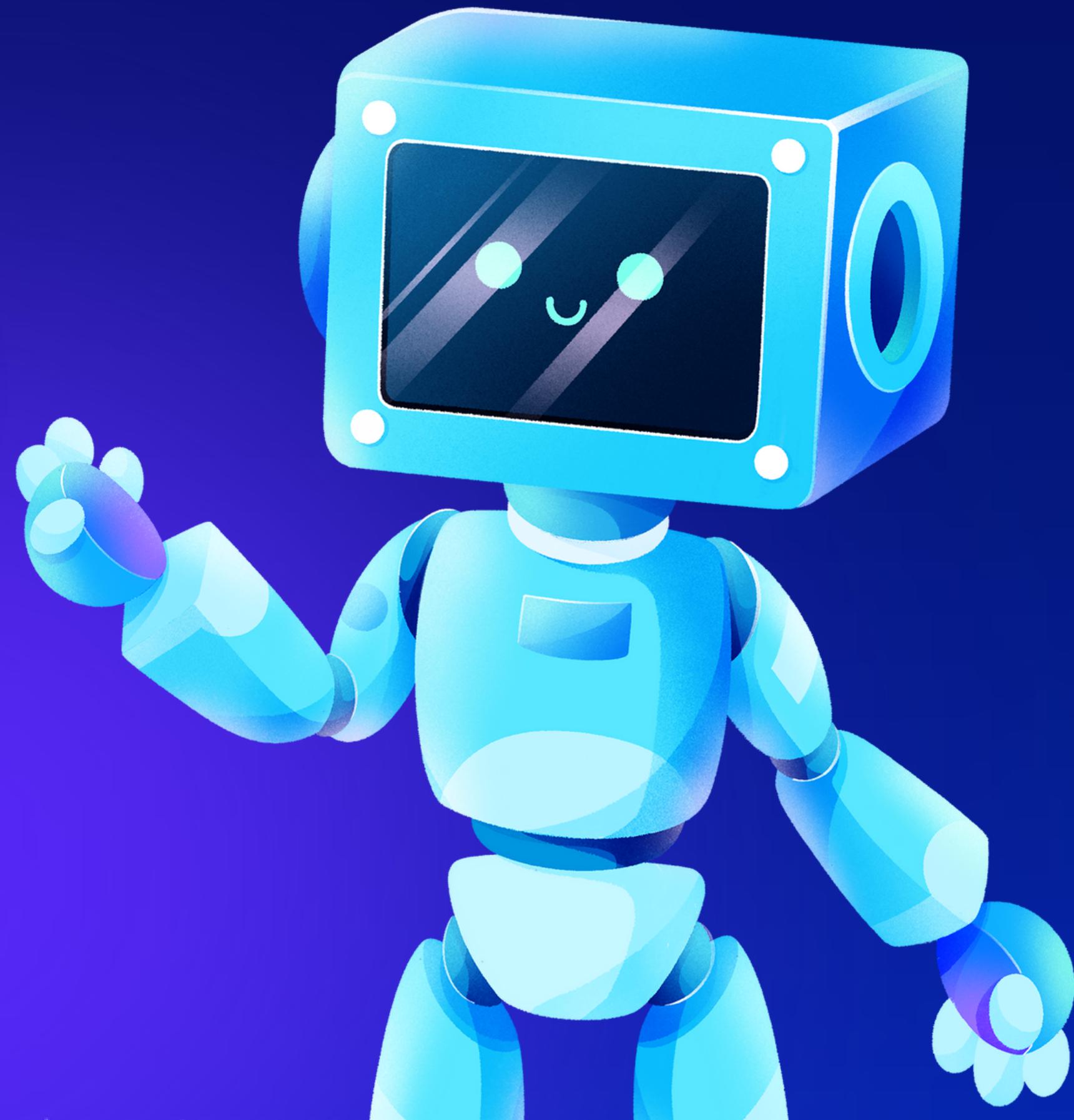


ALGORITMOS 2

UNIDAD 1

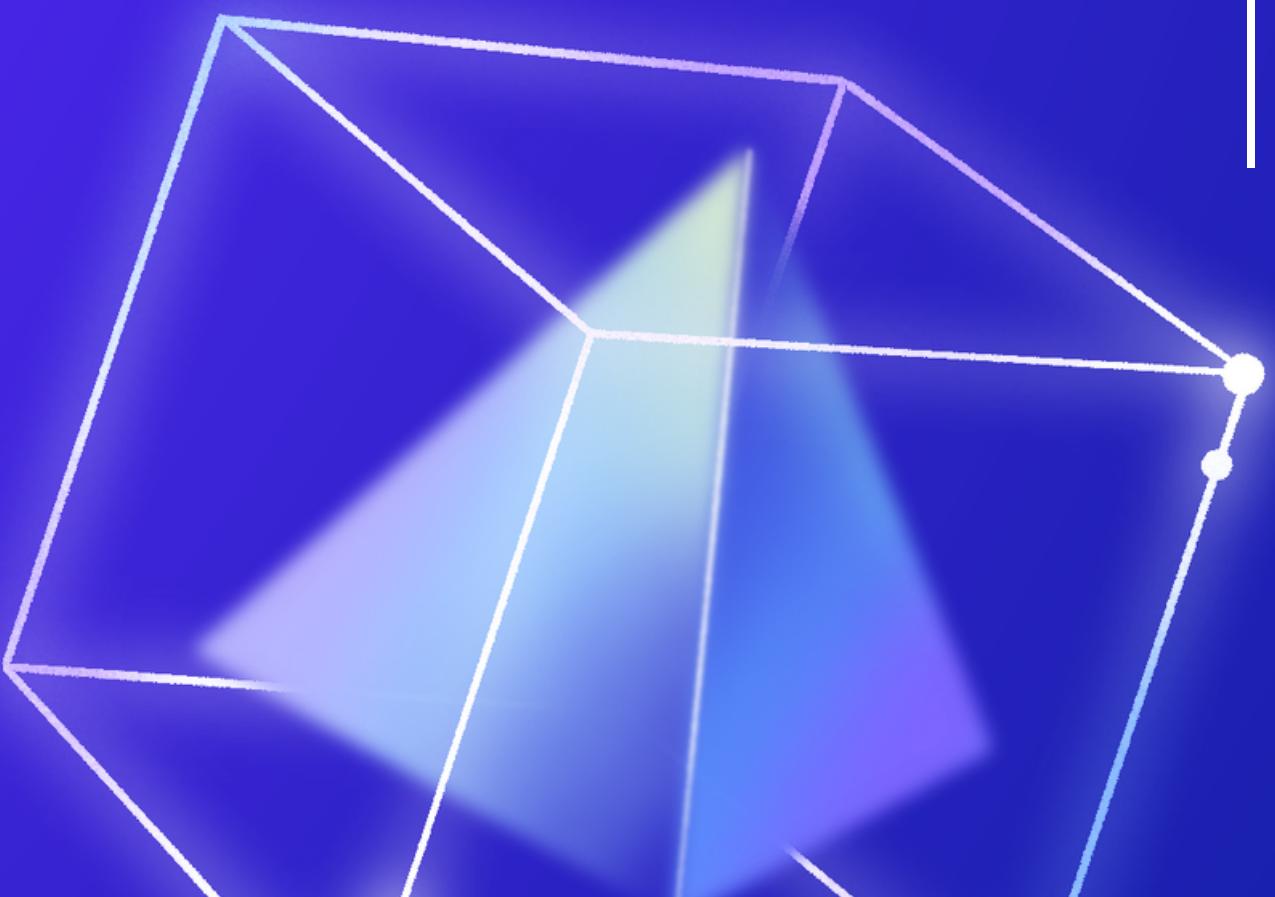
Licenciatura en Ciencias de Datos





ÍNDICE

- | | |
|----------------------------------|-----------|
| • Paradigma Funcional | <u>01</u> |
| • Funciones: Primera Clase | <u>02</u> |
| • Funciones: Orden Superior | <u>03</u> |
| • Funciones: Puras | <u>04</u> |
| • Funciones: Composición | <u>05</u> |
| • Funciones: Lambda | <u>06</u> |
| • Structural Pattern Matching | <u>07</u> |
| • Evaluación estricta y perezosa | <u>08</u> |
| • Inmutabilidad en Python | <u>09</u> |
| • Actividades | <u>10</u> |





PARADIGMA FUNCIONAL

Definición

Es un paradigma de tipo declarativo que se basa en un modelo matemático de composición de funciones.

En este modelo, el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado, tal como sucede en la composición de funciones matemáticas (cada cálculo se ve como una función matemática que hace corresponder entradas y salidas) .





PARADIGMA FUNCIONAL

Definición

Ventajas

Limitaciones

- Fácil de formular matemáticamente.
- Se enfoca en el logro (declarativo) y no en cómo lograrlo (imperativo).
- Simplicidad en el código y flujo.
- Rapidez en la codificación de los programas.
- Admite evaluación diferida: el valor solo se evalúa y almacena cuando sea necesario.

- Difícil de integrar con otras aplicaciones.
- Los valores inmutables en combinación con la recursividad pueden conllevar a una drástica reducción en el rendimiento del sistema.
- Los objetos puede que no representen el problema correctamente.



PARADIGMA FUNCIONAL

Imperativo

El concepto de declaratividad se contrapone a la tradicional programación imperativa en la cual el conocimiento y la lógica se encuentran muchas veces mezclados dentro de la misma porción de código resultando difícil determinar donde comienza uno y dónde termina el otro.

```
const numbers= [1,2,3,4,5,6,7,8,9,10]

// enfoque IMPERATIVO
let result= []
for (let i= 0; i <numbers.length; i++) {
  if (numbers[i]% 2 === 0) {
    if (numbers[i]<5) {
      result.push(5)
      Continue
    }
    result.push(numbers[i])
  }
}
```

OBJETIVO:

filtrar los números impares de una lista mientras se sustituyen por 5 los números pares menores de 5.

Es el mismo cálculo, con el mismo resultado. Sin embargo, como se puede ver, el código **imperativo** es detallado y no está claro de inmediato. Por otro lado, el enfoque **declarativo** es legible y explícito, porque se centra en lo que se quiere obtener.

Declarativo

En un programa construido de forma declarativa se produce una separación entre la descripción del problema por un lado y los algoritmos o estrategias para encontrar la solución por el otro.

```
// enfoque DECLARATIVO
numbers
.filter (n => n% 2 === 0)
.map (n => n <5 ? 5: n)
```



CARACTERÍSTICAS DEL PARADIGMA FUNCIONAL

- **Inmutabilidad:** en vez de modificar el estado de un objeto, se crean nuevos con el estado deseado. Todos los parámetros se pasan por valor y no por referencia.
- **Funciones puras:** no tienen efectos secundarios, siempre producen el mismo resultado para los mismos argumentos
- **Transparencia referencial:** siempre devuelve el mismo resultado
- **Evaluación perezosa:** demora la evaluación hasta el momento en que se necesite su valor
- **Recursión:** en vez de ciclos iterativos



FUNCIONES

Primera Clase

Cuando se pueden tratar como cualquier otro valor del lenguaje, es decir, cuando se pueden almacenar en variables, pasar como parámetro y devolver desde funciones, sin ningún tratamiento especial.

Por ejemplo, una función puede ser pasada como argumento a otras funciones, puede ser retornada por otra función, puede ser asignada a una variable o estructuras de datos.

Asignar función a una variable

JS

```
const foo = function () {  
    console.log("foobar");  
};  
  
// Invocación usando una variable  
foo();
```

Pasar la función como argumento

JS

```
function diHola() {  
    return "Hola ";  
}  
  
function saludar(saludo, nombre) {  
    console.log(saludo() + nombre);  
}  
  
// Pasamos `diHola` como argumento de la función `saludar`  
saludar(diHola, "JavaScript!");
```

Devolver una función

JS

```
function diHola() {  
    return function () {  
        console.log("¡Hola!");  
    };  
}
```



FUNCIONES

Orden Superior (viene de las matemáticas)

Ingresar por parámetro una variable y una función

```
def operation(x, fun):  
    return fun(x)
```

```
def squared(x):  
    return x**2
```

```
result_2 = operation(16, squared)  
print(result_2)
```

OBJETIVO:

obtener la raíz cuadrada de 16.

En la función anterior llamada operation, estamos recibiendo un parámetro de entrada x y adicionalmente, estamos recibiendo otro parámetro de entrada llamado fun, el cual será una función, por lo tanto esta función operation será una función de orden superior.

Adicionalmente note que dicha función también retorna una función como salida, según podemos ver en return fun(x)



FUNCIONES

Funciones de Orden Superior principales en Python:

- La función **filter()** devuelve un iterador donde los elementos se filtran a través de una función para probar si el elemento es aceptado o no.

```
filter(function, iterable)
```

- La función **map()** ejecuta una función específica para cada elemento en un iterable. El objeto se envía a la función como parámetro.

```
map(function, iterables)
```

- La función **reduce()** acepta una función y una secuencia y devuelve un único valor calculado:

```
reduce(function, iterables)
```

a. Inicialmente, se llama a la función con los dos primeros elementos de la secuencia y se devuelve el resultado.

b. A continuación, se vuelve a llamar a la función con el resultado obtenido en el paso 1 y el siguiente valor de la secuencia. Este proceso se repite hasta que hay elementos en la secuencia.



FUNCIONES

Puras

- 1) Cuando no tiene efectos secundarios observables, como alteración de las variables externas, cambios en el sistema de archivos, etc. No cambiarán expresamente ninguna variable de la que pudieran depender otras partes del código en algún momento.
- 2) Dados los mismos parámetros de entrada devuelve siempre el mismo valor.

Puras

Impuras

Pura: sin efectos secundarios

```
const add= (x1, x2) => {  
    return x1+x2  
}
```

Impura: modifica el estado externo

```
let contador= 0  
  
const incrementCounter= (n) => {  
    contador= contador + n  
    return contador  
}
```



FUNCIONES

Transparencia
Referencial

Permite que el valor que devuelve una función esté únicamente determinado por el valor de sus argumentos consiguiendo que una misma expresión tenga siempre el mismo valor. De esta manera, los valores resultantes son inmutables. No existe el concepto de cambio de estado.

Cuando el resultado devuelto por una función sólo depende de los argumentos que se le pasan en esa llamada.

Transparente

```
int sumarUno(int x){  
    return x + 1;  
}
```

Opaca

today();

La función o expresión en la que no podemos garantizar que el resultado dependa únicamente de sus argumentos. Esto ocurre cuando hay efectos secundarios involucrados



TIPOS COMUNES DE EFECTOS SECUNDARIOS

- 1) Modificación de variables globales
- 2) Modificación de argumentos si se pasan por referencia
- 3) Operaciones de I/O
- 4) Llamadas a funciones impuras, que vuelve a la función también impura
- 5) Generación de números aleatorios



FUNCIONES

Composición

La composición de funciones es la forma de combinar dos o más funciones de tal manera que la salida de una función se convierte en la entrada de la segunda función y así sucesivamente.

```
# Function to add 2
# to a number
def add(x):
    return x + 2

# Function to multiply
# 2 to a number
def multiply(x):
    return x * 2

# Printing the result of
# composition of add and
# multiply to add 2 to a number
# and then multiply by 2
print("Adding 2 to 5 and multiplying the result with 2: ", multiply(add(5)))
```

Sean dos funciones «F» y «G» y su composición se puede representar como $F(G(x))$ donde «x» es el argumento y la salida de la función $G(x)$ se convertirá en la entrada de $F()$ función.

Adding 2 to 5 and multiplying the result with 2: 14

Primero `add()` se llama a la función en la entrada 5. Segundo `add()` suma 2 a la entrada y la salida que es 7, se da como entrada a `multiply()` la que se multiplica por 2 y la salida es 14



FUNCIONES

Lambda

Son anónimas y pueden definir cualquier número de parámetros pero una única expresión; la cual es evaluada y devuelta. Se pueden usar en cualquier lugar en el que una función sea requerida y están restringidas al uso de una sola expresión.

```
lambda parameter(s):return value
```

lambda es la palabra clave que debe usar para definir una función lambda, seguida de una o más parámetros que debe tomar la función.

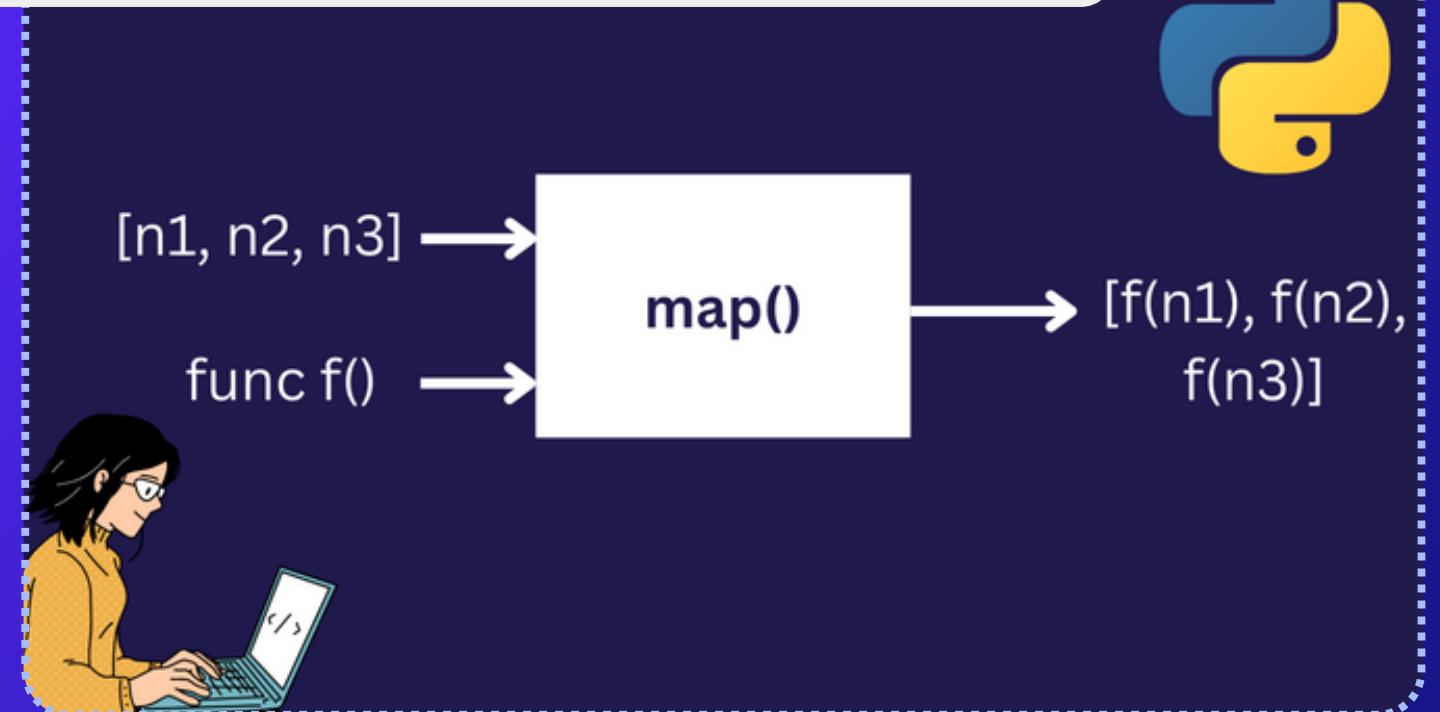
Con map()

OBJETIVO: toma un iterable y una función y aplica la función a cada elemento en el iterable.

Se crea la lista nums y usa el map() para crear una nueva lista que realiza el cuadrado de cada número en el nums lista.

La función lambda es utilizada para definir la operación de elevación al cuadrado.

```
>>> nums = [4,5,6,9]
>>> list(map(lambda num:num*num,nums))
[16, 25, 36, 81]
```





FUNCIONES

Lambda

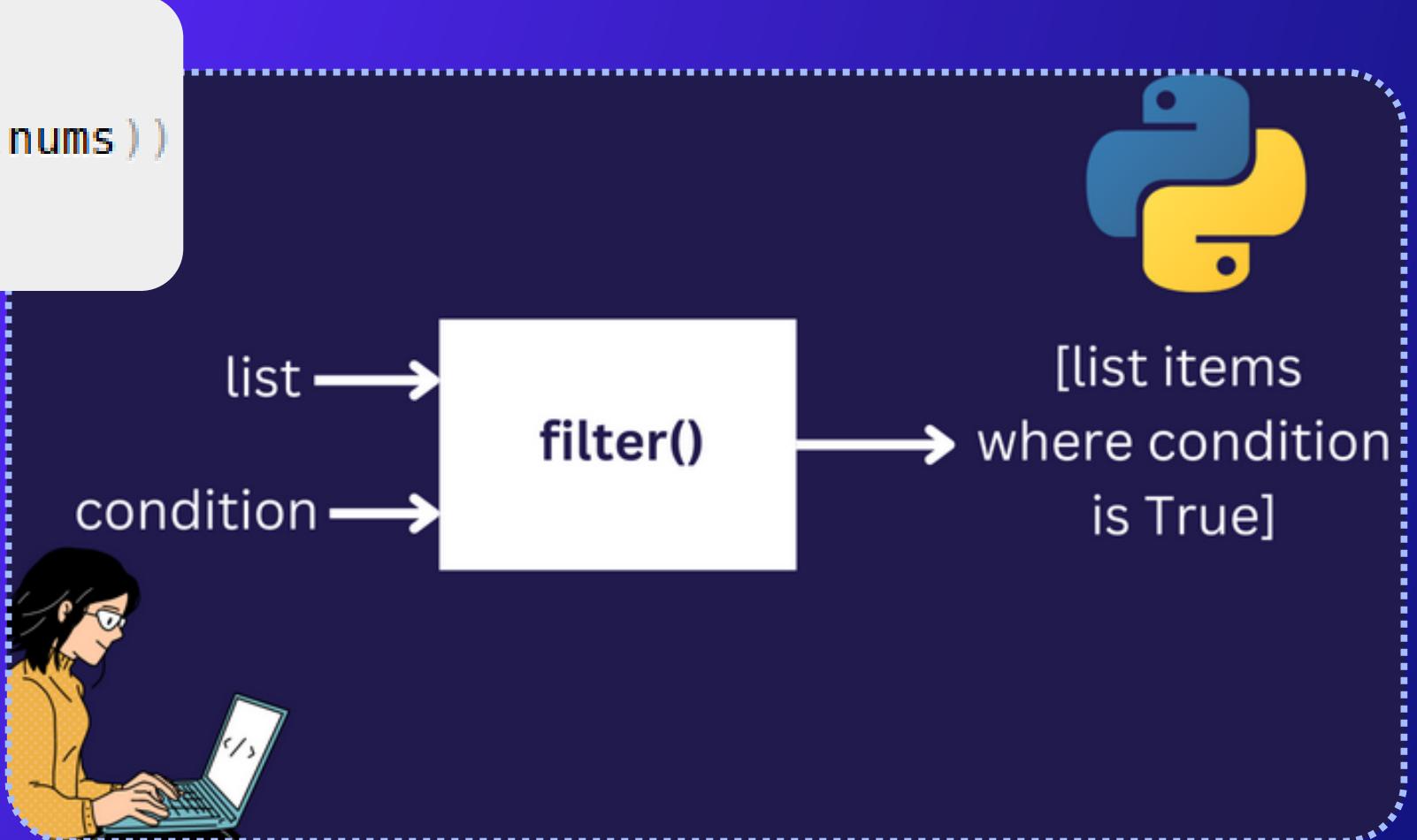
Con filter()

```
>>> nums = [4, 5, 6, 9]
>>> list(filter(lambda num: num%2!=0, nums ))
[5, 9]
```



OBJETIVO: toma un iterable y una función booleana y el resultado contiene solo los elementos del iterable original que satisfacen la condición.

Se crea la lista nums y usa el filter() para crear una nueva lista que evalúa si cada número en el nums lista es impar. La función lambda es utilizada para definir la condición True.





FUNCIONES

Lambda

Con reduce()

```
>>> from functools import reduce  
>>> nums = [4,5,6,9]  
>>> reduce(lambda num1,num2:num1+num2,nums)  
24
```

OBJETIVO: representa el concepto funcional de fold, donde se produce un valor a partir de la aplicación de una función acumuladora/combinadora/reductora sobre una estructura iterable.

Se crea la lista nums y usa el reduce() para calcular la suma de todos los números en la lista de números.

La función lambda es utilizada para definir la suma reductora: $f(f(f(4,5),6),9) = f(f(9,6),9) = f(15,9) = 24$.

