T09_LambdaExpressions_Reduce

January 29, 2021

1 Python de cero a experto

Autor: Luis Miguel de la Cruz Salas

1.1 Pythonico es más bonito: Pensando como pythonista (intermedio)

1.1.1 Lambda expressions

Programación funcional

- Paradigma de programación basado en el uso de funciones, entendiendo el concepto de función según su definición matemática, y no como los subprogramas de los lenguajes imperativos.
- Tiene sus raíces en el cálculo lambda (un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión).
- Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.
- Las funciones que se usan en este paradigma son *funciones puras*, es decir, que no tienen efectos secundarios, que no manejan datos mutables o de estado.
- Lo anterior está en contraposición con la programación imperativa.
- Uno de sus principales representantes es el lenguaje Haskell, que compite en belleza, elegancia y expresividad con Python.
- Los programas escritos en un estilo funcional son más fáciles de probar y depurar.
- Por su característica modular facilita el cómputo concurrente y paralelo.
- El estilo funcional se lleva muy bien con los datos, permitiendo crear algoritmos y programas más expresivos para trabajar en *Big Data*.

Descripción.

- Una expresión Lambda (*Lambda expressions*) nos permite crear una función "anónima", es decir podemos crear funciones *ad-hoc*, **sin** la necesidad de definir una función propiamente con el comando **def**.
- Una expresión Lambda o función anónima, es una expresión simple, no un bloque de declaraciones.

- Solo hay que escribir el resultado de una expresión en vez de regresar un valor explícitamente.
- Dado que se limita a una expresión, una función anónima es menos general que una función normal **def**.

Por ejemplo:

[1]:
```
# Una función normal que calcula el cuadrado
def square(n):
    result = n**2
    return result
```

[2]:
```
square(5)
```

[2]: 25

Se puede reducir el código anterior como sigue:

[3]:
```
def square(n):
    return n**2
```

[4]:
```
square(5)
```

[4]: 25

Se puede reducir aún más, pero puede llevarnos a un mal estilo de programación:

[5]:
```
def square(n): return n**2
```

[6]:
```
square(5)
```

[6]: 25

Definición. La sintáxis de una expresión lambda (función lambda o función anónima) es muy simple:

```
lambda argument_list: expression
```

1. La lista de argumentos consiste de objetos separados por coma.
2. La expresión es una declaración válida de Python.

Se puede asignar la función a una etiqueta para darle un nombre.

Ejemplo 1. Función anónima para el cálculo del cuadrado de un número:

[7]:
```
lambda n: n**2
```

[7]: <function __main__.<lambda>(n)>

[8]:
```
cuadrado = lambda num: num**2
```

[9]:
```
x = cuadrado(7)
print(x)
```

```
49
```

Ejercicio 1. Escribir una función lambda para calcular el cubo de un número usando la función lambda que calcula el cuadrado.

[10]:
```
cubo = lambda n: cuadrado(n) * n
```

[11]:
```
cubo(5)
```

[11]: 125

[12]:
```
cubo = lambda n: (lambda n: n**2 * n)
```

[13]:
```
cubo(5)(5)
```

[13]: 125

Ejercicio 2. Escribir una función lambda para multiplicar dos números.

Ejercicio 3. Checar si un número es par:

Ejercicio 4. Obtener el primer elemento de una lista o una cadena:

Ejercicio 5. Escribir en reversa una cadena y/o una lista:

[14]:
```
#rev = lambda l:l[::-1]
#c = 'reconocer'
#print(c)
#print(rev(c))
#print(rev('Anita lava la tina'))
```

Ejercicio 6. Convertir grados Fahrenheit a Celsius y viceversa combinando **map( )** con **lambda**.

[15]:
```
c = [0, 22.5, 40,100]
f = list(map(lambda T: (9/5) * T + 32, c))
print(f)
print(list(map(lambda T: (5/9)*(T - 32), f)))
```

```
[32.0, 72.5, 104.0, 212.0]
[0.0, 22.5, 40.0, 100.0]
```

Ejercicio 7. Sumar tres arreglos combinando **map( )** con **lambda**.

[16]:
```
a = [1,2,3,4]
b = [5,6,7,8]
c = [9,10,11,12]
list(map(lambda x,y,z : x+y+z, a,b,c))
```

[16]:
```
[15, 18, 21, 24]
```

Ejercicio 8. Encontrar todos los números pares de una lista combinando **filter( )** con **lambda**.

[17]:
```
nums = [0, 2, 5, 8, 10, 23, 31, 35, 36, 47, 50, 77, 93]
result = filter(lambda x : x % 2 == 0, nums)
print(list(result))
```

```
[0, 2, 8, 10, 36, 50]
```

Ejercicio 9. Encontrar todos los números primos en el conjunto $\{2, \ldots, 50\}$ combinando **filter( )** con **lambda**.

[18]:
```
nums = list(range(2, 50))
for i in range(2, 8):
    nums = list(filter(lambda x: x == i or x % i, nums))

print(nums)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

1.1.2 Funciones puras e impuras

- La programación funcional busca usar funciones *puras*, es decir, que no tienen efectos secundarios, no manejan datos mutables o de estado.
- Estas funciones puras devuelven un valor que depende solo de sus argumentos.

Por ejemplo:

[19]:
```
def pura(x, y):
    return (x + 2 * y) / (2 * x + y)
```

[20]:
```
pura(1,2)
```

[20]:
```
1.25
```

[21]:
```
lambda_pura = lambda x,y: (x + 2 * y) / (2 * x + y)
```

[22]:
```
lambda_pura(1,2)
```

[22]:
```
1.25
```

[23]:
```
# Esta es una función impura
lista = []
def impura(arg):
    potencia = 2
    lista.append(arg)
    return arg ** potencia
```

[24]:
```
impura(5)
```

[24]: 25

[25]:
```
lista
```

[25]: [5]

[26]:
```
# podemos crear funciones lambda impuras :o
lambda_impura = lambda l, arg : (l.append(arg),arg**2)
```

[27]:
```
print(lambda_impura(lista,5))
lista
```

```
(None, 25)
```

[27]: [5, 5]

Una buena práctica del estilo funcional es evitar los efectos secundarios, es decir, que nuestras funciones NO modifiquen los valores de sus argumentos.

Ejemplo 2.

[28]:
```
# Esta función calcula el cuadrado de una lista de números.
def cuadradosImpuros(lista):
    for i, v in enumerate(lista):
        lista[i] = v ** 2
    return lista
```

[29]:
```
numeros = list(range(11))
print(numeros)
cuadradosImpuros(numeros)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[29]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

[30]:
```
print(numeros)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Una manera alternativa es la siguiente:

[31]:
```
numeros = list(range(11))
list(map(lambda x: x ** 2, numeros))
```

[31]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[32]:
```
print(numeros)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[33]:
```
def cuadradosPuros(lista):
    return list(map(lambda x: x ** 2, numeros))

numeros = list(range(11))
print(numeros)

cuadradosPuros(numeros)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[33]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[34]:
```
print(numeros)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Reduce ()

- **Reducción** : Disminuir *algo* en tamaño, cantidad, grado, importancia, ..
- La operación de reducción es útil en muchos ámbitos, el objetivo es reducir un conjunto de objetos en un objeto más simple.

Definición.

`reduce(function, sequence)`

reduce( ) es una función que toma dos argumentos:

1. Una función de reducción $f()$.
2. Una secuencia iterable s.

Trabaja como sigue:

$$\underbrace{\underbrace{\underbrace{[\underbrace{s_1, s_2}_{a=f(s_1,s_2)}, s_3, s_4]}_{b=f(a,s_3)}}_{c=f(b,s_4)}}_{} \quad \Longrightarrow \quad \underbrace{\underbrace{\underbrace{f(f(\underbrace{f(s_1, s_2)}_{a}, s_3), s_4)}_{b}}_{c}}_{}$$

Por ejemplo:

$$1 + 2 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Si $n = 4$ entonces 1+2+3+4 = 10

[35]:
```
from functools import reduce
nums = [1,2,3,4]
print(nums)
result = reduce(lambda x, y: x + y, nums)
print(result)
```

```
[1, 2, 3, 4]
10
```

[36]:
```
import numpy as np
a = np.ones(10)
print(a)
result = reduce(lambda x, y: x + y, a)
print(result)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
10.0
```

[37]:
```
otra_lista = [3,4,5]
result = reduce(lambda x, y: 1/x + 1/y, otra_lista)
print(result)
```

```
1.9142857142857144
```

Ejercicio 10. Calcular el máximo de una lista de números.

[38]:
```
max_find = lambda a,b: a if (a > b) else b
```

[39]:
```
lst = [23,5,23,56,87,98,23]
```

Ejercicio 11. Calcular el factorial de un número.

Ejercicio 12. Contar el número de caractéres de un texto, combinando **reduce( )**, **map( )** y **lambda**.

[40]:
```
texto = 'Hola Mundo'
palabras = texto.split()
print(palabras)

print(reduce(lambda x,y: x+y, list(map(lambda palabras: len(palabras),␣
 ↪palabras))))

archivo = open('QueLesQuedaALosJovenes.txt','r')
suma = 0
```

```
for linea in archivo:
    palabras = linea.split()
    suma += reduce(lambda x,y: x+y, list(map(lambda palabras: len(palabras),␣
 ↪palabras)))
print(suma)
archivo.close()
```

```
['Hola', 'Mundo']
9
824
```

[41]:
```
def cuentaCaracteres(palabras):
    return reduce(lambda x,y: x+y, list(map(lambda palabras: len(palabras),␣
 ↪palabras)))

texto = 'Hello Motto'.split()
cuentaCaracteres(texto)
```

[41]:
```
10
```

[ ]:
```

```