Python para Análisis de Datos

Módulo 02

Funciones anónimas

Funciones anónimas

Python permite definir pequeñas funciones sin nombre: las **funciones anónimas** o ***lambda*.** Estas funciones no tienen nombre y su cuerpo está limitado a una única expresión. Para definirlas se utiliza la palabra reservada `lambda` seguida de los parámetros que tome la función y luego de los dos puntos sigue la expresión a evaluar y devolver. A continuación, algunos ejemplos de funciones ordinarias y las lambda equivalentes.

```
def sumar(x,y):
    return x + y

lambda x,y: x+y
```

```
def par(x):
    return x%2==0

lambda x: x%2==0
```

```
def cuenta(x):
    return x**2 + 25

lambda x: x**2 + 25
```

Si bien es posible asignarles un nombre a través de una variable, esto está desaconsejado (PEP 8) y no fue para eso que se diseñaron. Se usan para definir funciones de una forma rápida y compacta en el mismo momento en que deben ser pasadas. Típicamente se utilizan con funciones de orden superior como las vistas anteriormente.

Podemos pensar en las funciones anónimas como funciones “descartables” que se definen “al paso”. Generalmente, sirven cuando hay que pasar una función que realice una tarea sencilla que se pueda escribir en una única expresión. Si la tarea es más compleja es recomendable usar una función ordinaria.

Siguiendo los ejemplos anteriores, como la función `cuenta` se reduce a una expresión se puede usar una función anónima en su lugar. El código resultante es más compacto.

```
def cuenta(x):
    return x**2 + 25

list(map(cuenta, numeros))
```

```
[29, 41, 50, 89, 74, 89, 50, 34, 61, 50, 26]
```

```
list(map(lambda x: x**2+25, numeros))
```

```
[29, 41, 50, 89, 74, 89, 50, 34, 61, 50, 26]
```

Ejemplo

Supongamos que tenemos una lista de strings que contienen nombres pero están todos en formato distinto e incluso algunos tienen espacios en blanco que sobran. Queremos que queden todos igual y para eso tenemos que aplicar primero el método `strip` y luego `capitalize` a cada elemento. Podemos hacer uso de *map* y *lambda* para realizar esto de forma sencilla.

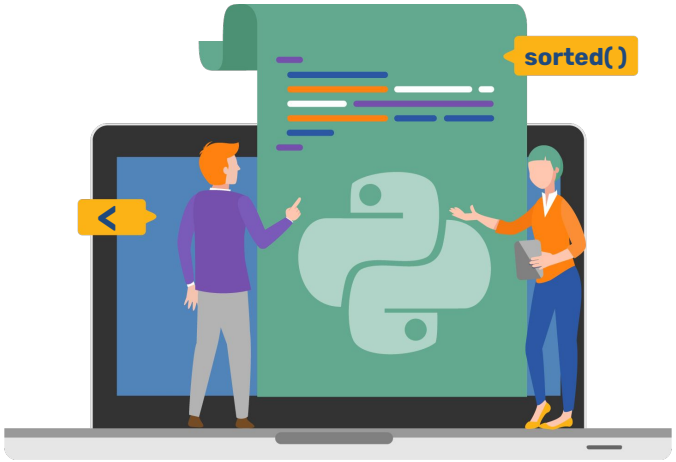
```
nombres = ["jUaN", "  carolina  ", "RAÚL", "  FlaVIA"]

list(map(lambda x: x.strip().capitalize(), nombres))
```

```
['Juan', 'Carolina', 'Raúl', 'Flavia']
```

Sorted

La función ***sorted*** permite ordenar elementos de un iterable siempre que esté implementado el operador **<** entre sus elementos. En principio, los ordena de menor a mayor, pero se puede controlar la forma en que se ordenan los elementos. La función tiene dos parámetros opcionales: `key` y `reverse`. Si `reverse` se define como `True` los elementos son ordenados de mayor a menor.

Sin embargo, el más interesante es `key`. Este parámetro permite definir un criterio para ordenar los elementos. Tiene que ser una función y los elementos se van a ordenar según el resultado de esa función.



```python
numeros = [1,2,5,6,3,2,5,7,9,6,8,0,5,4]
# numeros ordenados de menor a mayor
sorted(numeros)
```

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

```python
listas = [[55,70,31], [15,65,14,23], [8,15,23,5,20,16]]
# listas ordenadas según la suma de sus elementos
sorted(listas, key=sum)
```

```
[[8, 15, 23, 5, 20, 16], [15, 65, 14, 23], [55, 70, 31]]
```

```python
nombres = ["Juan", "Catalina", "Esteban", "Carlos"]
# nombres ordenados según la cantidad de caracteres
sorted(nombres, key=len)
```

```
['Juan', 'Carlos', 'Esteban', 'Catalina']
```

Ejemplo

Supongamos que tenemos una serie de nombres y apellidos y quisiéramos ordenar esos nombres según su apellido (la última palabra del string).

Tendríamos que poder seleccionar esa última palabra y pasarsela a la función **`sorted`** como criterio para ordenar los strings. Esto lo podemos hacer con una función `lambda`.

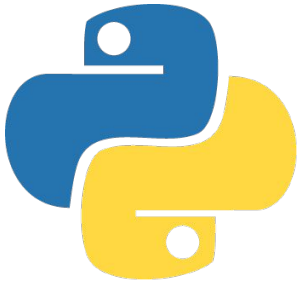
```
nombres = ["Juan Estevez", "María José Schwartz", "José Luis Castaño", "Victor Contreras"]
sorted(nombres, key=lambda x: x.split()[-1])
```

```
['José Luis Castaño',
 'Victor Contreras',
 'Juan Estevez',
 'María José Schwartz']
```

Método sort

Entre los métodos de listas encontramos también el método **`sort`.** Éste es muy similar a la función *sorted,* con la diferencia de que el método ordena la lista "inplace" mientras que la función crea otra lista dejando la lista original intacta. Además, el método **`sort`** sólo está implementado para listas mientras que *sorted* funciona con cualquier iterable (listas, tuplas, strings, diccionarios, etc.).

El método también tiene los parámetros `key` y `reverse` y funcionan igual que en la función.



¡Muchas gracias!

¡Sigamos trabajando!