listas por comprension

March 10, 2018

1 Compresion de listas

Las listas por comprension podriamos decir que son formas concisas y simplificadas de escribir bucles **for**, tambien se puede decir que es una forma mas natural de entender los bucles e iteradores.

En este tutorial se va a usar indistintamente la expresion **comprension de lista** como **lista por comprension**

Veamos un ejemplo, si quiero imprimir en pantalla los cuadrados de 1 a 10 podemos hacerlo de la manera tradicional:

```
In [1]: numeros = range(1,10)
        for numero in numeros:
            print(numero**2)
```

```
1
4
9
16
25
36
49
64
81
```

Veamos ahora como se hace con una lista por comprension, es importante aclarar que la comprension no solo se aplica a listas sino a cualquier tipo de coleccion iterable de python, se llama lista por comprension porque devuelve una lista.

```
In [2]: print([numero**2 for numero in range(1,10)])
```

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

1.1 Condiciones en las listas por compresion

1.1.1 Condicional IF

Tambie podemos usar el condicional **if** para filtrar los elementos de una coleccion con la que iteramos, la forma es simple de escribir y comprender:

[expresion **for** elemento **in** coleccion_iterable **if** condicion]

Veamos un ejemplo usando el metodo tradiciona de obtener solo los cuadrados de los numeros pares:

```
In [5]: numeros = range(1, 10)
        for numero in numeros:
            if numero%2 == 0:
                print(numero)
```

```
2
4
6
8
```

Veamos ahora como lo haria usando un condicional en el iterador de la coleccion:

```
In [6]: print([numero for numero in range(1, 10) if numero%2 ==0])
```

```
[2, 4, 6, 8]
```

1.1.2 Condicional ** if else**

En este caso, el orden de los enunciados dentro de la comprensión de lista será diferente de condicionales if simples. Cuando solo tienes una condición if, la condición va al final de la comprensión. Sin embargo, en el caso de una expresión **.. if .. else ..** las posiciones para el bucle for y la expresión condicional son intercambiadas. El nuevo orden es:

[expresion **if** condicion **else** otra_expresion **for** elemento **in** coleccion_iterable]

Veamoslo en un ejemplo, para obtener los cuadrados de los números pares y los cubos de los números impares de un rango de 1 a 10 se hace, de la manera tracional con el siguiente codigo:

```
In [9]: cuadrados_cubos = []

        for numero in range(1,10):
            if numero%2 == 0:
                cuadrados_cubos.append(numero**2)
            else:
                cuadrados_cubos.append(numero**3)

        print(cuadrados_cubos)
```

```
[1, 4, 27, 16, 125, 36, 343, 64, 729]
```

Ahora vamos a hacerlo usando una comprension de lista con el condicional **.. if .. else ..**:

```
In [11]: cuadrados_cubos = [numero**2 if numero%2==0 else numero**3 for numero in r
         print(cuadrados_cubos)
```

```
[1, 4, 27, 16, 125, 36, 343, 64, 729]
```

1.2 Bucles anidados for

Se pueden anidar bucles dentro de listas por comprension, al igual que los bucles for tradicionales no existen limites en la cantidad de anidaciones, adicionalmente se puede poner un condicional **if** en cada lista, la forma en que se expresa es la siguiente:

[expresion **for** elemento **in** coleccion (opcional **if** condicion) **for in** coleccion (opcional **if** condicion) **for in** coleccion (opcional **if** condicion) ... otro bucle ...]

Supongamos que queremos una lista con las tablas de multiplicar hasta 10 * 10:

```
In [3]: tabla = []

        for multiplicando in range(1, 11):
            for multiplicador in range(1, 11):
                tabla.append(multiplicando*multiplicador)

        print(tabla)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 3, 6, 9, 12, 15
```

Una vez que has escrito el bucle en esta forma, convertirlo a una comprensión de lista es fácil:

```
In [4]: tabla = [multiplicando*multiplicador for multiplicando in range(1,11) for m

        print(tabla)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 3, 6, 9, 12, 15
```

2 Anidacion de listas por comprension

La anidacion de listas por comprension aparentemente seria lo mismo que listas con bucles anidados, pero no es asi, en realidad son dos cosas muy diferentes, en el caso de bucles anidados son bucles for dentro de otros bucles for, en este caso son listas dentro de listas.

Veamos un ejemplo asi lo podemos entender mejor, veamos como hacer la transposicion de una matriz:

Con el metodo tradicional hariamos:

```
In [6]: matriz = [
            [1, 2, 3, 4],
            [5, 6, 7, 8],
            [9, 10, 11, 12],
        ]

        transposicion = []

        for i in range(4):
            temp = []
            for row in matriz:
```

```
                temp.append(row[i])
            transposicion.append(temp)

        print(transposicion)
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Ahora, usando listas anidadas:

```
In [7]: matriz = [
            [1, 2, 3, 4],
            [5, 6, 7, 8],
            [9, 10, 11, 12],
        ]

        transposicion = [[row[n] for row in matriz] for n in range(4)]

        print(transposicion)
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

2.1 Conclusiones

Espero que les sirva, la mejor forma de fijar estos conocimientos (al menos a mi me funciona) es practicar y practicar, asi que voy a preparar una serie de ejercicios para quienes quieran resolverlos y asi acostumbrarse a este metodo que le vas a reducir considerablemente el codigo a escribir y resulta una mas natura de pensar.

Prof. Luis Tomas Wayar