the rows have the same dimension)!!!

```
transpose=[[row[j] for row in m] for j in range(len(m[0]))]
```

Example:

Creating a matrix from another but repeating each row twice.

```
new=[[row[j] for j in range(len(row))] for row in m for n in
range(2)]
```

or, a more simple method:

```
new=[row for row in m for n in range(2)]
```

Example:

Duplicating the rows but repeating the complete matrix:

```
new=[fila for n in range(2) for fila in m]
```

Example:

Repeating rows and columns:

```
new=[[row[j] for j in range(len(row)) for k in range(2)] for row
in m for n in range(2)]
```

There is no limit in nesting but readability is the main thing when writing Python code; therefore, it is suggested than two nested list comprehension is enough…

Example:

Printing a matrix to the standard output! Again, the matrix should have all the rows with the same dimension.

```
[print(m[i][j],end=" ") if j<len(m[0])-1 else print(m[i][j]) for
i in range(len(m)) for j in range(len(m[0]))]
```

Or:

```
[print(row[j],end=" ") if j<len(row)-1 else print(row[j]) for
row in m for j in range(len(m[0]))]
```

Example:

Again but including formatted str:

```
[print("{0:3d}".format(row[j]),end="") if j<len(row)-1 else
print("{0:3d}".format(row[j])) for row in m for j in
range(len(row))]
```