Listas

Una **lista** es una secuencia ordenadas de objetos de distintos tipos.
Se construyen poniendo los elementos entre corchetes `[ ]` separados por comas.
Se caracterizan por:

- Tienen orden.
- Pueden contener elementos de distintos tipos.
- Son mutables, es decir, pueden alterarse durante la ejecución de un programa.

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

```
# Lista vacía
>>> type([])
<class 'list'>
# Lista con elementos de distintos tipos
>>> [1, "dos", True]
# Listas anidadas
>>> [1, [2, 3], 4]
```

---

1. Creación de listas mediante la función `list()`

Otra forma de crear listas es mediante la función `list()`.

- `list(c)`: Crea una lista con los elementos de la secuencia o colección `c`.

Se pueden indicar los elementos separados por comas, mediante una cadena, o mediante una colección de elementos iterable.

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

```
>>> list()
[]
>>> list(1, 2, 3)
[1, 2, 3]
>>> list("Python")
['P', 'y', 't', 'h', 'o', 'n']
```

---

2. Acceso a los elementos de una lista

Se utilizan los mismos operadores de acceso que para cadenas de caracteres.

- `l[i]`: Devuelve el elemento de la lista `l` con el índice `i`.

El índice del primer elemento de la lista es 0.

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

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> a[0]
'P'
>>> a[5]
'n'
>>> a[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> a[-1]
'n'
```

3. Sublistas

- `l[i:j:k]`: Devuelve la sublista desde el elemento de `l` con el índice `i` hasta el elemento anterior al índice `j`, tomando elementos cada `k`.

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

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> a[1:4]
['y', 't', 'h']
>>> a[1:1]
[]
>>> a[:-3]
['y', 't', 'h']
>>> a[:]
['P', 'y', 't', 'h', 'o', 'n']
>>> a[0:6:2]
['P', 't', 'o']
```

4. Operaciones que no modifican una lista

- `len(l)`: Devuelve el número de elementos de la lista `l`.
- `min(l)`: Devuelve el mínimo elemento de la lista `l` siempre que los datos sean comparables.
- `max(l)`: Devuelve el máximo elemento de la lista `l` siempre que los datos sean comparables.
- `sum(l)`: Devuelve la suma de los elementos de la lista `l`, siempre que los datos se puedan sumar.
- `dato in l`: Devuelve `True` si el dato `dato` pertenece a la lista `l` y `False` en caso contrario.
- `l.index(dato)`: Devuelve la posición que ocupa en la lista `l` el primer elemento con valor `dato`.
- `l.count(dato)`: Devuelve el número de veces que el valor `dato` está contenido en la lista `l`.
- `all(l)`: Devuelve `True` si todos los elementos de la lista `l` son `True` y `False` en caso contrario.
- `any(l)`: Devuelve `True` si algún elemento de la lista `l` es `True` y `False` en caso contrario.

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

```
>>> a = [1, 2, 2, 3]
>>> len(a)
4
>>> min(a)
1
>>> max(a)
3
>>> sum(a)
8
>>> 3 in a
True
>>> a.index(2)
1
>>> a.count(2)
2
>>> all(a)
True
>>> any([0, False, 3<2])
False
```

5. Operaciones que modifican una lista

- `l1 + l2`: Crea una nueva lista concatenan los elementos de la listas `l1` y `l2`.
- `l.append(dato)`: Añade `dato` al final de la lista `l`.
- `l.extend(sequencia)`: Añade los datos de sequencia al final de la lista `l`.
- `l.insert(índice, dato)`: Inserta `dato` en la posición `índice` de la lista `l` y desplaza los elementos una posición a partir de la posición `índice`.
- `l.remove(dato)`: Elimina el primer elemento con valor `dato` en la lista `l` y desplaza los que están por detrás de él una posición hacia delante.
- `l.pop([índice])`: Devuelve el dato en la posición `índice` y lo elimina de la lista `l`, desplazando los elementos por detrás de él una posición hacia delante.
- `l.sort()`: Ordena los elementos de la lista `l` de acuerdo al orden predefinido, siempre que los elementos sean comparables.
- `l.reverse()`: invierte el orden de los elementos de la lista `l`.

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

```
>>> a = [1, 3]
>>> b = [2 , 4, 6]
>>> a.append(5)
>>> a
[1, 3, 5]
>>> a.remove(3)
>>> a
[1, 5]
>>> a.insert(1, 3)
>>> a
[1, 3, 5]
```

```
>>> b.pop()
6
>>> c = a + b
>>> c
[1, 3, 5, 2, 4]
>>> c.sort()
>>> c
[1, 2, 3, 4, 5]
>>> c.reverse()
>>> c
[5, 4, 3, 2, 1]
```

6. Copia de listas

Existen dos formas de copiar listas:

- Copia por referencia `l1 = l2`: Asocia la la variable `l1` la misma lista que tiene asociada la variable `l2`, es decir, ambas variables apuntan a la misma dirección de memoria. Cualquier cambio que hagamos a través de `l1` o `l2` afectará a la misma lista.
- Copia por valor `l1 = list(l2)`: Crea una copia de la lista asociada a `l2` en una dirección de memoria diferente y se la asocia a `l1`. Las variables apuntan a direcciones de memoria diferentes que contienen los mismos datos. Cualquier cambio que hagamos a través de `l1` no afectará a la lista de `l2` y viceversa.

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

```
>>> a = [1, 2, 3]
>>> # copia por referencia
>>> b = a
>>> b
[1, 2, 3]
>>> b.remove(2)
>>> b
[1, 3]
>>> a
[1, 3]
```

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

```
>>> a = [1, 2, 3]
>>> # copia por referencia
>>> b = list(a)
>>> b
[1, 2, 3]
>>> b.remove(2)
>>> b
[1, 3]
>>> a
[1, 2, 3]
```