David Gopar Morales
Programación avanzada
CINVESTAV

Tarea 5. Explicar con detalle y ejemplos los métodos map, filter y reduce.

Comenzaremos con el método map().

Esta es una función que nos ayuda a transformar elementos de un objeto iterables sin tener que hacer uso explícito de un ciclo for, esta técnica se conoce como mapeo.

La función map() aplica una función a cada elemento de un iterable (lista, túpla, etc) y nos regresa un iterador, como se nos muestra en el ejemplo a continuación:

```
>>> def cuadrado(n):
...     return n*n
...
>>> aux=[1,2,3,4,5]
>>> ite=map(cuadrado, aux)
>>> cuadrado=list(ite)
>>> cuadrado
[1, 4, 9, 16, 25]
```

Como podemos ver primero definimos la función, creamos después el objeto iterable aux, creamos el elemento iterable ite que es el resultado de a aux aplicarle la función cuadrado, luego convertimos este a lista y obtenemos el resultado deseado, aplicarle a aux la función de elevar cada elemento al cuadrado.

La sintaxis de la función map() es:

map(función, iterable[iterable1, iterable2, …])

Ya vimos un ejemplo utilizando una función que nosotros creamos, veamos ahora otro ejemplo del posible uso:

```
>>> cad=["1","2","3"]
>>> r=map(int, cad)
>>> r
<map object at 0x7f5cef9bd370>
>>> list(r)
[1, 2, 3]
>>> cad
['1', '2', '3']
```

En el cual podemos ver que se transforma una cadena de ‘números’, a formato entero, lo cual es muy útil cuando se lee texto y precisamente se quiere transformar a int en python.

Otro ejemplo podría ser con la función para obtener valores absolutos

```
>>> aux=[1,2,-3,-1]
>>>
>>> list(map(abs, aux))
[1, 2, 3, 1]
```

Así como para saber la longitud de cadenas.

```
>>> aux=["cadena","a","aa"]
>>> list(map(len, aux))
[6, 1, 2]
```

Como vimos en la clase, también es común usar la función lambda como argumento, para entenderlo mostramos el siguiente ejemplo:

```
[21, 22, 23, 24, 25, 26]
>>> aux=[1,2,3,4,5,6]
>>> list(map(lambda x: x+20, aux))
[21, 22, 23, 24, 25, 26]
```

Más aún, se pueden utilizar dos elementos iterables y hacerlos interactuar entre de ellos como:

```
>>> aux
[1, 2, 3, 4, 5, 6]
>>> aux1=[2,3,4,5,6,7]
>>> list(map(pow, aux, aux1))
[1, 8, 81, 1024, 15625, 279936]
```

```
>>> aux
[1, 2, 3, 4, 5, 6]
>>> aux1
[2, 3, 4, 5, 6, 7]
>>> list(map(lambda n1, n2 : n1+n2, aux, aux1))
[3, 5, 7, 9, 11, 13]
```

Los ejemplos anteriores nos ilustran a muchas posibilidades que existen a la hora de usar la función map(). A continuación anexamos algunos otros ejemplos, donde más que ilustrar la función map(), nos muestran otras funciones que existen en python y para ver su posible utilidad basta pensar un poco.

```
>>> string_it = ["processing", "strings", "with", "map"]
>>> list(map(str.capitalize, string_it))
['Processing', 'Strings', 'With', 'Map']

>>> list(map(str.upper, string_it))
['PROCESSING', 'STRINGS', 'WITH', 'MAP']

>>> list(map(str.lower, string_it))
['processing', 'strings', 'with', 'map']
```

```
>>> with_spaces = ["processing ", "  strings", "with   ", " map   "]

>>> list(map(str.strip, with_spaces))
['processing', 'strings', 'with', 'map']
```

```
>>> import re

>>> def remove_punctuation(word):
...     return re.sub(r'[!?.:;,"()-]', "", word)

>>> remove_punctuation("...Python!")
'Python'
```

```
>>> text = """Some people, when confronted with a problem, think
... "I know, I'll use regular expressions."
... Now they have two problems. Jamie Zawinski"""

>>> words = text.split()
>>> words
['Some', 'people,', 'when', 'confronted', 'with', 'a', 'problem,', 'think'
, '"I', 'know,', "I'll", 'use', 'regular', 'expressions."', 'Now', 'they',
 'have', 'two', 'problems.', 'Jamie', 'Zawinski']

>>> list(map(remove_punctuation, words))
['Some', 'people', 'when', 'confronted', 'with', 'a', 'problem', 'think',
'I', 'know', "I'll", 'use', 'regular', 'expressions', 'Now', 'they', 'have
', 'two', 'problems', 'Jamie', 'Zawinski']
```

```
>>> def powers(x):
...     return x ** 2, x ** 3
...

>>> numbers = [1, 2, 3, 4]

>>> list(map(powers, numbers))
[(1, 1), (4, 8), (9, 27), (16, 64)]
```

Y así como estos ejemplos podemos pensar en muchos mas.
Por lo cual es muy clara la utilidad de la función map().

Método filter()

Como su nombre lo índica, este método nos ayudará a filtrar elementos de un objeto iterable haciendo uso de una función, la sintaxis es similar al método map() y veremos como se pueden ocupara ambas funciones.

La sintaxis:

filter(función, iterable[…])

La función actúa sobre cada elemento del iterable y revisa si esta función nos regresa verdadero o falso, al final filter nos regresa solo los elementos que satisficieron obtener el valor verdadero bajo la función en un objeto iterable.

En el ejemplo a continuación definimos una función que nos dirá cuando un número se estrictamente mayor que cero, y de una lista obtendremos sólo los que satisfacen ser mayores que 0 utilizando el método filter():

```
>>> def pos(num):
...     if num>0:
...             return True
...     return False
...
>>> aux=[1,2,0,-3,-4,-5,90]
>>> list(filter(pos,aux))
[1, 2, 90]
```

Al igual que map(), también se puede hacer uso de la función lambda en filter como tenemos en el siguiente ejemplo:

```
>>> aux=[2,3,4,5,6,7,8]
>>> list(filter(lambda n: n%2!=0, aux))
[3, 5, 7]
```

En el cual vemos como nos devuelve solo los número impares del objeto iterable.

A continuación vemos un ejemplo de como utilizar el método filter() con diccionarios:

```
>>> def may(l):
...     if l["precio"]>20:
...             return True
...     return False
...
>>> aux=[{1:"Peli1","precio":20},{1:"Peli2","precio":30}]
>>> list(filter(may,aux))
[{1: 'Peli2', 'precio': 30}]
```

También se puede usar None como función y sólo nos regresará los valores que al convertirlos a booleano sean verdaderos.

```
>>> aux = [1, 'a', 0, False, True, '0']
>>> list(filter(None, aux))
[1, 'a', True, '0']
```

Y terminamos con un ejemplo de como utilizar el filter() dentro de la función map() para poder hacer operaciones sobre los conjuntos que sean válidos.

```
>>> import math

>>> def is_positive(num):
...     return num >= 0
...

>>> def sanitized_sqrt(numbers):
...     cleaned_iter = map(math.sqrt, filter(is_positive, numbers))
...     return list(cleaned_iter)
...

>>> sanitized_sqrt([25, 9, 81, -16, 0])
[5.0, 3.0, 9.0, 0.0]
```

Método reduce()

reduce() es una función que vive en el módulo llamado *functools* en la librería estándar de python. Esta función reduce() es útil cuando se quiere aplicar una función a un objeto iterable y se quiere reducir a un solo objeto iterable, esta operación es comúnmente conocida como reducción o folding.

La sintaix de reduce() es:

reduce(función, iterable[…])

La cual se puede importar poniendo

```
>>> from functools import reduce
```

La documentación de python nos indica que reduce() es aproximadamente equivalente a la siguiente función:

Python

```
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value
```

Como la función mostrada, reduce() funciona aplicando una función de dos argumentos a los elementos de un objeto iterable en un ciclo de izquierda a derecha para finalizar reduciendo el iterable a un solo valor acumulativo. Esta función reduce acepta un tercer argumento llamado inicializador que proporciona un valor semilla al computo que se va a realizar.

Mostremos el siguiente ejemplo para comenzar a entender como funciona reduce().

```
>>> def suma(a, b):
...     r = a + b
...     print(f"{a} + {b} = {r}")
...     return r
...
>>> from functools import reduce
>>> aux = [0, 1, 2, 3, 4, 5]
>>> reduce(suma, aux)
0 + 1 = 1
1 + 2 = 3
3 + 3 = 6
6 + 4 = 10
10 + 5 = 15
15
```

Ubuntu Software

Dónde vemos todas las operaciones que efectúa la función suma y como nos regresa como resultado el valor acumulado que es 15.

Y ahora vemos el ejemplo poniendo como tercer parámetro el incializador.

```
>>> reduce(suma, aux,100)
100 + 0 = 100
100 + 1 = 101
101 + 2 = 103
103 + 3 = 106
106 + 4 = 110
110 + 5 = 115
115
```

En este caso se realiza una operación más de la que se realizaría en caso de no poner el inicializador, el cual es importante agregar en caso de que se trabaje con iterables que tenga la posibilidad de estar vacíos para así evitar posibles errores.

También podemos usar funciones lambda con reduce() como mostramos a continuación.

```
>>> aux
[0, 1, 2, 3, 4, 5]
>>> reduce(lambda a,b: a+b, aux)
15
>>> reduce(lambda a,b: a*b, aux)
0
>>> aux=[1,2,3,4,5]
>>> reduce(lambda a,b: a*b, aux)
120
```

Ahora exponemos un ejemplo de como obtener el valor mínimo de cierto conjunto de datos utilizando reduce().

```
>>> def min(a,b):
...     if a<b:
...             return a
...     return b
...
>>> l=[20,2,5,3,1,4]
>>> reduce(min, l)
1
```

A partir de este es fácil ver como obtener el máximo de manera análoga.
Además hay otras maneras de obtener mínimo.

```
>>> l
[20, 2, 5, 3, 1, 4]
>>> reduce(lambda a,b: a if a<b else b, l)
1
```

Incluso al ser un problema tan común se tiene como función reservada mi(), con parámetro un iterable, y nos da el mínimo y max() para el máximo.

Como nota igual se puede usar reduce para ver si todos los valores en una lista son verdaderos o si alguno lo es u algunas otras variaciones de esto.

Si entramos a help(functools) nos dice lo siguiente sobre la función reduce:

```
    reduce(...)
        reduce(function, sequence[, initial]) -> value

        Apply a function of two arguments cumulatively to the items of a sequenc
e,
        from left to right, so as to reduce the sequence to a single value.
        For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates
        ((((1+2)+3)+4)+5).  If initial is present, it is placed before the items
        of the sequence in the calculation, and serves as a default when the
        sequence is empty.
```

Y nos dice lo siguiente de functools:

```
Help on module functools:

NAME
    functools - functools.py - Tools for working with functions and callable obj
ects

MODULE REFERENCE
    https://docs.python.org/3.8/library/functools

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

CLASSES
    builtins.object
        cached_property
        partial
        partialmethod
        singledispatchmethod
```

Con lo cual concluimos la tarea.

Fuente:
https://realpython.com/python-map-function/
https://www.programiz.com/python-programming/methods/built-in/filter
https://www.simplilearn.com/tutorials/python-tutorial/filter-in-python
https://www.programiz.com/python-programming/methods/built-in/map
https://www.pythontutorial.net/python-basics/python-reduce-list/
https://realpython.com/python-reduce-function/