

# ***Programación Avanzada***

## **II C2233 2024-2**

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán



# Anuncios

1. Hoy tenemos la tercera actividad evaluada.
2. El próximo jueves tendremos el Midterm a las 17:30.
3. Dedicaremos un rato a responder la ETC.
4. Encuesta de Carga Académica.  
¡Respóndanla!

---



**AÚN NO ESTUDIO PARA EL  
MIDTERM DE LA OTRA SEMANA**

Tomémonos unos  
minutos para  
responder la ETC



# Repaso

# Programación Funcional

---

# Motivación

En el mundo de la programación existen distintos **paradigmas de las programación**:

## Procedimental

- Un programa lineal.
- **Lista de instrucciones** que indican al computador qué hacer en cada paso.

## Orientada a Objetos

- Modela funcionalidades a través **objetos** y la **interacción** de estos.
- Da sentido al programa, a través de los objetos.

## Funcional

- Se estructura la solución como un **conjunto de funciones**.
- Las **funciones no tienen estado**, es decir, el *output* depende exclusivamente del *input*.

# Programación Estrictamente Funcional

---

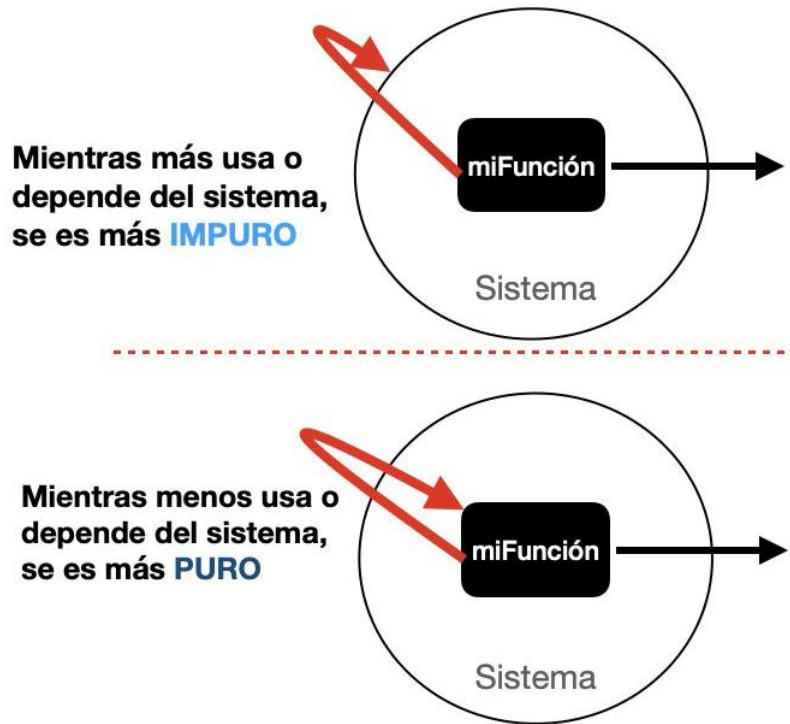


# Programación Estrictamente Funcional

Las funciones solo dependen de los argumentos (*input*) al momento de retornar un valor (*output*).

Cuando lo anterior se cumple, diremos que el **código es puro**.

La "pureza" mide el grado de **dependencia de la función** hacia dependencia de la función hacia el sistema.



# Programación Estrictamente Funcional

Se puede confirmar la pureza de una función, revisando si cumple con las siguientes preguntas:

1. ¿Usa solo argumentos de entrada?
2. ¿Muta valores existentes?
3. Si se mantiene el *input*, ¿siempre entrega el mismo *output*?

# Generadores y Funciones Generadoras

---

# Generadores







Los **generadores** son un caso especial de los **iteradores**. Los cuales son **muy eficientes en memoria**.

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

**Generador**

# Generadores

Cada generador "recuerda" dónde quedó la ejecución, es capaz de continuarla y acceder al siguiente elemento al hacer **next**.

```
ingredientes = [, , ]  
generador = (emojis[i] for i in range(len(ingredientes)))  
  
print(next(generator))  
>>   
print(next(generator))  
>>   
print(next(generator))  
>>   
print(next(generator))  
>> StopIteration
```

# Funciones Generadoras

En Python, podemos definir **funciones que creen generadores** por medio de la sentencia **yield**.

```
def cargar_ingredientes():  
    path = 'ingredientes.txt'  
    with open(path) as file:  
        for line in file:  
            yield line.strip()
```

```
# ingredientes.txt
```

```
# 🌽
```

```
# 🥔
```

```
# 🥚
```

```
generador = cargar_ingredientes()
```

```
print(next(generator))
```

```
>> 🌽
```

```
print(next(generator))
```

```
>> 🥔
```

```
print(next(generator))
```




```
>> 🥚
```





```
print(next(generator))
```

```
>> StopIteration
```

# Funciones Generadoras

En Python, podemos definir **funciones que creen generadores** por medio de la sentencia **yield**.

```
def ingredientes_inf():  
    ingredientes = [, ,     i = 0  
    while True:  
        yield ingredientes[i]  
        i = (i + 1) % 3
```

```
generador = ingredientes_inf()  
  
print(next(generator))  
>>   
print(next(generator))  
>>   
print(next(generator))  
>>   
print(next(generator))  
>> 
```

# Funciones Generadoras

En Python, podemos definir **funciones que creen generadores** por medio de la sentencia **yield**.

```
def contador_infinito():  
    número_actual = 0  
    while True:  
        yield número_actual  
        número_actual += 1
```

```
generador = contador_infinito()  
  
print(next(generator))  
>> 0  
print(next(generator))  
>> 1  
  
□  
print(next(generator))  
>> 26092024
```

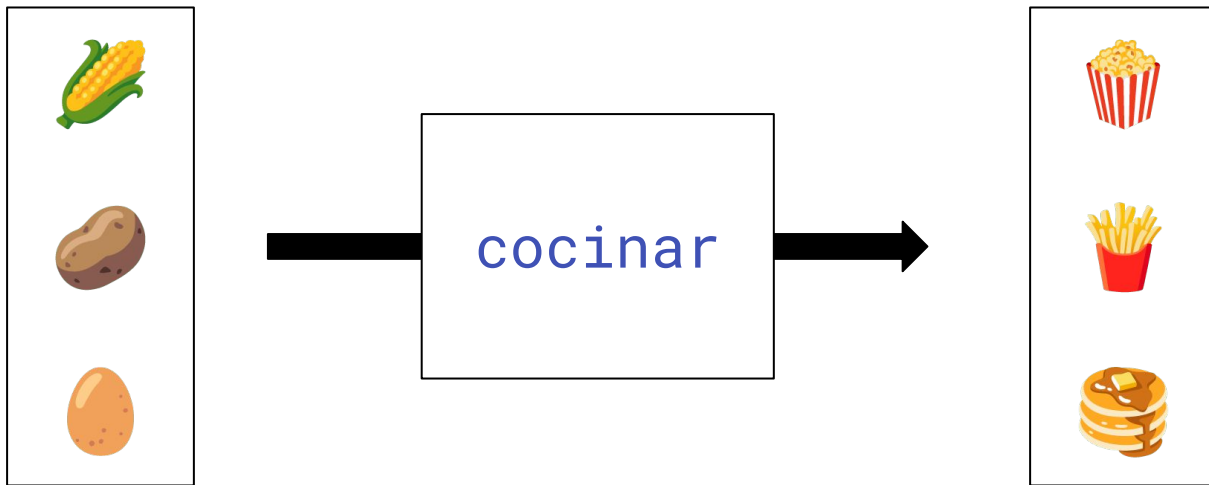


**Utilizar  
generadores**

---

# map

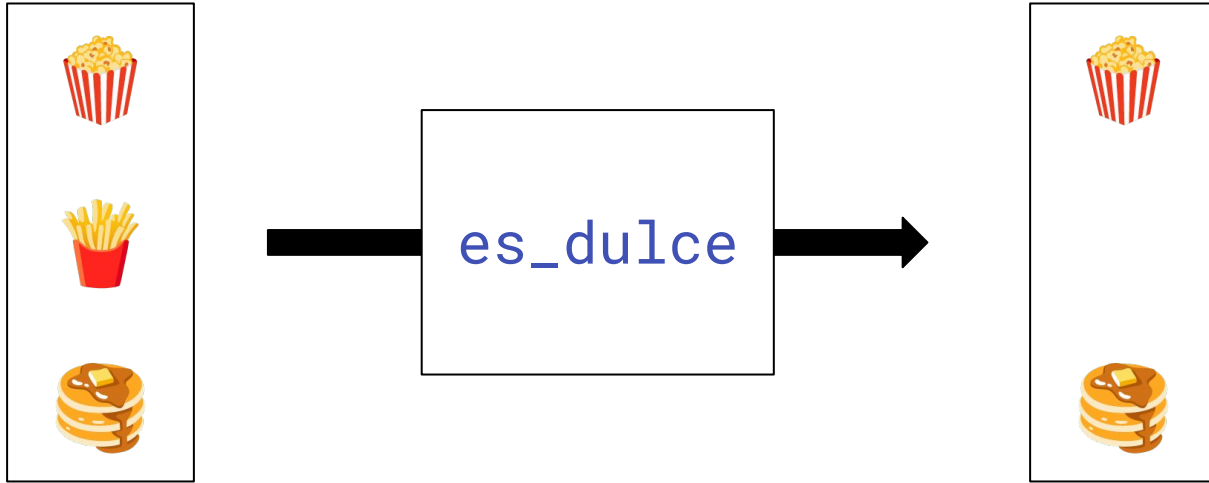
La función **map** aplica la **función** a cada elemento de un **iterable**.



```
map(cocinar, [🌽, 🥔, 🥚]) → [🍿, 🍟, 🥞]
```

# filter

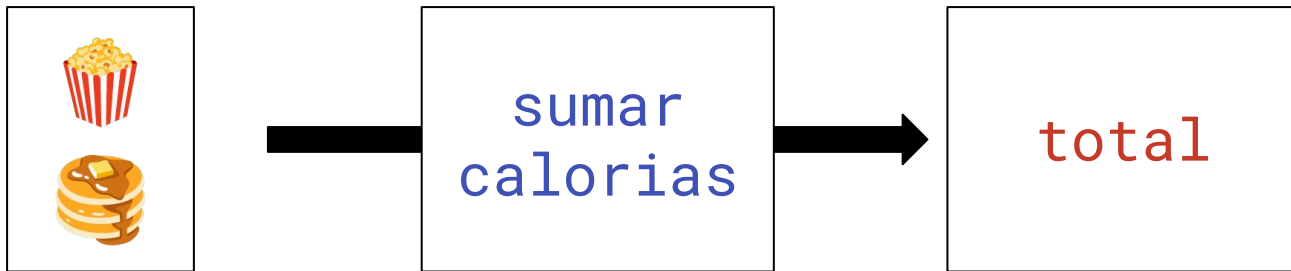
La función ***filter*** aplica la **función** para seleccionar elementos.



```
filter(es_dulce, [🍿, 🍟, 🥞]) → [🍿, 🥞]
```

# reduce

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

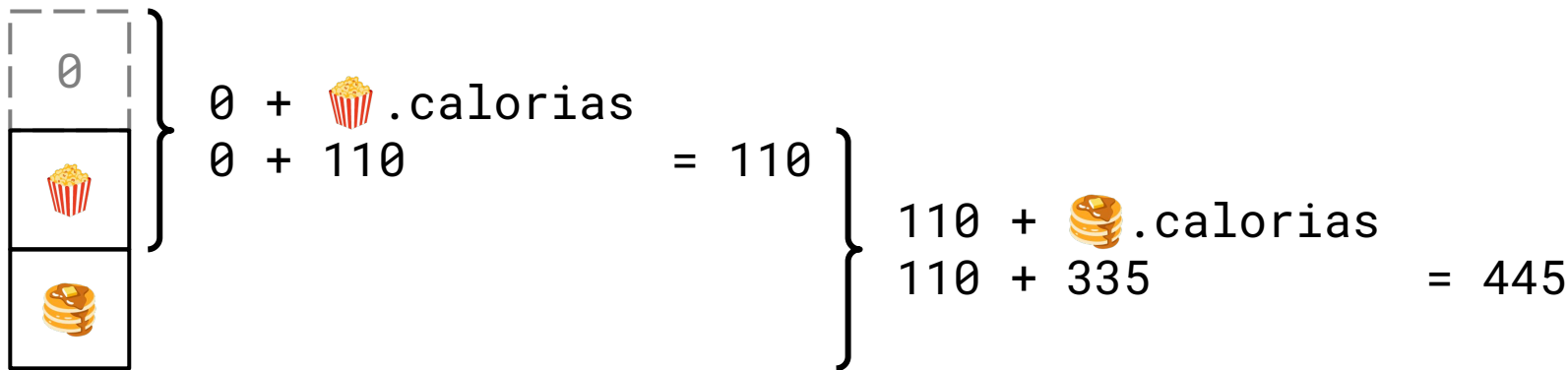


`reduce(sumar_calorias, [🍿, 🥞], 0) → (total)`

# reduce

```
reduce(sumar_calorias, [🍿, 🥞], 0)
```

```
def sumar_calorias(cal_acumuladas, alimento):  
    return cal_acumuladas + alimento.calorias
```



# ***Lambda functions***

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

```
lambda x:
```

## **Lambda**

```
lambda x: x * 2
```

```
lambda p: p.procesar()
```

```
lambda a, b: a + b
```

```
lambda a, p: a + p.precio
```

# Pregunta de Evaluación Escrita

## Tema: Programación Funcional (Examen 2024-1)

8. En el contexto de Programación Funcional, ¿qué es lo **primordial** que debe cumplir una función para que sea considerada una **función generadora**?
- A) Utilizar el comando **yield**.
  - B) Utilizar estructuras por comprensión.
  - C) No utilizar el comando **return**.
  - D) No utilizar los comandos **for** y **while**.
  - E) Retornar el resultado tras ejecutar las funciones **map**, **filter** y/o **reduce**.

# Pregunta de Evaluación Escrita

## Tema: Programación Funcional (Examen 2024-1)

8. En el contexto de Programación Funcional, ¿qué es lo **primordial** que debe cumplir una función para que sea considerada una **función generadora**?

A) Utilizar el comando **yield**.

B) Utilizar estructuras por comprensión.

C) No utilizar el comando **return**.

D) No utilizar los comandos **for** y **while**.

E) Retornar el resultado tras ejecutar las funciones **map**, **filter** y/o **reduce**.



# ***Programación Avanzada***

## **IIC2233 2024-2**

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán



# Comentarios AC3

NO GIT PUSH NO GAIN!