Python Comprehensions (Comprensiones)

¡Comencemos!

¿Qué aprenderás?

- Codificar un programa en Python utilizando ciclos de instrucciones iterativas combinadas con sentencias if/elif/else para resolver un problema.

Introducción

Un aspecto bastante único de Python son las llamadas Python Comprehensions, las que son una manera de reescribir un ciclo `for` en una sola línea. Esta característica, única en este lenguaje, tiene la ventaja de reducir y compactar el código en el caso de operaciones cortas, lo cual es una característica propia de los programadores experimentados.

Es por esto, que en este capítulo aprenderemos cómo traducir ciclos `for` en comprehensions y algunos casos específicos de uso.

¡Comencemos!

Transformando un ciclo for en un Comprehension

Supongamos el siguiente problema: queremos generar los 10 primeros números pares y almacenarlos en una lista:

```
import sys
# solicitamos el número de pares a generar
n = 10

# generamos una lista vacía para almacenar los pares
lista_par = []

for i in range(n):
      # podemos hacer append de los valores generados
      # en este caso partimos desde el 2
      lista_par.append(2*i + 2)

# mostramos el resultado
print(lista_par)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Antes aplicar comprehension veamos la sintaxis a utilizar:

```
[fórmula for variable in iterable]
```

Esta sintaxis se lee de la siguiente manera:
Empezamos por el ciclo, entonces, para cada variable en el iterable realiza la fórmula

El código, en términos de comprehension se puede escribir de la siguiente manera:

```
lista_par = [2*i + 2 for i in range(n)]
print(lista_par)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Se puede ver que en solo dos líneas se obtiene el mismo resultado (de hecho es posible dejarlo sólo en una línea); y lo que acabamos de ver es un List Comprehension, porque el resultado final se almacena en una lista.

Haciendo la lectura del código `[2*i + 2 for i in range(n)]` decimos entonces que: Para cada i en el rango de n (10) voy a guardar el resultado de la operación 2*i + 2. En la primera iteración como i vale 0 se realizará la multiplicación `2*0 + 2`, en la siguiente iteración i vale 1 `2*1 +2`, en la siguiente i vale 2 `2*2 + 2`, etc

Condicionales con List Comprehensions

Existen otras versiones de un List Comprehension que permiten combinar Control de Flujo condicional con los ciclos `for`.

Por ejemplo, supongamos la siguiente lista de valores, donde queremos entregar una lista que nos diga si el número es divisible por dos o no.

```
valores = [0,4,5,6,7,8,9]
divisibles = []
for valor in valores:
      if valor % 2 == 0:
          lista.append('Divisible')
      else:
          lista.append('No Divisible')
```

List comprehensions tiene la siguiente sintaxis que equivale a lo mostrado anteriormente:

```
[expresión1 if condición1 else expresión2 for variable in iterable]
```

En este caso, si convertimos el código anterior en List comprehension queda de la siguiente manera:

```
['Divisible' if valor % 2 == 0 else 'No Divisible' for valor in valores
]
```

Ambos códigos devuelven el siguiente resultado:

```
['Divisible',
 'Divisible',
 'No Divisible',
 'Divisible',
 'No Divisible',
 'Divisible',
 'No Divisible']
```

¿Cómo se lee este código?

```
['Divisible' if valor % 2 == 0 else 'No Divisible' for valor in valores ]
```

Empezamos con el ciclo, entonces, para cada valor almacenado en nuestra lista de valores, vamos mostrar la palabra 'Divisible' **si** el valor módulo de 2 es igual a 0.

A continuación, te presentamos un ejercicio a partir del cual podrás poner en práctica lo aprendido siguiendo los pasos anteriormente explicados.

Operaciones de Filtrado

Finalmente, los List Comprehensions pueden ser utilizados para filtrar datos de una lista. Por ejemplo, contemos la cantidad de datos de esta lista que son string:

```
lista = ['Lechugas', 'Tomates', 5, 10,
                 True, False, True, 'Papas',
                 5.1, 45.2, 1, 2, 0]
count_str = 0
for elemento in lista:
      if type(elemento) is str:
      count_str += 1

print(count_str)
```

```
3
```

NOTA: En la prueba lógica se utiliza la palabra `is`, la que sirve para poder detectar si un cierto tipo de dato es el que se espera, en este caso si el tipo del elemento es un `str` (Un string).

Es posible utilizar una tercera alternativa de sintaxis de los List Comprehensions:

```
[expresión for variable in iterable if condición ]
```

Esta expresión se aplicará sólo a los valores que cumplan la condición dada. Por ejemplo:

```
[elemento for elemento in lista if type(elemento) is str ]
```

Luego para contar el resultado podemos utilizar la función `len()`, de manera que el código completo se reduce a lo siguiente:

```
lst_str = [elemento for elemento in lista if type(elemento) is str ]
print(len(lst_str))
```

Dictionary Comprehensions

Los Dictionary Comprehensions son la equivalencia en Diccionarios de los Python Comprehensions. Recuerda que los diccionarios son estructuras clave-valor por lo que se debe considerar al momento de iterar.

```
claves = ['nombre','apellido','edad','altura']
valores = ['Juan','Pérez', 33, 1.75]
print({k:v for k,v in zip(claves, valores)})
```

```
{'nombre': 'Juan', 'apellido': 'Pérez', 'edad': 33, 'altura': 1.75}
```

Un dictionary comprehension seguirá exactamente las mismas reglas y estructuras que una List Comprehension pero utilizando llaves (`{}`) y `:` como separador entre la clave y el valor.

NOTA: Normalmente, se utilizan las variables `k` para las claves (*key*) y `v` para los valores (*value*).