Programación Funcional

En Python las funciones son objetos de primera clase, es decir, que pueden pasarse como argumentos de una función, al igual que el resto de los tipos de datos.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def aplica(funcion, argumento):
...     return funcion(argumento)
...
>>> def cuadrado(n):
...     return n*n
...
>>> def cubo(n):
...     return n**3
...
>>> aplica(cuadrado, 5)
25
>>> aplica(cubo, 5)
125
```

---

1. Funciones anónimas (`lambda`)

Existe un tipo especial de funciones que no tienen nombre asociado y se conocen como **funciones anónimas** o **funciones lambda**.

La sintaxis para definir una función anónima es

`lambda <parámetros> : <expresión>`

Estas funciones se suelen asociar a una variable o parámetro desde la que hacer la llamada.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> area = lambda base, altura : base * altura
>>> area(4, 5)
10
```

---

2. Aplicar una función a todos los elementos de una colección iterable (`map`)

`map(f,c)`: Devuelve una objeto iterable con los resultados de aplicar la función `f` a los elementos de la colección `c`. Si la función `f` requiere `n` argumentos entonces deben pasarse `n` colecciones con los argumentos. Para convertir el objeto en una lista, tupla o diccionario hay que aplicar explícitamente las funciones `list()`, `tuple()` o `dic()` respectivamente.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def cuadrado(n):
...     return n * n
...
>>> list(map(cuadrado, [1, 2, 3])
[1, 4, 9]
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def rectangulo(a, b):
...     return a * b
...
>>> tuple(map(rectangulo, (1, 2, 3), (4, 5, 6)))
(4, 10, 18)
```

3. Filtrar los elementos de una colección iterable (`filter`)

`filter(f,c)`: Devuelve una objeto iterable con los elementos de la colección `c` que devuelven `True` al aplicarles la función `f`. Para convertir el objeto en una lista, tupla o diccionario hay que aplicar explícitamente las funciones `list()`, `tuple()` o `dic()` respectivamente.

f debe ser una función que recibe un argumento y devuelve un valor booleano.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> def par(n):
...     return n %2 == 0
...
>>> list(filter(par, range(10))
[0, 2, 4, 6, 8]
```

4. Combinar los elementos de varias colecciones iterables (`zip`)

`zip(c1,c2, ...)`: Devuelve un objeto iterable cuyos elementos son tuplas formadas por los elementos que ocupan la misma posición en las colecciones `c1`, `c2`, etc. El número de elementos de las tuplas es el número de colecciones que se pasen. Para convertir el objeto en una lista, tupla o diccionario hay que aplicar explícitamente las funciones `list()`, `tuple()` o `dic()` respectivamente.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> asignaturas = ['Matemáticas', 'Física', 'Química', 'Economía']
>>> notas = [6.0, 3.5, 7.5, 8.0]
>>> list(zip(asignaturas, notas))
[('Matemáticas', 6.0), ('Física', 3.5), ('Química', 7.5), ('Economía', 8.0)]
>>> dict(zip(asignaturas, notas[:3]))
{'Matemáticas': 6.0, 'Física': 3.5, 'Química': 7.5}
```

5. Operar todos los elementos de una colección iterable (`reduce`)

`reduce(f,l)`: Aplicar la función `f` a los dos primeros elementos de la secuencia `l`. Con el valor obtenido vuelve a aplicar la función `f` en la lista. Devuelve el valor resultado de la última aplicación de la función `f`.

La función `reduce` está definida en el módulo `functools`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> from functools import reduce
>>> def producto(n, m):
...     return n * m
...
>>> reduce(producto, range(1, 5))
24
```