Listas por comprensión: procesado de listas sin codificación de bucles

En Python, el procesado de listas puede ser realizado mediante un mecanismo que, además de reducir el código a escribir, resulta en una ejecución eficiente: las listas por comprensión o list comprehensions. Este guion pretende dar una visión del diseño de este tipo de código.

Parte 1. Comprensión simple.

Las listas por comprensión permiten crear listas al vuelo sin necesidad de emplear instrucciones de bucles o condicionales. En su lugar se emplean bucles y condicionales implícitos o internos que no son propiamente instrucciones del lenguaje.

Hay varias formas de crear listas al vuelo. La primera tiene esta sintaxis genérica:

```
[<expresion> for <elemento> in <iterable>]
```

donde:

- *<expresion> es una constante o variable o cualquier expresión en general,*
- *<elemento> es una variable interna implícita que también se crea y se destruye de forma automática,*
- *<iterable> es cualquier cosa sobre la que se pueda iterar, por ejemplo una función como "range", otra lista, una cadena de caracteres, una tupla, un fichero, un conjunto, un diccionario o cualquier tipo de datos semejante.*

En todos los casos se genera una lista nueva sobre la que por supuesto se puede realizar cualquier otra operación.

Ejemplo:

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

crea la lista "a" que vale [0,1,2,3,4,5] aunque en este caso no haría falta emplear las listas por comprensión puesto que sería más breve hacerlo con a=list(range(6)).

Ejemplo:

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

crearía una lista "b" que contendría los inversos de las raíces cuadradas de los números del 1 al 100. La forma de hacer este último ejemplo sin usar listas por comprensión sería:

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

Ejemplo:

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

crea una lista con las cinco primeras potencias del número complejo `1+1j`.

Ejemplo:

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

realiza el producto escalar de dos vectores v1 y v2. Observar como `zip(v1,v2)` genera un iterable conteniendo pares de elementos, uno de v1 y otro de v2.

Ejemplo:

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

realiza el producto externo de dos vectores por medio de la anidación de dos comprensiones de listas.

Ejemplo:

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

asigna a menores una lista con el menor de los elementos de dos listas. Nuevamente, se utiliza la función `zip`.

Ejemplo:

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

retorna la lista con los caracteres iniciales de cada cadena de caracteres.

Ejemplo:

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

calcula el factorial de un número "n". El factorial es un producto de enteros crecientes que aquí se calcula como el antilogaritmo de la suma de logaritmos.

Ejemplo:

```
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)]
```

realiza la suma de las tres listas.

Ejemplo:

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

El fragmento de código anterior imprime la lista por la salida estándar dos veces y finalmente se eliminan todos los elementos de la lista.

Las tres últimas líneas devuelven listas pero no interesa lo que devuelven sino que se ejecute la función "print" o el método "pop". En el último caso si se desease utilizar la lista resultante sería la reversa de la original. Con [lista.pop(0) for i in range(len(lista))] se obtendría una copia de la original pero se vaciaría de elementos esta última.

Ejemplo:

Sea "matriz" una lista de listas: matriz=[[1,2,3,4],[5,6,7,8],[9,10,11,12]], la suma de todos sus elementos sería:

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

Uso de listas por comprensión con datos en ficheros

Los datos provenientes de la lectura de un fichero se pueden almacenar en una lista con una cadena por línea en el fichero, cada cadena contendrá el salto de línea. Supongamos que en `numeros` *tenemos el resultado de leer un fichero, el siguiente código intentará convertir cada elemento leído a flotante.*

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

Obsérvese que primero se crea la lista [float(x) for x in numeros] y después se asigna a la variable "numeros".

Parte 2. Comprensión condicionada.

La segunda forma de usar listas por comprensión de permite utilizar condicionales, de manera que solo una serie de elementos que cumplan la condición sean procesados. En este caso, se emplea esta la siguiente sintaxis:

```
[<expresion> for <elemento> in <iterable> if <expresion-
booleana>]
```

Ejemplo:

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

genera la lista [-22,-18] al ser éstos los únicos valores negativos.

Utilizando instrucciones de bucles y condicionales habría que hacerlo así:

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

Ejemplo:

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

cuenta la cantidad de negativos.

Ejemplo:

¿Y si se desease quitar los negativos de esa misma lista? Cualquiera de estas opciones sirve:

```
[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]
```

De nuevo en este caso la lista resultante no interesa, lo que importa es ejecutar la expresión lista.remove(x). Obsérvese que el bucle debe recorrer una copia de la lista ya que se está modificando la original, de ahí el uso de las distintas formas de obtener una copia de la lista original.

Ejemplo:

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

almacena en `numeros` los elementos de `lista` que cumplan la condición de ser enteros o reales.

Ejemplo:

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

muestra los `números` perfectos menores de 1000.

Parte 3. Comprensión condicionada completa.

En el caso de que haya que emplear un condicional completo `if-else`, la sintaxis cambia ligeramente:

```
 [<expresion1> if <expresion-booleana> else <expresion2> for
<elemento> in <iterable>]
```

donde

- *<expresion1> es el valor que se inserta en la lista si la expresión booleana es cierta,*
- *<expresion2> es el valor que se inserta en la lista si la expresión booleana es falsa.*

Ejemplo:

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

genera una nueva lista cambiando negativos por ceros.

Ejemplo:

```
cadena="Esto es una prueba"
sin_vocales="".join([c for c in cadena if c not in
"AEIOUaeiouÁÉÍÓÚáéíóúü"])
```

elimina las vocales de una cadena de caracteres.

Ejemplo:

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

cambia las vocales por guiones bajos.

Ejemplo:

Es posible enlazar condicionales en las listas por comprensión. Si se desea obtener una lista que indique los tipos de datos que hay en otra seleccionando enteros, reales y otros:

```
lista=["hola",12,6.7,[],True,1.5,-1]
tipos=["entero" if type(x)==int else "real" if type(x)==float
else "otro" for x in lista]
```

Ejemplo:

```
en_medio=[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)]
```

genera una lista formada con el valor central entre tres elementos, cada elemento proveniente de una lista independiente.

Parte 4. Bucles anidados.

Es posible anidar listas por comprensión, por ejemplo para trabajar con diferentes dimensiones en matrices.

Ejemplo:

Dada una matriz, crear otra que sea la original pero con el signo de sus elementos cambiado. En este ejemplo "fila" recorre cada fila de la matriz y "j" es el índice de columna.

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

Ejemplo:

Para determinar la traspuesta de una matriz se podría ir recorriendo cada columna (bucle de la "j") y formando una nueva fila con los elementos de cada columna. En el ejemplo "fila" se refiere a la fila de la matriz original.

```
traspuesta=[[fila[j] for fila in matriz] for j in
range(len(matriz[0]))]
```

Ejemplo:

Para generar una matriz repitiendo cada fila 2 veces:

```
nueva=[[fila[j] for j in range(len(fila))] for fila in matriz
for n in range(2)]
```

o, de forma más sencilla:

```
nueva=[fila for fila in matriz for n in range(2)]
```

Ejemplo:

Para duplicar todas las filas pero manteniendo el orden:

```
nueva=[fila for n in range(2) for fila in matriz]
```

Ejemplo:

Para duplicar filas y columnas sería:

```
nueva=[[fila[j] for j in range(len(fila)) for m in range(2)] for
fila in matriz for n in range(2)]
```

No hay límite al número de bucles implícitos anidados, pero en Python debe primar la legibilidad y con más de dos ya empieza a ser complicado mantenerla.

Ejemplo:

Por supuesto pueden emplearse formas más complejas, como el siguiente caso. Para mostrar una matriz por la salida estándar:

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

o bien:

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

Ejemplo:

Como en el caso anterior, pero con salida formateada:

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