Arrays y listas

En Python no existen arrays propiamente dichos, es decir, no existe una estructura de datos de tamaño prefijado e inamovible donde añadir elementos localizados por un índice o posición. En su lugar, se utilizan **listas**, con un comportamiento similar pero flexible, donde podemos añadir y quitar elementos fácilmente en/de cualquier posición.

1. Definición de listas

Una lista en Python será una secuencia de elementos que *pueden ser de distintos tipos*. Se representa con los elementos separados por comas, entre corchetes. Por ejemplo:

```
datos = ['Nacho', 'Pepe', 2015, 2013]
```

Además, podemos crear una lista sin valores iniciales, usando corchetes vacíos o la instrucción `list`:

```
datos = []
datos = list()
```

Una lista puede tener tipos simples (textos, números, etc) o tipos complejos (como por ejemplo, otras listas, u objetos de clases como veremos en otros documentos del curso):

```
datos = [
    ["Nacho", 40, 233],
    ["Pepe", 70, 231],
    ...
]
```

2. Acceder a los elementos de una lista

Como hemos dicho, los elementos de una lista se referencian por un índice o posición numérica, empezando en cero. Así, si tenemos una lista como esta:

```
datos = ['Nacho', 'Pepe', 2015, 2013]
```

y ejecutamos la instrucción `print(datos[1])`, se imprimirá por pantalla el segundo elemento de la lista, que es *Pepe*. Además, Python ofrece la posibilidad de imprimir una lista entera usando la misma instrucción `print`. En este caso, se mostrarían los datos tal cual están almacenados, separados por comas, y entre corchetes:

```
print(datos[1])     # Pepe
print(datos)        # ['Nacho', 'Pepe', 2015, 2013]
```

3. Añadir, modificar y borrar elementos

Si queremos añadir un elemento **al final** de la lista usamos la instrucción `append`. Por ejemplo:

```
datos.append(2017)
```

En el caso de querer insertar un elemento **en una posición determinada**, usamos la instrucción `insert`, indicando en qué posición queremos insertar, y el dato que queremos insertar. Automáticamente, todos los elementos a la derecha de esa posición se desplazarán para hacer hueco al nuevo elemento.

```
datos.insert(3, 1996)
```

También podemos **actualizar el valor** de un dato de la lista, simplemente indicando su posición y el nuevo valor que queremos asignar:

```
datos[2] = 2016
```

Finalmente, si queremos **eliminar** un dato de la lista, usamos la instrucción `del`, seguida del elemento que queremos borrar.

```
del datos[2]
```

Alternativamente, podemos usar la instrucción `remove` de la propia lista, indicando la posición a borrar:

```
datos.remove(2)
```

4. Otras operaciones simples

Hay otras operaciones que nos pueden resultar útiles sobre una lista. Por ejemplo, la instrucción `len(lista)` devuelve el número de elementos actualmente almacenado en la lista (tamaño de la lista):

```
datos = [1, 2, 3, 4]
print(len(datos))       # 4
```

La instrucción `list(valor)` convierte un elemento en una lista de valores.

```
datos = list('123')
print(datos)            # ['1', '2', '3']
```

La operación `lista1 + lista2` concatena los datos de dos listas

```
datos1 = [1, 2, 3, 4]
datos2 = [4, 5, 6]
datosTotales = datos1 + datos2  # [1, 2, 3, 4, 4, 5, 6]
```

La operación `lista * N` genera una nueva lista donde los elementos de la lista original aparecen repetidos N veces

```
datos = [4, 5, 6]
datosRepetidos = datos * 3      # [4, 5, 6, 4, 5, 6, 4, 5, 6]
```

La expresión `n in lista` comprueba si el dato *n* está en la lista

```
datos = [4, 5, 6]
if 4 in datos:
    print("Existe el dato 4")
```

Las instrucciones `max(lista)` y `min(lista)` obtienen el mayor / menor valor de la lista, respectivamente

```
datos = [4, 5, 6]
print(max(datos))       # 6
```

La instrucción `lista.count(objeto)` obtiene el número de apariciones del objeto en la lista

```
datos = [4, 5, 6, 4]
print(datos.count(4))   # 2
```

La instrucción `lista.index(objeto)` obtiene la posición donde aparece por primera vez el objeto en la lista

```
datos = [4, 5, 6, 4]
print(datos.index(4))   # 0
```

Finalmente, la instrucción `lista.reverse()` invierte el orden de la lista. Este método NO devuelve una nueva lista, sino que afecta a la original. Si queremos mantener el orden original y que la lista invertida sea otra nueva, podemos usar la instrucción `lista[::-1]` y asignar el valor a otra variable

```
datos = [1, 2, 3, 4]
datos.reverse()                # [4, 3, 2, 1]
datosInvertidos = datos[::-1]  # [1, 2, 3, 4]
```

> **Ejercicio 1**:
>
> Crea un programa llamado `ListaInvertida.py` que le pida al usuario que introduzca un conjunto de nombres separados por comas, y le muestre por pantalla la misma lista en orden inverso.

> **Ejercicio 2**:
>
> Crea un programa llamado `Loteria.py` que le pida al usuario que introduzca los 6 números que juega a la lotería (separados por espacios). Entonces, deberá crear una lista con ellos, ordenarla ascendentemente e imprimirla en pantalla. Además, el programa debe indicar si es una lista válida (es decir, los números deben estar entre 1 y 49, inclusive, sin repetirse). Por ejemplo:

```
Introduce los 6 números de la lotería separados por espacios
1 20 12 20 6 50
[1, 6, 12, 20, 20, 50]
La lista NO es válida:
Hay números repetidos
Hay números menores que 1 o mayores que 49
```

5. Algunas operaciones avanzadas con listas

En este apartado veremos algunas operaciones algo más complejas que se pueden realizar con listas.

5.1. Ordenación de listas

La instrucción `lista.sort(funcion)` ordena una lista según el criterio especificado en la función indicada. Para listas de datos simples (listas de enteros, de strings...) no hace falta indicar ninguna función: las listas se ordenan automáticamente de menor a mayor. Si queremos una ordenación inversa, usamos un parámetro adicional llamado `reverse`:

```
datos = [4, 2, 7, 5]
datos.sort()                # [2, 4, 5, 7]
datos.sort(reverse=True)    # [7, 5, 4, 2]
```

Esta instrucción afecta a la lista original, que queda automáticamente ordenada por el criterio que hayamos elegido. Si queremos ordenar algunos datos más complejos (como objetos, o tuplas, debemos proporcionar una función que indique el criterio de comparación). Por ejemplo, esta lista de tuplas queda ordenada ascendentemente por su edad (segundo campo):

```
def ordenarPorEdad(persona):
    return (persona[1])

gente = [("John Doe", 36, "611223344"),
         ("Mary Stewart", 54, "733445566"),
         ...]
gente.sort(key=ordenarPorEdad)
```

5.2. Mapeo de listas

La instrucción `map` aplica una función de transformación a una lista y devuelve los elementos transformados. Estos elementos pueden formar de nuevo una lista usando la instrucción `list`. Este ejemplo obtiene una lista con los cuadrados de los números de la lista original:

```
def cuadrado(x):
    return (x * x)

lista = [1, 2, 3, 4]
cuadrados = list(map(cuadrado, lista))
```