Iterables

Repaso - Jueves 30 de abril 2020

Lo básico

Iteración

Tomar cada elemento de un algo, uno después del otro

Cuando usamos `for`, es una iteración

Cuando dentro de un `while` aumentamos un índice, es una iteración

---

Lo básico

Un **iterable** es cualquier objeto sobre el cual se puede **iterar**. Un **iterador** es quien itera sobre dicho iterable.

Lo básico

Metáfora para entender: Un **repartible** es cualquier objeto sobre el cual se puede **repartir**.

~~Iterable~~
Repartible

~~Iterador~~
Repartidor

El que algo sea “repartible” indica que puede ser “repartido”. Cuando en verdad queremos “repartir”, el “repartidor” lo hace.

Lo básico

```
class Repartible:
    def __init__(self, pedidos):
        self.pedidos = pedidos

    def __iter__(self):
        return RepartidorDePedidos(self)
```

1

Siguiendo con nuestra metáfora, un “repartible” se puede “repartir”, por lo que cada vez que queramos recorrer nuestros pedidos, lo hace un **Repartidor**.

Lo básico

```
class RepartidorDePedidos:
    def __init__(self, repartible):
        self.repartible = copy(repartible)  # Para no modificar original

    def __iter__(self):
        return self

    def __next__(self):
        if not self.repartible.pedidos:
            raise StopIteration("No quedan mas pedidos")
        proximo_pedido = self.repartible.pedidos.pop(0)
        return proximo_pedido
```

3

2

Cada vez que el **Repartidor** pasa al siguiente pedido (2) este se elimina de la lista, es consumido. En un iterable, solo está la información y no se modifica, mientras que un iterador va avanzando en el iterable y consumiendo cada elemento.

Lo básico

```
class RepartidorDePedidos:
    def __init__(self, repartible):
        self.repartible = copy(repartible)  # Para no modificar original

    def __iter__(self):
        return self

    def __next__(self):
        if not self.repartible.pedidos:
            raise StopIteration("No quedan mas pedidos")
        proximo_pedido = self.repartible.pedidos.pop(0)
        return proximo_pedido
```

3

2

En (3) vemos otra propiedad especial. Para que algo sea iterable, debe implementar el método __iter__ y retornar un iterador. En (3), **Repartidor** se retorna a si mismo, por lo que es tanto iterador como iterable.

Lo básico

```
iterable = Iterable()  # 🌽, 🥔, 🥚
iterador = iter(iterable)  # Iterable.__iter__

print(next(iterador))  # Iterador.__next__
>> 🌽
print(next(iterador))
>> 🥔
print(next(iterador))
>> 🥚
print(next(iterador)) # Si no quedan elementos...
>> StopIteration
```

Lo básico

Los **generadores** son un caso especial de los **iteradores**.

```
(i for i in range(10))
```

Generador

```
yield elemento
```

Función generadora

Lo básico

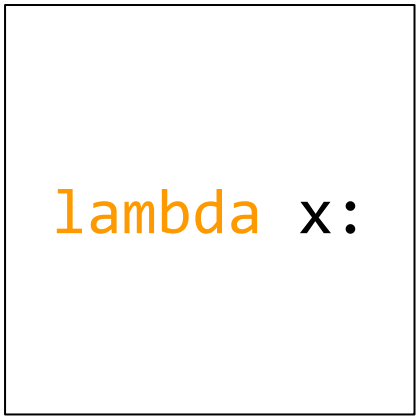
```
def ingredientes():
    yield 🌽
    yield 🥔
    yield 🥚

generador = ingredientes()

# El generador “recuerda”
# donde quedó la ejecución
# y continúa al hacer next
```

```
print(next(generador))
>> 🌽
print(next(generador))
>> 🥔
print(next(generador))
>> 🥚
print(next(generador))
>> StopIteration
```

Trabajando con iterables

Trabajando con iterables

Las **funciones *lambda*** son funciones anónimas y de uso fugaz.

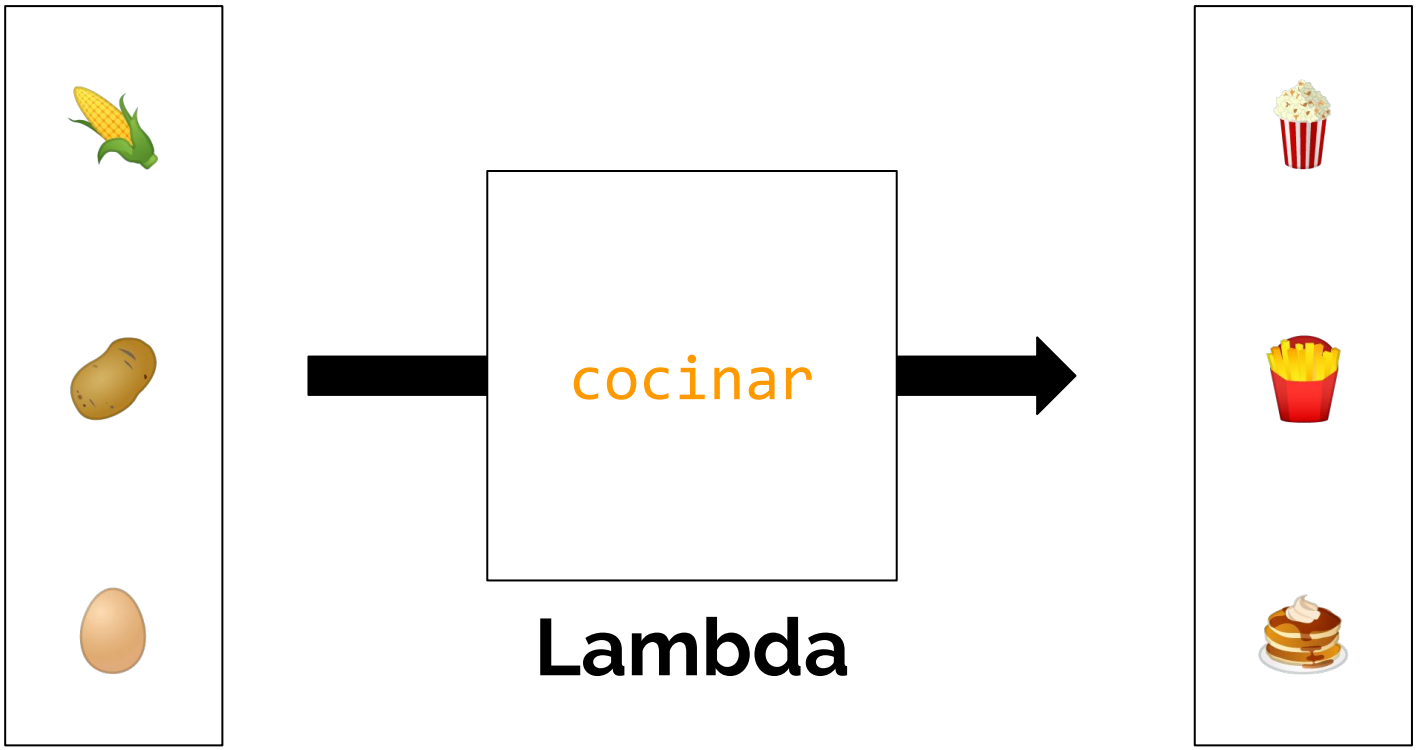


Lambda

```
lambda x: x * 2
lambda a, b: a + b
lambda p: p.procesar()
lambda a, p: a + p.precio
```

Trabajando con iterables

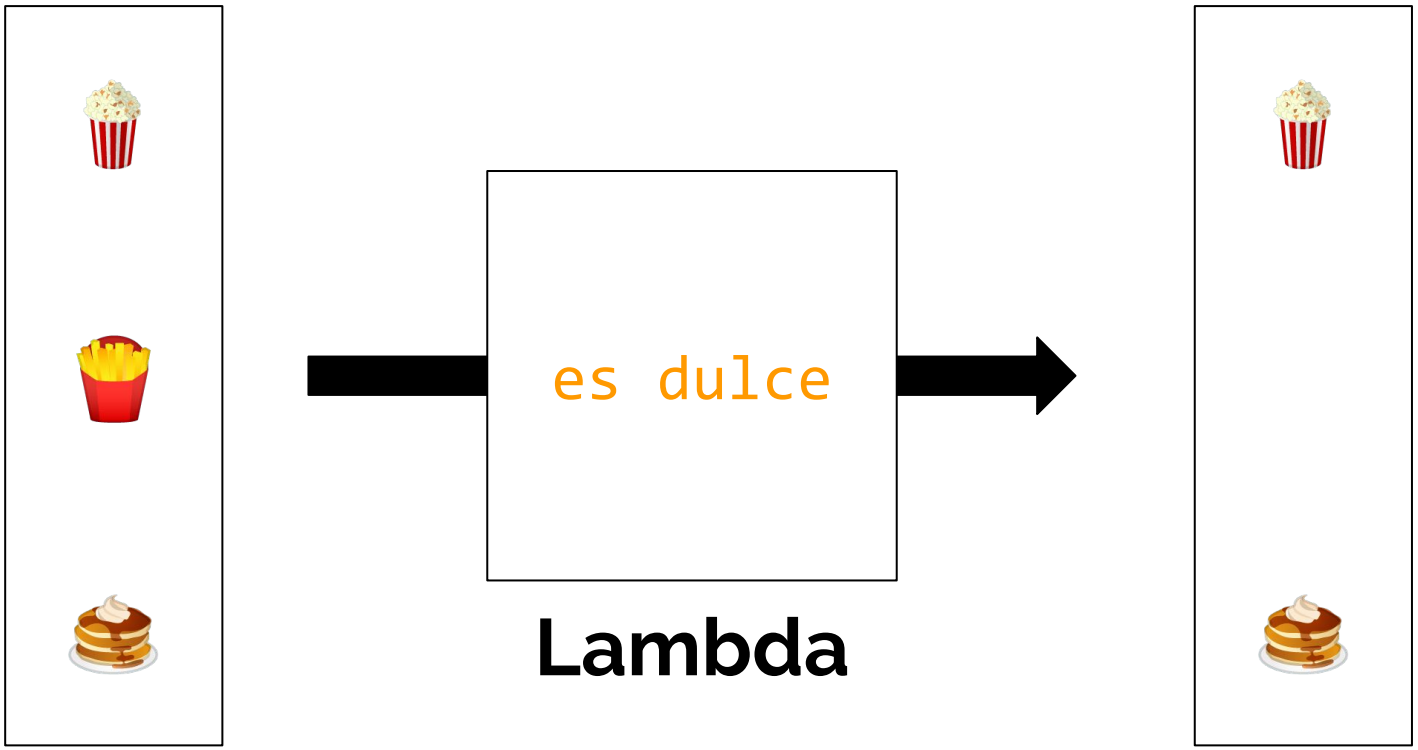
La función ***map*** aplica la **función** a cada elemento de un **iterable**.



`map(cocinar, [🌽, 🥔, 🥚]) → [🍿, 🍟, 🥞]`

Trabajando con iterables

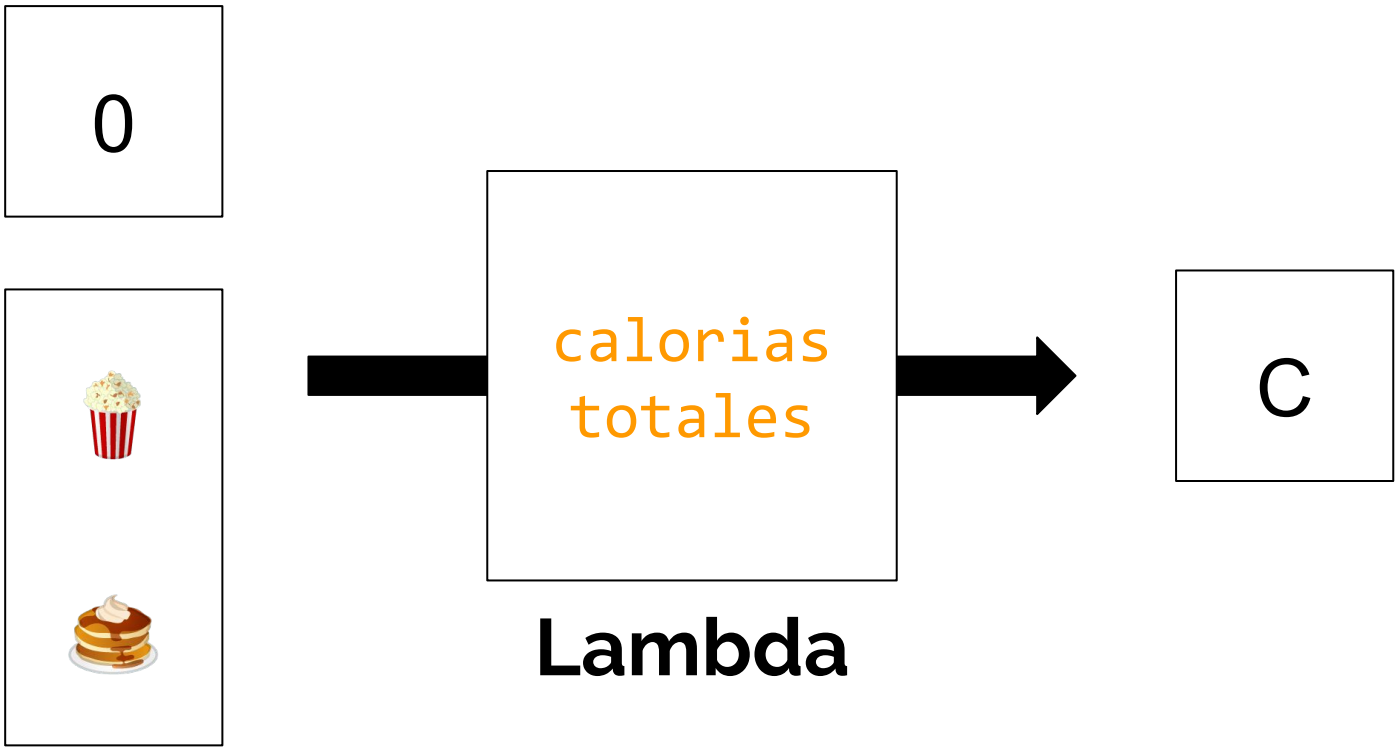
La función ***filter*** aplica la **función** para seleccionar elementos.



filter(es_dulce, [, ,]) → [,]

Trabajando con iterables

La función ***reduce*** aplica la **función** para componer el resultado hasta que quede solo un elemento.



`reduce(sumar_calorias, [` 🍿 , 🥞 `], 0)` → C (número)

Otros ejercicios

¡Si tienen dudas resolviendolos, no duden en hacer una *issue* con su duda!

- Ejercicios propuestos sobre *Iterables*. (Contenidos: semana 6)
- Actividad 2 en 2018-2 (Actividad sobre programación funcional, la cual incluye iterables)
- Actividad 3 en 2019-2 (Actividad sobre iterables)
- Actividad 11 en 2019-2 (Actividad sobre Recuperativa, donde la parte 1 es sobre iterables)

Iterables

Repaso - Jueves 30 de abril 2020