PROGRAMACIÓN

PH.D. SANTIAGO ECHEVERRI ARTEAGA

INTRODUCCIÓN A LA PROGRAMACIÓN FUNCIONAL

ZIP / ENUMERATE / ITERTOOLS

CICLOS FOR Y WHILE

- Los ciclos en un lenguaje de programación hacen el código muy poco eficiente y no escalable. Para sobrellevar eso existe el paradigma de programación llamado programación funcional.
- Los lenguajes de programación puramente funcionales carecen de bucles for y while.
- Se dice que en la programación funcional, las funciones son **ciudadanas de primera clase**, lo que nos viene a decir que prácticamente todo se hace con funciones, y no con bucles.
- La programación funcional prefiere también las funciones puras, es decir, funciones sin efectos colaterales. Las funciones puras no dependen de variables externas o globales. Esto significa que para las mismas entradas, siempre se producen las mismas salidas.
- Por otro lado, **en la programación funcional se prefiere variables inmutables,** lo que significa que una vez creadas no pueden ser modificadas. Esto es algo que verdaderamente ahorra problemas, aunque la eficiencia puede ser discutible.
- En general, se considera que los lenguajes de programación funcionales son más seguros
- Aunque Python NO es un lenguaje puramente funcional, ofrece algunas funciones propias de lenguajes funcionales como map, filter y reduce.

MAP

- La función map toma dos entradas:
  - Una lista o iterable que será modificado en una nueva.
  - Una función, que será aplicada a cada uno de los elementos de la lista o iterable anterior.
- Nos devuelve una nueva lista donde todos y cada uno de los elementos de la lista original han sido pasados por la función.

```
lista = [1, 2, 3, 4, 5]
lista_pordos = map(lambda x: 2*x, lista)

print(list(lista_pordos))
# [2, 4, 6, 8, 10]
```

FILTER

- La función filter también recibe una función y una lista pero el resultado es la lista inicial filtrada. Es decir, se pasa cada elemento de la lista por la función, y sólo si su resultado es True, se incluye en la nueva lista.

```
lista = [7, 4, 16, 3, 8]
pares = filter(lambda x: x % 2 == 0, lista)

print(list(pares))
# [4, 16, 8]
```

REDUCE

- Se usa para reducir todos los elementos de la entrada a un único valor aplicando un determinado criterio.
- la función recibe dos argumentos: el acumulador y cada uno de los valores de la lista.
  - El acumulador es el valor devuelto en la iteración anterior, que va acumulando un resultado hasta que llegamos al final.
  - El valor es cada uno de los elementos de nuestra lista, que en nuestro caso vamos añadiendo al acumulador.
- Se puede pasar al final el valor inicial del acumulador

```python
from operator import add
from functools import reduce
lista = [1, 2, 3, 4, 5]
suma = reduce(add, lista)
print(suma) # 15
```

```python
from functools import reduce
lista = [1, 2, 3, 4, 5]
suma = reduce(lambda acc, val: acc + val, lista)
print(suma) # 15
```

```python
from functools import reduce
lista = [1, 2, 3, 4, 5]
multiplicacion = reduce(lambda acc, val: acc * val, lista)
print(multiplicacion) # 120
```

```python
from functools import reduce
personas = [
    {'Nombre': 'Alicia', 'Edad': 22},
    {'Nombre': 'Bob', 'Edad': 29},
    {'Nombre': 'Charlie', 'Edad': 33}
]
suma_edad = reduce(lambda total, p: total + p['Edad'], personas, 0)
print(suma_edad/len(personas)) # 28.0
```