# Más sobre listas y diccionarios

Mathieu Kessler

Departamento de Matemática Aplicada y Estadística
Universidad Politécnica de Cartagena

@kessler_mathieu

# Más sobre listas

Vimos que, para acceder a elementos de listas, indexamos su posición (con corchetes)

```
>>> my_list = ['data', 'science', 'analytics', 'statistics', 'mathematics']
>>> print(my_list[1])
science
```

## Cortes (slices)

A menudo necesitamos extraer elementos consecutivos de una lista. Para ello, usamos los llamados cortes (*slicing*), which toman la forma:

`my_list[start:stop:step]`

y extrae los elementos desde la posición `start` hasta la posición `stop` (excluyendo ésta), con un paso de `step`.

- Si no se especifica, el tercer parámetro `step` toma el valor por defecto igual a 1.

  `my_list[1:3]` devolverá `['science', 'analytics']`
- Si no se especifica, el primer parámetro `start` toma el valor por defecto igual a 0.

  `my_list[:2]` devuelve `['data', 'science']`
- Si no se especifica el parámetro `stop`, el corte irá hasta el fina de la lista.

  `my_list[1:]` devuelve `['science', 'analytics', 'statistics', 'mathematics']`

# Algunas operaciones con listas

Las listas, junto con los diccionarios, son tipos de datos esenciales en Python.

Algunos comandos o métodos básicos para listas:

```
>>> my_list = [2, 1, -0.5, 0.287, 'abc']
>>> len(my_list) # devuelve el número de elementos
5
>>> my_list[4] = 'cba' # cambia el elemento en la posición 4
>>> my_list
[2, 1, -0.5, 0.287, 'cba']
>>> del my_list[2] # elimina el elemento en la posición 2, modifica la
>>> my_list
[2, 1, 0.287, 'cba']
>>> my_list.append('joe') # adjunta, modifica la lista.
>>> my_list
[2, 1, 0.287, 'cba', 'joe']
>>> my_list.insert(1, 9.2) # inserta en posición 1, modifica la lista.
>>> my_list
[2, 9.2, 1, 0.287, 'cba', 'joe']
```

# Cómo hacer una copia de una lista

Para hacer una copia de una lista, podríamos pensar en usar el operador de asignación "=":

```
>>> my_duplicate = my_list
>>> my_duplicate
[2, 1, -0.5, 0.287, 'abc']
```

Sin embargo, eso implica que `my_duplicate` que es el mismo objeto que `my_list`, refiriendo a la misma localización en memoria: si modifico una de ellas, la otra resulta modificada también:

```
>>> my_duplicate[0] = -1.2
>>> my_duplicate
[-1.2, 1, -0.5, 0.287, 'abc']
>>> my_list
[-1.2, 1, -0.5, 0.287, 'abc']
```

# Cómo hacer una copia de una lista

La solución es usar el método copy, que crea un nuevo objeto de tipo lista

```
>>> my_duplicate = my_list.copy()
>>> my_duplicate[0] = -1.2
>>> my_duplicate
[-1.2, 1, -0.5, 0.287, 'abc']
>>> my_list
[2, 1, -0.5, 0.287, 'abc']
```

Es también posible hacer una copia usando el operador de corte:

```
>>> my_duplicate = my_list[:]
>>> my_duplicate[0] = -1.2
>>> my_duplicate
[-1.2, 1, -0.5, 0.287, 'abc']
>>> my_list
[2, 1, -0.5, 0.287, 'abc']
```

# Iterar sobre diccionarios

## Obtener claves, valores o elementos de un diccionario

Los tres métodos `keys`, `values` y `items` permiten extraer respectivamente las claves, valores o elementos de un diccionario

```
>>> personas = {
...     'Pedro': 28,
...     'María': 21,
...     'Marta': 22
... }
>>> print(personas.keys())
dict_keys(['Pedro', 'María', 'Marta'])
>>> print(personas.values())
dict_values([28, 21, 22])
>>> print(personas.items())
dict_items([('Pedro', 28), ('María', 21), ('Marta', 22)])
```

# Iterar sobre diccionarios

Podemos iterar sobre un diccionario aprovechando `values` o `items`:

```
>>> personas = {
...     'Pedro': 28,
...     'María': 21,
...     'Marta': 22
... }
>>> for nombre, edad in personas.items():
...     print(f'{nombre} tiene {edad} años')
...
Pedro tiene 28 años
María tiene 21 años
Marta tiene 22 años
```

# Comprehensiones de listas o diccionarios

Una característica muy conveniente de Python es la construcción de "Comprehensiones" de listas o diccionarios (en inglés "list comprehension", "dictionary comprehension").

## Comprehensiones

Permiten construir listas o diccionarios usando el constructor con corchetes, pero aprovechando iteraciones y condiciones en la construcción. Las comprehensiones resultan consisas y legibles a la vez.

Por ejemplo, si tenemos una lista con las edades de Pedro, María y Marta:

```
>>> edades = [28, 21, 22]
```

Podemos construir una lista con sus edades dentro de 10 años:

```
>>> [edad + 10 for edad in edades]
[38, 31, 32]
```

Podemos incluso incorporar una consulta a nuestra comprehensión:

```
>>> [edad + 10 for edad in edades if edad < 25]
[31, 32]
```

# Comprehensiones de listas o diccionarios

Podemos proceder de manera similar para diccionarios:
Consideramos de nuevo el diccionario de personas:

```
>>> personas = {
...     'Pedro': 28,
...     'María': 21,
...     'Marta': 22
... }
```

Podemos construir un nuevo diccionario con sus edades dentro de 10 años:

```
>>> {nombre: (edad + 10) for nombre, edad in personas.items()}
{'Pedro': 38, 'María': 31, 'Marta': 32}
```

**Reto:**

Si consigo del INE la esperanza de vida adicional en años para cada de sus edades:

```
esperanza_adicional = {
      28: 53.4
      21: 65.6
      22: 64,5
}
```

Podríais construir un diccionario con la esperanza de vida de Pedro, María y Marta?