List Comprehension

January 31, 2018

```
lambda argument_list: expression
```

```
In [3]: sum = lambda x, y : x + y
        sum(3,4)
```

```
Out[3]: 7
```

```
In [5]: # lambda function
        foo_lam = lambda a: 2


        # regular function
        def foo_def(a):
           return 2
        foo.__qualname__ = '<lambda>'
```

```
In [6]: foo_lam(2)
```

```
Out[6]: 2
```

```
In [7]: foo_def(2)
```

```
Out[7]: 2
```

```
In [8]: print(type(foo_def))
        print(type(foo_lam))
```

```
<class 'function'>
<class 'function'>
```

```
In [9]: # lambda function
        sum_lam = lambda x, y : x + y


        # regular function
        def sum_def(x, y):
           return x + y
        foo.__qualname__ = '<lambda>'
```

```
In [10]: sum_def(32,28)

Out[10]: 60

In [11]: sum_lam(28,32)

Out[11]: 60
```

0.1 Sobre el futuro de map, filter, reduce y lambda

0.1.1 All Things Pythonic: The fate of reduce() in Python 3000

by Guido van van Rossum, March 10, 2005 http://www.artima.com/weblogs/viewpost.jsp?thread=98196

Adiós funciones lambda, hola comprensión de listas https://docs.python.org/3/tutorial/datastructures.html

La comprensión de listas es una forma elegante de crear nuevas listas al basarse en listas existentes. Al usar estos patrones, se pueden construir listas mediante cualquier método `iterable` de listas, strings, tuplas y sets.

Su sintáxis: `nueva_lista = [x for x in iterable]`

Donde una lista, u otro iterable, es asignado a una variable. Pudiendo ser seguido de otra sentencia `for` o `if`. La palabra reservada `in` es usada de manera similar en loops `for` para iterar sobre el objeto.

Observemos el siguiente ejemplo, `letras_ml`:

```
In [14]: letras_ml = [symbol for symbol in 'Aprendizaje Automático']
```

En este caso, la nueva lista asignada a la variable `lista_ml`, y `symbol` son usados para iterceptar los elementos contenidos en el objeto iterable `Aprendizaje Automático` <<-- Que es de tipo string.

Para confirmar de manera visual que contiene la nueva lista `lista_ml`, podemos usar `print()` o una celda de Jupyter

```
In [16]: letras_ml

Out[16]: ['A',
          'p',
          'r',
          'e',
          'n',
          'd',
          'i',
          'z',
          'a',
          'j',
          'e',
          ' ',
          'A',
          'u',
          't',
          'o',
```

```
 'm',
 'á',
 't',
 'i',
 'c',
 'o']
```

Para tener un mejor entendimiento del concepto, podemos re-escribir la comprensión de listas como un loop `for`:

```
In [17]: letras_ml = []

         for symbol in 'Aprendizaje Automático':
             letras_ml.append(symbol)

         print(letras_ml)
```

```
['A', 'p', 'r', 'e', 'n', 'd', 'i', 'z', 'a', 'j', 'e', ' ', 'A', 'u', 't', 'o', 'm', 'á', 't'
```

Al crear una lista usando un loop `for`, la variable asignada a la lista debe ser inicializada como una lista vacia, como en la primer línea del ejemplo.

Después, el loop `for` itera sobre el elemento de tipo string usando la variable `symbol`. Por cada paso/iteración del ciclo, un componente del objeto string se añade a la lista usando el método `list.append(x)`.

Obtenemos el mismo resultado en ambas expresiones.

Las comprensiones de listas pueden ser re-escritas como loops `for`, y algunos loops `for` pueden ser re-escritos en la sintáxis de comprensión de lista para obtener una expresión concisa o de mayor claridad en el código.

Uso de condicionales en la comprensión de listas Como se mencionaba con anterioridad, podemos aplicar elementos condicionales para modificar listas existentes u otros objetos secuencuales al crear nuevas listas.

```
In [18]: tupla_razas = ('akita', 'beagle', 'bulldog', 'samoyedo')

         lista_razas = [item for item in tupla_razas if item != 'bulldog']

In [21]: lista_razas

Out[21]: ['akita', 'beagle', 'samoyedo']
```

Operadores matemáticos

```
In [51]: lista_numeros = [ x ** 2 for x in range(10) if x % 2 == 0]

In [49]: lista_numeros = [ x + 2 for x in range(10) if x % 2 == 0]

In [52]: lista_numeros
```

```
Out[52]: [0, 4, 16, 36, 64]

In [24]: lista_numeros = [ x for x in range(10)]

In [25]: lista_numeros

Out[25]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Agregando una condición:

```
In [26]: lista_numeros = [x for x in range(10) if x % 2 == 0]

In [27]: lista_numeros

Out[27]: [0, 2, 4, 6, 8]
```

Generando el cuadrado de cada número:

```
In [58]: lista_numeros = [x ** 2 for x in range(10) if x % 2 == 0]

In [59]: lista_numeros

Out[59]: [0, 4, 16, 36, 64]
```

Incluso podemos anidar condiciones `if` en una comprensión de lista:

```
In [32]: lista_numeros = [x for x in range(100) if x % 3 == 0 if x % 5 == 0]

In [33]: lista_numeros

Out[33]: [0, 15, 30, 45, 60, 75, 90]
```

Nested Loops Podemos anidar loops para generar multiples iteraciones en nuestros programas.

Sin comprensión de listas:

```
In [35]: mi_lista = []

         for x in [20, 40, 60]:
             for y in [2, 4, 6]:
                 mi_lista.append(x * y)

In [36]: mi_lista

Out[36]: [40, 80, 120, 80, 160, 240, 120, 240, 360]

In [37]: mi_lista = [x * y for x in [20, 40, 60] for y in [2, 4, 6]]

In [38]: mi_lista

Out[38]: [40, 80, 120, 80, 160, 240, 120, 240, 360]

In [39]: # http://www.mathsisfun.com/sets/set-builder-notation.html
```

0.2 Ejercicios - Backup

Para la lista a, de contenido [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]. Escribe una lína de código en Python que tome esta lista y genere una nueva basada en los elementos pares de la lista original:

`a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`

Solución: `[element for element in a if element % 2 == 0]`

let's say we need to create a list of integers which specify the length of each word in a certain sentence, but only if the word is not the word "the".

Crear una lista de enteros que especifique la longitud de cada palabra en una oración, pero solo si la palabra no es 'xxxx'

Migrar el siguiente codigo a comprension de listas

```
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = []
for word in words:
      if word != "the":
          word_lengths.append(len(word))
print(words)
```

```
In [48]: sentence = "the quick brown fox jumps over the lazy dog"
         words = sentence.split()
         word_lengths = []
         for word in words:
               if word != "the":
                   word_lengths.append(len(word))
         print(word_lengths)
```

```
[5, 5, 3, 5, 4, 4, 3]
```

```
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = [len(word) for word in words if word != "the"]
print(words)
```

```
In [45]: sentence = "the quick brown fox jumps over the lazy dog"
         words = sentence.split()
         word_lengths = [len(word) for word in words if word != "the"]
         print(word_lengths)
```

```
[5, 5, 3, 5, 4, 4, 3]
```