ntic master
revolucionamos la comunicación

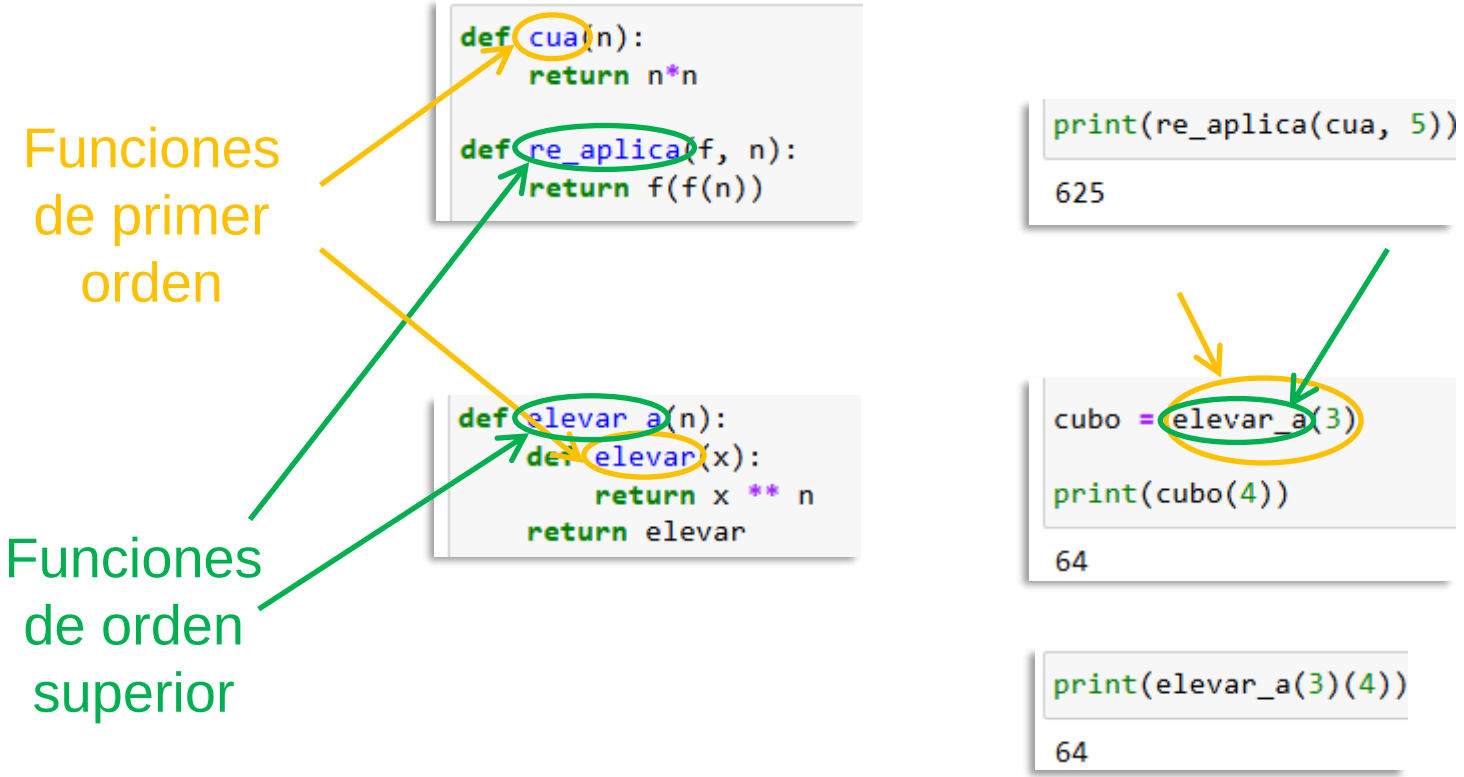
UNIVERSIDAD COMPLUTENSE DE MADRID

Programación. Python

Programación funcional y orden superior

Cristóbal Pareja Flores

Orden superior



Dos ejemplos más de f.o.s.

```
def componer(f, g):
    def fun_result(x):
        return f(g(x))
    return fun_result

print(componer(cua, cua)(3))
```

```
81
```

```
def integral(f, a, b):
    n = 100   # num de trozos iguales en que dividimos el intervalo [a, b]
    suma_acum = 0.0
    """
        Calcular el sumatorio
    """
    return suma_acum

def f(x):
    return x*x

def g(x):
    return 2*x - 1

print(integral(f, 0, 2))
print(integral(g, 0, 1))
```

```
0.0
0.0
```

Expresiones lambda

$$x \rightarrow x^2$$

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

```
# x -> x**2

print((lambda x : x**2)(5))
```

```
25
```

Expresiones lambda

$$x \to x^2$$

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

```
# x -> x**2

print((lambda x : x**2)(5))
```

```
25
```

$$x \to 2x^2 + 3x + 4$$

```
lambda x : 2*x*x + 3*x + 4
```

```
(lambda x : 2*x*x + 3*x + 4)(2.0)
```

```
18.0
```

Expresiones lambda

$$x \to x^2$$

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

```
# x -> x**2

print((lambda x : x**2)(5))
```

```
25
```

$$x \to 2x^2 + 3x + 4$$

```
lambda x : 2*x*x + 3*x + 4
```

```
(lambda x : 2*x*x + 3*x + 4)(2.0)
```

```
18.0
```

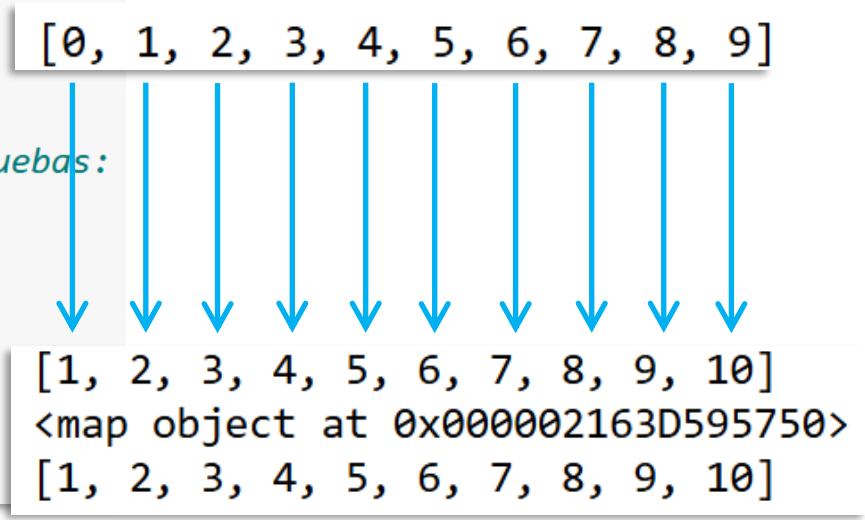
$$a, b, c \to (x \to ax^2 + bx + c)$$

```
fun_pol = lambda a, b, c : (lambda x : a*x*x + b*x + c)
f = fun_pol(2, 3, 4)
print(f(4))
print(fun_pol(1, 0, 1)(2))
```

```
48
5
```

F.O.S. predefinidas: map

```
def mymap(funcion, lista):
    lista_nueva = []
    for a in lista:
        lista_nueva.append(funcion(a))
    return lista_nueva

lista_a = list(range(10))
print(lista_a)

# Incremento en una unidad, para pruebas:

def incr_1(n):
    return n+1

print(mymap(incr_1, lista_a))
print(map(incr_1, lista_a))
print(list(map(incr_1, lista_a)))
```



F.O.S. predefinidas: map

```
lista_a = [1, 2, 3, 4, 5]
lista_b = map(lambda d : 10 / d, lista_a)
print(lista_b)
print(list(lista_b))

print("..............")

# lo siguiente produciría un error con evaluación impaciente:

lista_a = [5, 4, 3, 2, 1, 0]
lista_b = map(lambda d : 10 / d, lista_a)
print(lista_b)
for e in range(3):
    print(next(lista_b))
```

```
1D6D6BB8D90>
33333335, 2.5, 2.0]
```

```
000001D6D7BB5670>
```

```
3.333333333333335
```

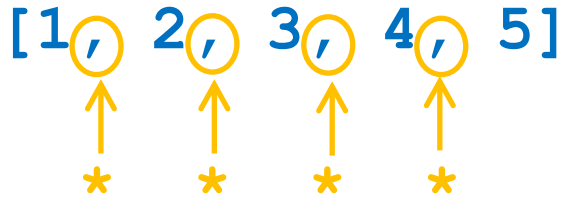
F.O.S. predefinidas: filter

```
def myfilter(predicado, lista):
    lista_nueva = []
    for a in lista:
        if predicado(a):
            lista_nueva.append(a)
    return lista_nueva

lista_a = list(range(10))
print(lista_a)

# La función es_par nos servirá para las pruebas:

def es_par(n):
    return n%2 == 0

print(myfilter(es_par, lista_a))
lista_b = list(filter(es_par, lista_a))
print(lista_b)
```

```
, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

F.O.S. predefinidas: reduce




```
def prod(a, b):
    return a*b

from functools import reduce

lista = range(1, 6)
print(list(lista))

fact = reduce(prod, range(1, 6))
print(fact)
```

```
[1, 2, 3, 4, 5]
120
```

Notación intensional para listas

```
print([i**2 for i in range(11)])                    # una expresión (sencilla) y un generador
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
print([(i, i**2) for i in range(11)])               # una expresión (una tupla) y un generador
```

```
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100)]
```

```
print([(i, i**2) for i in range(11) if i%2==0])     # una expresión, un generador y un filtro
```

```
[(0, 0), (2, 4), (4, 16), (6, 36), (8, 64), (10, 100)]
```

```
print([(i, j) for i in range(5) for j in range(3)])   # una expresión y dos generadores
```

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2),
(4, 0), (4, 1), (4, 2)]
```

```
print([(i, j) for i in range(10) for j in range(i)])  # una expresión y dos generadores, el segundo
```

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

Generador perezoso

```
# Generador (perezoso) definido con la notación intensional

lista = [5, 4, 3, 2, 1, 0]
generador = (10 / d for d in  lista_a)
print(generador)
for e in range(3):
    print(next(generador))

# OJO: la variable generador es una estructura perezosa:
# Si se evaluara de forma impaciente, daría lugar a un error
# Compruébalo tú mismo:

# lista_b = [10 / d for d in  lista_a]
```

```
<generator object <genexpr> at 0x000001D6D7BABC80>
2.0
2.5
3.3333333333333335
```

Listas infinitas. La función yield

```python
def natural_numbers():
    i = 0
    while True:
        yield i
        i = i+1

def tomar_n_valores_de(generador, n):
    cont = 0
    for i in generador:
        yield i
        cont = cont + 1
        if cont == n:
            return

lista = tomar_n_valores_de(natural_numbers
print(list(lista))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 2
5, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
```

```python
zeta = natural_numbers()

for i in zeta:
    print(i, end= " ")
    if i >= 50:
        print()
        break

print("----------------------------------")

for i in zeta:
    print(i, end= " ")
    if i > 100:
        print()
        break
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
----------------------------------
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96 97 98 99 100 101
```

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Programación. Python

Programación funcional y orden superior

Cristóbal Pareja Flores