**Listas**

Las listas son una estructura de datos muy flexible. Python permite manipular listas de muchas maneras. En esta lección aprenderás algunas de ellas:

**Listas**

Las listas son conjuntos ordenados de elementos (números, cadenas, listas, etc). Las listas se delimitan por corchetes ([ ]) y los elementos se separan por comas.

Las listas pueden contener elementos del mismo tipo:

>>> primos = [2, 3, 5, 7, 11, 13]

>>> diasLaborables = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]

O pueden contener elementos de tipos distintos:

>>> fecha = ["Lunes", 27, "Octubre", 1997]

O pueden contener listas:

>>> peliculas = [ ["Senderos de Gloria", 1957], ["Hannah y sus hermanas", 1986]]

Las listas pueden tener muchos niveles de anidamiento:

>>> directores = [ ["Stanley Kubrick", ["Senderos de Gloria", 1957]], ["Woody Allen", ["Hannah y sus hermanas", 1986]] ]

Las variables de tipo lista hacen referencia a la lista completa.

>>> lista = [1, "a", 45]

>>> lista

# [1, 'a', 45]

Una lista que no contiene ningún elemento se denomina **lista vacía**:

>>> lista = [ ]

>>> lista

# [ ]

Al definir una lista se puede hacer referencia a otras variables.

>>> nombre = "Pepe"

>>> edad = 25

>>> lista = [nombre, edad]

>>> lista

# ['Pepe', 25]

Como siempre, hay que tener cuidado al modificar una variable que se ha utilizado para definir otras variables, porque esto puede afectar al resto de variables:

**Nota**: Este punto se trata en la [lección Variables 2](https://www.mclibre.org/consultar/python/lecciones/python-variables-2.html).

Si se trata objetos inmutables, el resto de variables no resultan afectadas, como muestra el siguiente ejemplo:

>>> nombre = "Pepe"

>>> edad = 25

>>> lista = [nombre, edad]

>>> lista

# ['Pepe', 25]

>>> nombre = "Juan"

>>> lista

# ['Pepe', 25]

* Pero si se trata de objetos mutables y al modificar la variable se modifica el objeto, el resto de variables sí resultan afectadas, como muestra el siguiente ejemplo:

>>> nombres = ["Ana", "Bernardo"]

>>> edades = [22, 21]

>>> lista = [nombres, edades]

>>> lista

[['Ana', 'Bernardo'], [22, 21]]

>>> nombres += ["Cristina"]

>>> lista

[['Ana', 'Bernardo', 'Cristina'], [22, 21]]

Una lista puede contener listas (que a su vez pueden contener listas, que a su vez etc.):

>>> persona1 = ["Ana", 25]

>>> persona2 = ["Benito", 23]

>>> lista = [persona1, persona2]

>>> lista

# [['Ana', 25], ['Benito', 23]]

Se puede acceder a cualquier elemento de una lista escribiendo el nombre de la lista y entre corchetes el número de orden en la lista. El primer elemento de la lista es el número 0.

|  |
| --- |
| >>> lista = [10, 20, 30, 40]  >>> lista[2]  30  >>> lista[0]  10 |

Se pueden concatenar dos listas utilizando la operación suma.

>>> lista1 = [10, 20, 30, 40]

>>> lista2 = [30, 20]

>>> lista = lista1 + lista2 + lista1

>>> lista

# [10, 20, 30, 40, 30, 20, 10, 20, 30, 40]

# Concatenar listas

Las listas se pueden concatenar con el símbolo de la suma (+):

>>> vocales = ["E", "I", "O"]

>>> vocales

## ['E', 'I', 'O']

>>> vocales = vocales + ["U"]

>>> vocales

## ['E', 'I', 'O', 'U']

>>> vocales = ["A"] + vocales

>>> vocales

## ['A', 'E', 'I', 'O', 'U']

El operador suma (+) necesita que los dos operandos sean listas:

|  |
| --- |
| >>> vocales = ["E", "I", "O"]  >>> vocales = vocales + "Y"  Traceback (most recent call last): File "<pyshell#2>", line 1, in <module> vocales = vocales + "Y"  TypeError: can only concatenate list (not "str") to list |

También se puede utilizar el operador += para añadir elementos a una lista:

>>> vocales = ["A"]

>>> vocales += ["E"]

>>> vocales

## ['A', 'E']

Aunque en estos ejemplos, los operadores + y += den el mismo resultado, no son equivalentes, como se explica en la lección de [Variables 2](https://www.mclibre.org/consultar/python/lecciones/python-variables-2.html" \l "variables-objeto-mutable).

# Manipular elementos individuales de una lista

Cada elemento se identifica por su posición en la lista, teniendo en cuenta que se empieza a contar por 0.

|  |
| --- |
| >>> fecha = [27, "Octubre", 1997]  >>> fecha[0]  27  >>> fecha[1]  Octubre  >>> fecha[2]  1997 |

No se puede hacer referencia a elementos fuera de la lista:

>>> fecha = [27, "Octubre", 1997]

>>> fecha[3]

Traceback (most recent call last): File "<pyshell#3>", line 1, in <module> fecha[3]

Index error: list index out of range

Se pueden utilizar números negativos (el último elemento tiene el índice -1 y los elementos anteriores tienen valores descendentes):

|  |
| --- |
| >>> fecha = [27, "Octubre", 1997]  >>> fecha[-1]  1997  >>> fecha[-2]  Octubre  >>> fecha[-3]  27 |

Se puede modificar cualquier elemento de una lista haciendo referencia a su posición:

|  |
| --- |
| >>> fecha = [27, "Octubre", 1997]  >>> fecha[2] = 1998  >>> fecha[0]  27  >>> fecha[1]  Octubre  >>> fecha[2]  1998 |

# Manipular sublistas

De una lista se pueden extraer sublistas, utilizando la notación nombreDeLista[inicio:límite], donde inicio y límite hacen el mismo papel que en el tipo range(inicio, límite).

>>> dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]

>>> dias[1:4] # Se extrae una lista con los valores 1, 2 y 3

['Martes', 'Miércoles', 'Jueves']

>>> dias[4:5] # Se extrae una lista con el valor 4

['Viernes']

>>> dias[4:4] # Se extrae una lista vacía

[]

>>> dias[:4] # Se extrae una lista hasta el valor 4 (no incluido)

['Lunes', 'Martes', 'Miércoles', 'Jueves']

>>> dias[4:] # Se extrae una lista desde el valor 4 (incluido)

['Viernes', 'Sábado', 'Domingo']

>>> dias[:] # Se extrae una lista con todos los valores

['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo']

Se puede modificar una lista modificando sublistas. De esta manera se puede modificar un elemento o varios a la vez e insertar o eliminar elementos.

|  |
| --- |
| >>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  >>> letras[1:4] = ["X"] # Se sustituye la sublista ['B','C','D'] por ['X'] >>> letras  ['A', 'X', 'E', 'F', 'G', 'H']  >>> letras[1:4] = ["Y", "Z"] # Se sustituye la sublista ['X','E','F'] por ['Y','Z']  ['A','Y', 'Z', 'G', 'H']  >>> letras[0:1] = ["Q"] # Se sustituye la sublista ['A'] por ['Q']  >>> letras  ['Q', 'Y', 'Z', 'G', 'H']  >>> letras[3:3] = ["U", "V"] # Inserta la lista ['U','V'] en la posición 3  >>> letras  ['Q', 'Y', 'Z', 'U', 'V', 'G', 'H']  >>> letras[0:3] = [] # Elimina la sublista ['Q','Y', 'Z']  >>> letras  ['U', 'V', 'G', 'H'] |

Al definir sublistas, Python acepta valores fuera del rango, que se interpretan como extremos (al final o al principio de la lista).

|  |
| --- |
| >>> letras = ["D", "E", "F"]  >>> letras[3:3] = ["G", "H"] # Añade ["G", "H"] al final de la lista  >>> letras  ['D', 'E', 'F', 'G', 'H']  >>> letras[100:100] = ["I", "J"] # Añade ["I", "J"] al final de la lista  >>> letras  ['D', 'E', 'F', 'G', 'H', 'I', 'J']  >>> letras[-100:-50] = ["A", "B", "C"] # Añade ["A", "B", "C"] al principio de la lista  >>> letras  ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'] |

# La palabra reservada del

La palabra reservada del permite eliminar un elemento o varios elementos a la vez de una lista, e incluso la misma lista.

|  |
| --- |
| >>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  >>> del letras[4] # Elimina la sublista ['E']  >>> letras  ['A', 'B', 'C', 'D', 'F', 'G', 'H']  >>> del letras[1:4] # Elimina la sublista ['B', 'C', 'D']  >>> letras  ['A', 'F', 'G', 'H']  >>> del letras # Elimina completamente la lista  >>> letras  Traceback (most recent call last): File "<pyshell#1>", line 1, in ? letras  NameError: name 'letras' is not defined |

Si se intenta borrar un elemento que no existe, se produce un error:

|  |
| --- |
| >>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  >>> del letras[10]  Traceback (most recent call last): File "<pyshell#1>", line 1, in <module> del letras[10]  IndexError: list assignment index out of range |

Aunque si se hace referencia a sublistas, Python sí que acepta valores fuera de rango, pero lógicamente no se modifican las listas.

>>> letras = ["A", "B", "C", "D", "E", "F", "G", "H"]

>>> del letras[100:200] # No elimina nada

>>> letras

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Copiar una lista

Con variables de tipo entero, decimal o de cadena, es fácil tener una copia de una variable para conservar un valor que en la variable original se ha perdido:

>>> a = 5

>>> b = a # Hacemos una copia del valor de a

>>> a, b

## (5, 5)

>>> a = 4 # de manera que aunque cambiemos el valor de a ...

>>> a, b # ... b conserva el valor anterior de a en caso de necesitarlo (4, 5)

Pero si hacemos esto mismo con listas, nos podemos llevar un sorpresa:

>>> lista1 = ["A", "B", "C"]

>>> lista2 = lista1 # Intentamos hacer una copia de la lista lista1

>>> lista1, lista2

## (['A', 'B', ,'C'] ['A', 'B', ,'C'])

>>> del lista1[1] # Eliminamos el elemento ['B'] de la lista lista1 ...

>>> lista1, lista2 # ... pero descubrimos que también ha desaparecido de la lista lista2 (['A', 'C'] ['A', 'C'])

El motivo de este comportamiento (que se explica con más detalle en la lecció[nVariables 2](https://www.mclibre.org/consultar/python/lecciones/python-variables-2.html)), es que los enteros, decimales y cadenas son objetos inmutables y las listas son objetos mutables.

Si queremos copiar una lista, de manera que conservemos su valor aunque modifiquemos la lista original debemos utilizar la notación de sublistas.

>>> lista1 = ["A", "B", "C"]

>>> lista2 = lista1[:] # Hacemos una copia de la lista lista1

>>> lista1, lista2

## (['A', 'B', 'C'] ['A', 'B', 'C'])

>>> del lista1[1] # Eliminamos el elemento ['B'] de la lista lista1 ...

>>> lista1, lista2 # ... y en este caso lista2 sigue conservando el valor original de lista1 (['A', 'C'] ['A', 'B', 'C'])

En el primer caso las variables *lista1* y *lista2* hacen referencia a la misma lista almacenada en la memoria del ordenador. Por eso al eliminar un elemento de *lista1*, también desaparece de *lista2*.

Sin embargo en el segundo caso *lista1* y *lista2* hacen referencia a listas distintas (aunque tengan los mismos valores, están almacenadas en lugares distintos de la memoria del ordenador). Por eso, al eliminar un elemento de *lista1*, no se elimina en *lista2*.

### Métodos

El objeto de tipo *lista* integra una serie de métodos integrados a continuación:

#### append()

Este método agrega un elemento al final de una lista.

**>>>** versiones\_plone = [2.5, 3.6, 4, 5]

**>>> print** versiones\_plone

[2.5, 3.6, 4, 5]

**>>>** versiones\_plone.append(6)

**>>> print** versiones\_plone

[2.5, 3.6, 4, 5, 6]

#### count()

Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** "6 ->", versiones\_plone.count(6)

6 -> 1

**>>> print** "5 ->", versiones\_plone.count(5)

5 -> 1

**>>> print** "2.5 ->", versiones\_plone.count(2.5)

2.5 -> 1

#### extend()

Este método extiende una lista agregando un iterable al final.

**>>>** versiones\_plone = [2.1, 2.5, 3.6]

**>>> print** versiones\_plone

[2.1, 2.5, 3.6]

**>>>** versiones\_plone.extend([4])

**>>> print** versiones\_plone

[2.1, 2.5, 3.6, 4]

**>>>** versiones\_plone.extend(range(5,7))

**>>> print** versiones\_plone

[2.1, 2.5, 3.6, 4, 5, 6]

#### index()

Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la lista.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6, 4]

**>>> print** versiones\_plone.index(4)

3

El método admite como argumento adicional un índice inicial a partir de donde comenzar la búsqueda, opcionalmente también el índice final.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6, 4]

**>>>** versiones\_plone[2]

3.6

**3.9. Tipo listas**

**>>> print** versiones\_plone.index(4, 2)

3

**>>>** versiones\_plone[3]

4

**>>> print** versiones\_plone.index(4, 5)

6

**>>>** versiones\_plone[6]

4

El método devuelve un excepción *ValueError* si el elemento no se encuentra en la lista, o en el entorno definido.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6, 4]

**>>> print** versiones\_plone.index(9) Traceback (most recent call last):

File "<stdin>", line 1, in <module> ValueError: 9 is not in list

#### insert()

Este método inserta el elemento x en la lista, en el índice i.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** versiones\_plone

[2.1, 2.5, 3.6, 4, 5, 6]

**>>>** versiones\_plone.insert(2, 3.7)

**>>> print** versiones\_plone [2.1, 2.5, 3.7, 3.6, 4, 5, 6]

#### pop()

Este método devuelve el último elemento de la lista, y lo borra de la misma.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** versiones\_plone.pop()

6

**>>> print** versiones\_plone [2.1, 2.5, 3.6, 4, 5]

Opcionalmente puede recibir un argumento numérico, que funciona como índice del elemento (por defecto, -1)

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** versiones\_plone.pop(2)

3.6

**>>> print** versiones\_plone

[2.1, 2.5, 4, 5, 6]

#### remove()

Este método recibe como argumento un elemento, y borra su primera aparición en la lista.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** versiones\_plone

[2.1, 2.5, 3.6, 4, 5, 6]

**>>>** versiones\_plone.remove(2.5)

**>>> print** versiones\_plone [2.1, 3.6, 4, 5, 6]

El método devuelve un excepción *ValueError* si el elemento no se encuentra en la lista.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** versiones\_plone

[2.1, 2.5, 3.6, 4, 5, 6]

**>>>** versiones\_plone.remove(7)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: list.remove(x): x not in list

#### reverse()

Este método invierte el orden de los elementos de una lista.

**>>>** versiones\_plone = [2.1, 2.5, 3.6, 4, 5, 6]

**>>> print** versiones\_plone

[2.1, 2.5, 3.6, 4, 5, 6]

**>>>** versiones\_plone.reverse()

**>>> print** versiones\_plone

[6, 5, 4, 3.6, 2.5, 2.1]

#### sort()

Este método ordena los elementos de una lista.

**>>>** versiones\_plone = [4, 2.5, 5, 3.6, 2.1, 6]

**>>> print** versiones\_plone

[4, 2.5, 5, 3.6, 2.1, 6]

**>>>** versiones\_plone.sort()

**>>> print** versiones\_plone [2.1, 2.5, 3.6, 4, 5, 6]

El método sort() admite la opción reverse, por defecto, con valor False. De tener valor True, el ordenamiento se hace en sentido inverso.

**>>>** versiones\_plone.sort(reverse=True)

**>>> print** versiones\_plone [6, 5, 4, 3.6, 2.5, 2.1]

### Convertir a listas

Para convertir a *tipos listas* debe usar la función *list()* está integrada en el intérprete Python.

De la misma forma que str(42) devuelve '42', esto es, la forma cadena del entero 42, las funciones list() y tuple() devuelven las versiones tipo lista y tipo tupla de los valores que les pasemos. Escribe lo siguiente en el shell interactivo, y observa que el valor de retorno es de un tipo de datos distinto al valor pasado:

### 

# Recorrer una lista

* Una forma es recorrer directamente los elementos de la lista, es decir, que la variable de control del bucle tome los valores de la lista que estamos recorriendo: (ver la siguiente animación)

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

* La otra forma es recorrer indirectamente los elementos de la lista, es decir, que la variable de control del bucle tome como valores los índices de la lista que estamos recorriendo (0,1 ,2 , etc.). En este caso, para acceder a los valores de la lista hay que utilizar letras[i]: (ver la siguiente animación)

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

La primera forma es más sencilla, pero sólo permite recorrer la lista de principio a fin y utilizar los valores de la lista.

La segunda forma es más complicada, pero permite más flexibilidad, como muestran los siguientes ejemplos:

* recorrer una lista al revés

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

* modificar los elementos de una lista

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

* eliminar elementos de la lista
  + Para eliminar los elementos de una lista necesitamos recorrer la lista al revés. Si recorremos la lista de principio a fin, al eliminar un valor de la lista, la lista se acorta y cuando intentamos acceder a los últimos valores se produce un error de índice fuera de rango, como muestra el siguiente ejemplo en el que se eliminan los valores de una lista que valen "B":

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

* Eliminar valores de una lista (incorrecto)

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

* La solución es recorrer la lista en orden inverso, de manera que aunque se eliminen elementos y la lista se acorte, los valores que todavía no se han recorrido siguen existiendo en la misma posición que al principio.

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

Saber si un valor está o no en una lista

Para saber si un valor está en una lista se puede utilizar el operador in. La sintaxis sería "elemento in lista" y devuelve un valor lógico: True si el elemento está en la lista, False si el elemento no está en la lista.

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

Por ejemplo, el programa siguiente comprueba si el usuario es una persona autorizada:

Para saber si un valor no está en una lista se pueden utilizar los operadores not in. La sintaxis sería "elemento not in lista" y devuelve un valor lógico: True si el elemento no está en la lista, False si el elemento está en la lista.

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

Por ejemplo, el programa siguiente comprueba si el usuario es una persona autorizada:

[https://www.mclibre.org/consultar/python/lecciones/python-listas.html#Paso12](https://www.mclibre.org/consultar/python/lecciones/python-listas.html%23Paso12)

### Ejemplos

A continuación, se presentan algunos ejemplos de su uso:

Ejemplo de establecer una colección ordenada/arreglos o vectores

lista = [2, "CMS", True, ["Plone", 10]]

**print** lista

type(lista)

Ejemplo de acceder a un elemento especifico de una lista

l2 = lista[1] **print** l2

**Tipo listas**

Ejemplo de acceder a un elemento a una lista anidada

l3 = lista[3][0]

**print** l3

Ejemplo de establecer nuevo valor de un elemento de lista

lista[1] = 4

**print** lista lista[1] = "CMS"

Ejemplo de obtener un rango de elemento especifico

l3 = lista[0:3]

**print** l3

Ejemplos de obtener un rango con saltos de elementos específicos

l4 = lista[0:3:2]

**print** l4

l5 = lista[1::2]

**print** l5

Ejemplo de iterar sobre cualquier secuencia

Usted puede iterar sobre cualquier secuencia (cadenas de caracteres, lista, claves en un diccionario, líneas en un archivo, ...):

*Ejemplo de iterar sobre una cadenas de caracteres*

**>>>** vocales = 'aeiou'

**>>> for** letra **in** 'hermosa':

**... if** letra **in** vocales: **...**

**print** letra, e o a

*Ejemplo de iterar sobre una lista*

Para separar una cadena en frases, los valores pueden separarse con la función integrada split().

**>>>** mensaje = "Hola, como estas tu?"

**>>>** mensaje.split() *# retorna una lista*

['Hola,', 'como', 'estas', 'tu?']

**>>> for** palabra **in** mensaje.split():

**... print** palabra

**...** Hola, como estas tú?

*Ejemplo de iterar sobre dos o más secuencias*

Para iterar sobre dos o más secuencias al mismo tiempo, los valores pueden emparejarse con la función integrada *zip()*

**>>>** preguntas = ['nombre', 'objetivo', 'sistema operativo']

**>>>** respuestas = ['Leonardo', 'aprender Python y Plone', 'Linux']

**>>> for** pregunta, respuesta **in** zip(preguntas, respuestas):

**... print** '¿Cual es tu {0}?, la respuesta es: {1}.'.format(

**...** pregunta, respuesta)

**...**

¿Cual es tu nombre?, la respuesta es: Leonardo.

¿Cual es tu objetivo?, la respuesta es: aprender Python y Plone.

¿Cual es tu sistema operativo?, la respuesta es: Linux.