

# Listas

Las listas en Python, representadas por el tipo *list*, son colecciones ordenadas y **mutables** de elementos que pueden ser de diferentes tipos de datos. Se definen utilizando corchetes [...] y permiten almacenar, acceder, modificar y eliminar elementos de forma flexible.

## Creación de listas

El siguiente código realiza dos operaciones con una lista en Python: imprime el contenido de la lista y muestra el tipo de la lista.

```
lista = [1,2,3,4,5]
print(f"La lista es: {lista}")
print(f"El tipo de la lista es: {type(lista)}")
```

```
La lista es: [1, 2, 3, 4, 5]
El tipo de la lista es: <class 'list'>
```

El siguiente código crea una lista que contiene diferentes tipos de datos y luego imprime el contenido de esa lista.

```
lista = [1, 2.45, "Jose"]
print(f"La lista es: {lista}")
```

```
La lista es: [1, 2.45, 'Jose']
```

El siguiente código define una lista vacía y luego imprime su contenido.

```
# Definir una lista vacía
lista = []
print(lista)
```

```
[]
```

```
# Definir una lista vacía
lista = list()
print(lista)
```

```
[]
```

## Conversión entre listas y cadenas de caracteres

El siguiente código convierte una cadena en una lista de caracteres y luego imprime el resultado.

```
# Cadena a lista
cadena = "Hola mundo"
lista = list(cadena)
print(lista)
```

```
['H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o']
```

El siguiente código convierte una lista de caracteres en una cadena y luego imprime el resultado.

- **Conversión de Lista a Cadena:**

- Se define una lista llamada *lista* que contiene caracteres individuales: ['H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o'].
- La función “*join(lista)*” toma cada elemento de la lista y los une en una sola cadena. El argumento "" especifica que no se debe usar ningún delimitador entre los elementos de la lista.

```
# Lista a cadena
lista = ['H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o']
cadena = "".join(lista)
print(cadena)
```

```
Hola mundo
```

El siguiente convierte una lista de palabras en una cadena con espacios entre ellas y luego imprime el resultado.

- **Conversión de Lista a Cadena con Separación:**

- Se define una lista llamada *lista* que contiene palabras individuales: ["Este", "es", "un", "ejemplo"].
- La función `join(lista)` toma cada elemento de la lista y los une en una sola cadena, utilizando un espacio (" ") como delimitador entre los elementos de la lista.

```
# Lista a cadena con separación
lista = ["Este", "es", "un", "ejemplo"]
cadena = " ".join(lista)
print(cadena)
```

Este es un ejemplo

El siguiente convierte una cadena en una lista de palabras separadas por espacios y luego imprime el resultado.

- **Conversión de Cadena a Lista:**

- Se define una cadena llamada *cadena* con el contenido: "Este es un ejemplo".
- La función `cadena.split()` divide la cadena en una lista de palabras, utilizando los espacios en blanco como delimitadores. La función `split()` sin argumentos divide la cadena en cada espacio en blanco.

```
# Cadena a lista, separando por palabras
cadena = "Este es un ejemplo"
lista = cadena.split()
print(lista)
```

['Este', 'es', 'un', 'ejemplo']

## Obtener elementos de una lista

El siguiente código accede e imprime elementos individuales de una lista de frutas utilizando sus índices.

- **Definición de la Lista:**

- Se define una lista llamada *frutas* con tres elementos: "Manzana", "Plátano", y "Pera".

- **Acceso a Elementos por Índice:**

- La instrucción `print(frutas[0])` imprime el primer elemento de la lista, que es "Manzana".
- La instrucción `print(frutas[1])` imprime el segundo elemento de la lista, que es "Plátano".
- La instrucción `print(frutas[2])` imprime el tercer elemento de la lista, que es "Pera".

```
frutas = ["Manzana", "Plátano", "Pera"]

print(frutas[0])
print(frutas[1])
print(frutas[2])
```

Manzana  
Plátano  
Pera

El siguiente código accede e imprime elementos individuales de una lista de frutas utilizando índices negativos.

- **Definición de la Lista:**

- Se define una lista llamada *frutas* con tres elementos: "Manzana", "Plátano", y "Pera".

- **Acceso a Elementos por Índice Negativo:**

- La instrucción `print(frutas[-1])` imprime el último elemento de la lista, que es "Pera".
- La instrucción `print(frutas[-2])` imprime el penúltimo elemento de la lista, que es "Plátano".
- La instrucción `print(frutas[-3])` imprime el primer elemento de la lista, que es "Manzana".

```
frutas = ["Manzana", "Plátano", "Pera"]

print(frutas[-1])
print(frutas[-2])
print(frutas[-3])
```

Pera  
Plátano  
Manzana

## Trocear una lista mediante Slicing

El siguiente código muestra cómo acceder a diferentes sublistas de una lista de números utilizando el operador de slicing en Python.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Acceso a Sublistas:**

- La instrucción `print(f"Los primeros 5 elementos de mi lista son: {numeros[0:5]}")` imprime los primeros 5 elementos de la lista, que son [0, 1, 2, 3, 4]. El slicing [0:5] empieza en el índice 0 y termina en el índice 4 (excluyendo el índice 5).
- La instrucción `print(f"Los primeros 5 elementos de mi lista son: {numeros[:5]}")` también imprime los primeros 5 elementos de la lista, que son [0, 1, 2, 3, 4]. El slicing [:5] omite el índice de inicio (asumiendo el inicio desde 0) y termina en el índice 4 (excluyendo el índice 5).
- La instrucción `print(f"Los últimos 5 elementos de mi lista son: {numeros[5:10]}")` imprime los últimos 5 elementos de la lista, que son [5, 6, 7, 8, 9]. El slicing [5:10] empieza en el índice 5 y termina en el índice 9 (excluyendo el índice 10).
- La instrucción `print(f"Los últimos 5 elementos de mi lista son: {numeros[5:]}")` también imprime los últimos 5 elementos de la lista, que son [5, 6, 7, 8, 9]. El slicing [5:] empieza en el índice 5 y continúa hasta el final de la lista.

```
numeros = [0,1,2,3,4,5,6,7,8,9]
print(f"Los primeros 5 elementos de mi lista son: {numeros[0:5]}")
print(f"Los primeros 5 elementos de mi lista son: {numeros[:5]}")
print(f"Los últimos 5 elementos de mi lista son: {numeros[5:10]}")
print(f"Los últimos 5 elementos de mi lista son: {numeros[5:]}")
```

```
Los primeros 5 elementos de mi lista son: [0, 1, 2, 3, 4]
Los primeros 5 elementos de mi lista son: [0, 1, 2, 3, 4]
Los últimos 5 elementos de mi lista son: [5, 6, 7, 8, 9]
Los últimos 5 elementos de mi lista son: [5, 6, 7, 8, 9]
```

El siguiente código muestra cómo acceder a diferentes sublistas de una lista de números utilizando índices negativos y el operador de slicing en Python.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Acceso a Sublistas:**

- La instrucción `print(f"Los primeros 5 elementos de mi lista son: {numeros[-10:-5]}")` imprime los primeros 5 elementos de la lista, que son `[0, 1, 2, 3, 4]`. El slicing `[-10:-5]` empieza en el índice -10 (el primer elemento de la lista) y termina en el índice -5 (excluyendo el índice -5), que corresponde a los elementos `[0, 1, 2, 3, 4]`.
- La instrucción `print(f"Los primeros 5 elementos de mi lista son: {numeros[:5]}")` también imprime los primeros 5 elementos de la lista, que son `[0, 1, 2, 3, 4]`. El slicing `[:5]` empieza en el inicio de la lista y termina en el índice 5 (excluyendo el índice 5), que corresponde a los elementos `[0, 1, 2, 3, 4]`.
- La instrucción `print(f"Los últimos 5 elementos de mi lista son: {numeros[-5:]})` imprime los últimos 5 elementos de la lista, que son `[5, 6, 7, 8, 9]`. El slicing `[-5:]` empieza en el índice -5 (el primer de los últimos 5 elementos) y continúa hasta el final de la lista.

```
numeros = [0,1,2,3,4,5,6,7,8,9]
print(f"Los primeros 5 elementos de mi lista son: {numeros[-10:-5]}")
print(f"Los primeros 5 elementos de mi lista son: {numeros[:5]}")
print(f"Los últimos 5 elementos de mi lista son: {numeros[-5:]}")
```

Los primeros 5 elementos de mi lista son: `[0, 1, 2, 3, 4]`

Los primeros 5 elementos de mi lista son: `[0, 1, 2, 3, 4]`

Los últimos 5 elementos de mi lista son: `[5, 6, 7, 8, 9]`

El siguiente código muestra cómo imprimir una lista completa y cómo utilizar el operador de slicing para obtener una copia completa de la lista.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Copia Completa de la Lista:**

- La instrucción `print(numeros[:])` utiliza el operador de slicing `::`, que significa que se está solicitando una copia completa de la lista. El slicing `::` no especifica un inicio, un final o un paso, por lo que devuelve toda la lista original sin modificaciones.

```
numeros = [0,1,2,3,4,5,6,7,8,9]
print(numeros)
print(numeros[:])
```

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

El siguiente código muestra cómo utilizar el operador de slicing para acceder a elementos específicos de una lista con un paso definido.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Slicing con Paso 2:**

- La instrucción `print(numeros[::2])` utiliza el slicing `::2`, que selecciona todos los elementos de la lista con un paso de 2. Esto significa que se imprimen los elementos en las posiciones 0, 2, 4, 6, y 8 de la lista.
- La salida es:

```
[0, 2, 4, 6, 8]
```

- **Slicing con Inicio, Fin y Paso:**

- La instrucción `print(numeros[0:10:2])` utiliza el slicing `[inicio:fin:paso]`. Aquí se especifica el inicio en 0, el fin en 10 (que es el final de la lista), y el paso en 2. Es equivalente a `::2` en este caso.
- La salida es:

```
[0, 2, 4, 6, 8]
```

- **Slicing desde Inicio hasta el Final con Paso 2:**

- La instrucción `print(numeros[0::2])` especifica el inicio en 0, el final no se especifica (lo que significa que va hasta el final de la lista), y el paso en 2. Esto también selecciona todos los elementos con un paso de 2.
- La salida es:

```
[0, 2, 4, 6, 8]
```

- **Repetición del Slicing con Paso 2:**

- La instrucción `print(numeros[::2])` es igual a la primera instrucción, imprimiendo nuevamente todos los elementos con un paso de 2.
- La salida es:

```
[0, 2, 4, 6, 8]
```

```
numeros = [0,1,2,3,4,5,6,7,8,9]
print(numeros[::2])
print(numeros[0:10:2])
print(numeros[0::2])
print(numeros[::2])
```

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

Este código muestra cómo utilizar el operador de slicing para acceder a los elementos de una lista en orden inverso con un paso definido.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Slicing en Orden Inverso con Paso -2:**

- La instrucción `print(numeros[::-2])` utiliza el slicing `[::-2]`, donde:
  - \* El primer valor (vacío) indica que el inicio es desde el final de la lista.
  - \* El segundo valor (vacío) indica que el fin es desde el principio de la lista.
  - \* El paso de -2 indica que se seleccionan los elementos en orden inverso, saltando cada segundo elemento.
- Esto significa que se imprimen los elementos en las posiciones 9, 7, 5, 3, y 1 de la lista en orden inverso.

```
numeros = [0,1,2,3,4,5,6,7,8,9]
print(numeros[::-2])
```

```
[9, 7, 5, 3, 1]
```

## Invertir una lista

El siguiente código muestra cómo invertir una lista utilizando el operador de slicing en Python.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Invertir la Lista:**

- La instrucción `lista_invertida = numeros[::-1]` utiliza el slicing `[::-1]`, donde:
  - \* El primer valor (vacío) indica que el inicio es desde el final de la lista.
  - \* El segundo valor (vacío) indica que el fin es desde el principio de la lista.
  - \* El paso de -1 indica que se seleccionan los elementos en orden inverso.
- Esto invierte la lista original.



```
numeros = [0,1,2,3,4,5,6,7,8,9]
lista_invertida = numeros[::-1]
print(lista_invertida)
```

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

El siguiente código muestra cómo invertir una lista utilizando la función `reversed()` en Python.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Invertir la Lista con `reversed()`:**

- La instrucción `lista_invertida = list(reversed(numeros))` utiliza la función `reversed()` para obtener un iterador que recorre la lista *numeros* en orden inverso.
  - \* La función `reversed()` no modifica la lista original, sino que devuelve un iterador con los elementos en orden inverso.
  - \* La función `list()` convierte este iterador en una nueva lista.

```
numeros = [0,1,2,3,4,5,6,7,8,9]
lista_invertida = list(reversed(numeros))
print(lista_invertida)
```

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

El siguiente código muestra cómo invertir una lista en su lugar utilizando el método `reverse()` en Python.

- **Definición de la Lista:**

- Se define una lista llamada *numeros* que contiene los números del 0 al 9.

- **Invertir la Lista con `reverse()`:**

- La instrucción `numeros.reverse()` utiliza el método `reverse()` de la lista para invertir los elementos en su lugar.
  - \* A diferencia de `reversed()`, que devuelve una nueva lista invertida, `reverse()` modifica la lista original directamente y no devuelve ningún valor.

```
numeros.reverse()
print(numeros)
```

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

## Añadir elementos al final de una lista

El siguiente código muestra cómo añadir un elemento al final de una lista en Python utilizando el método `append()`.

- **Definición de la Lista:**

- Se define una lista llamada *frutas* que contiene tres elementos: “Manzana”, “Plátano” y “Pera”.

- **Añadir un Elemento con `append()`:**

- La instrucción *frutas.append(“Uva”)* utiliza el método `append()` para añadir el elemento “Uva” al final de la lista.
  - \* El método `append()` modifica la lista original directamente, añadiendo el nuevo elemento al final.

```
frutas = ["Manzana", "Plátano", "Pera"]
frutas.append("Uva")
print(frutas)
```

['Manzana', 'Plátano', 'Pera', 'Uva']

## Añadir un elemento en cualquier posición de una lista

El siguiente código muestra cómo insertar un elemento en una posición específica dentro de una lista en Python utilizando el método `insert()`.

- **Definición de la Lista:**

- Se define una lista llamada *frutas* que contiene tres elementos: “Manzana”, “Plátano” y “Pera”.

- **Insertar un Elemento con `insert()`:**

- La instrucción *frutas.insert(1, “Uva”)* utiliza el método `insert()` para añadir el elemento “Uva” en la posición 1 de la lista.

- \* El primer parámetro de `insert()` especifica el índice en el que se debe insertar el nuevo elemento.
- \* El segundo parámetro es el elemento que se quiere añadir.
- \* Los elementos existentes se desplazan a la derecha para hacer espacio para el nuevo elemento.

```
frutas = ["Manzana", "Plátano", "Pera"]
frutas.insert(1, "Uva")
print(frutas)
```

```
['Manzana', 'Uva', 'Plátano', 'Pera']
```

## Crear una lista a partir de una lista vacía

El siguiente código muestra cómo construir una lista de números utilizando un bucle `for` y el método `append()`.

- **Inicialización de la Lista:**
  - Se define una lista vacía llamada *numeros*.
- **Construcción de la Lista con un Bucle for:**
  - El bucle *for i in range(20)* itera sobre una secuencia de números del 0 al 19, generada por `range(20)`.
  - En cada iteración, el valor de `i` se añade a la lista *numeros* utilizando el método `append()`. Esto agrega cada número al final de la lista.

```
numeros = []
for i in range(20):
    numeros.append(i)
print(numeros)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

## Repetir elementos de una lista

El siguiente código muestra cómo duplicar una lista varias veces utilizando el operador de multiplicación `*`.

- **Duplicación de la Lista:**

- Se crea una nueva lista llamada *mas\_frutas* que es el resultado de multiplicar la lista *frutas* por 4. Esto se logra mediante el operador `*`, que repite el contenido de la lista original el número de veces especificado.

```
frutas = ["Manzana", "Plátano", "Pera"]
mas_frutas = frutas * 4
print(mas_frutas)
```

```
['Manzana', 'Plátano', 'Pera', 'Manzana', 'Plátano', 'Pera', 'Manzana', 'Plátano', 'Pera', 'Manzana', 'Plátano', 'Pera']
```

## Combinar listas

El siguiente código muestra cómo combinar dos listas en una sola utilizando el operador de concatenación `+`.

- **Inicialización de Listas:**

- Se definen dos listas:
  - \* *frutas\_rojas*, que contiene “Fresa”, “Arándano”, y “Frambuesa”.
  - \* *frutas\_verdes*, que contiene “Manzana verde”, “Kiwi”, y “Pera”.

- **Concatenación de Listas:**

- Se crea una nueva lista llamada *frutas* que resulta de concatenar *frutas\_rojas* y *frutas\_verdes* utilizando el operador `+`. Esto une los elementos de ambas listas en una sola.

```
# No modifica las listas originales
frutas_rojas = ["Fresa", "Arándano", "Frambuesa"]
frutas_verdes = ["Manzana verde", "Kiwi", "Pera"]
frutas = frutas_rojas + frutas_verdes
print(frutas)
```

```
['Fresa', 'Arándano', 'Frambuesa', 'Manzana verde', 'Kiwi', 'Pera']
```

El siguiente código muestra cómo modificar una lista existente añadiendo los elementos de otra lista utilizando el método `extend()`.

- **Inicialización de Listas:**

- Se definen dos listas:

- \* *frutas\_rojas*, que contiene “Fresa”, “Arándano”, y “Frambuesa”.
    - \* *frutas\_verdes*, que contiene “Manzana verde”, “Kiwi”, y “Pera”.

- **Modificación de la Lista Original:**

- Se usa el método *extend()* para añadir todos los elementos de *frutas\_verdes* al final de *frutas\_rojas*. El método `extend()` modifica la lista original, en este caso *frutas\_rojas*, agregando los elementos de la lista proporcionada como argumento.

```
# Modifica la lista original
frutas_rojas = ["Fresa", "Arándano", "Frambuesa"]
frutas_verdes = ["Manzana verde", "Kiwi", "Pera"]
frutas_rojas.extend(frutas_verdes)
print(frutas_rojas)
```

```
['Fresa', 'Arándano', 'Frambuesa', 'Manzana verde', 'Kiwi', 'Pera']
```

## Modificar una lista

El siguiente código muestra cómo modificar un elemento específico en una lista.

- **Modificación de un Elemento:**

- La instrucción *frutas[0] = “Uva”* cambia el primer elemento de la lista (índice 0) de “Manzana” a “Uva”. En las listas en Python, los índices comienzan en 0, por lo que *frutas[0]* se refiere al primer elemento.

```
frutas = ["Manzana", "Plátano", "Pera"]
frutas[0] = "Uva"
print(frutas)
```

```
['Uva', 'Plátano', 'Pera']
```

El siguiente código muestra cómo reemplazar un rango de elementos en una lista con un nuevo conjunto de elementos.

- **Reemplazo de un Rango de Elementos:**

- La instrucción `frutas[0:2] = ["Uva"]` reemplaza los elementos en los índices 0 hasta 1 (es decir, el primer y el segundo elemento) con el nuevo elemento “Uva”. En Python, el rango de índices `0:2` incluye los elementos en los índices 0 y 1.

```
frutas = ["Manzana", "Plátano", "Pera"]
frutas[0:2] = ["Uva"]
print(frutas)
```

```
['Uva', 'Pera']
```

## Borrar elementos de una lista

El siguiente código muestra cómo eliminar un elemento de una lista utilizando su índice.

- **Eliminación de un Elemento por Índice:**

- La instrucción `del(frutas[1])` elimina el elemento en el índice 1 de la lista. En este caso, “Plátano” es el elemento en el índice 1.

```
# Borrar mediante su índice
frutas = ["Manzana", "Plátano", "Pera"]
del(frutas[1])
print(frutas)
```

```
['Manzana', 'Pera']
```

El siguiente código muestra cómo eliminar un elemento de una lista utilizando su valor.

- **Eliminación de un Elemento por Valor:**

- La instrucción `frutas.remove("Plátano")` elimina la primera ocurrencia del valor “Plátano” en la lista. Si el valor aparece más de una vez, solo se elimina la primera aparición.

```
# Borrar mediante su valor
frutas = ["Manzana", "Plátano", "Plátano", "Pera"]
frutas.remove("Plátano")
print(frutas)
```

```
['Manzana', 'Plátano', 'Pera']
```

El siguiente código muestra cómo eliminar y extraer el último elemento de una lista, utilizando el método *pop*.

- **Eliminación y Extracción del Último Elemento:**

- La instrucción *fruta\_borrada = frutas.pop()* elimina el último elemento de la lista *frutas* y lo asigna a la variable *fruta\_borrada*. El método *pop* sin argumentos elimina y devuelve el último elemento de la lista.

```
# Borrar y extraer el elemento borrado en la última posición de la lista
frutas = ["Manzana", "Plátano", "Pera"]
fruta_borrada = frutas.pop()
print(fruta_borrada)
print(frutas)
```

Pera

['Manzana', 'Plátano']

El siguiente código muestra cómo eliminar y extraer un elemento específico de una lista mediante su índice utilizando el método *pop*.

- **Eliminación y Extracción del Elemento en una Posición Específica:**

- La instrucción *fruta\_borrada = frutas.pop(-1)* elimina el elemento en la última posición de la lista (índice -1) y lo asigna a la variable *fruta\_borrada*. El método *pop* con un argumento de índice elimina y devuelve el elemento en la posición especificada. En este caso, -1 se refiere al último elemento de la lista.

```
# Borrar y extraer el elemento borrado en alguna posición de la lista
frutas = ["Manzana", "Plátano", "Pera"]
fruta_borrada = frutas.pop(-1)
print(fruta_borrada)
print(frutas)
```

Pera

['Manzana', 'Plátano']

El siguiente código muestra cómo eliminar elementos de una lista utilizando el slicing.

- **Eliminación mediante Slicing:**

- La instrucción `frutas[:2] = []` utiliza slicing para seleccionar los elementos desde el inicio hasta, pero sin incluir, el índice 2 (es decir, los dos primeros elementos de la lista). Luego, estos elementos se reemplazan con una lista vacía, lo que efectivamente elimina los elementos seleccionados de la lista original.

```
# Borrar mediante slicing
frutas = ["Manzana", "Plátano", "Pera"]
frutas[:2] = []
print(frutas)
```

```
['Pera']
```

El siguiente código muestra cómo borrar todos los elementos de una lista utilizando el método `clear()`.

- **Borrado de Todos los Elementos:**

- La instrucción `frutas.clear()` utiliza el método `clear()` para eliminar todos los elementos de la lista `frutas`. Después de ejecutar este método, la lista queda vacía.

```
# Borrar toda la lista mediante clear()
frutas = ["Manzana", "Plátano", "Pera"]
frutas.clear()
print(frutas)
```

```
[]
```

El siguiente código muestra cómo reiniciar una lista asignándole una nueva lista vacía.

- **Reinicio de la Lista:**

- La instrucción `frutas = []` asigna una nueva lista vacía a la variable `frutas`. Esto efectivamente borra todos los elementos anteriores y reinicia la lista a su estado vacío.

```
# Borrar de toda la lista reiniciando la lista
frutas = ["Manzana", "Plátano", "Pera"]
frutas = []
print(frutas)
```

```
[]
```



## Encontrar un elemento en una lista

El siguiente código demuestra cómo encontrar el índice de la primera ocurrencia de un valor en una lista y acceder a ese valor utilizando el índice.

- **Encontrar el Índice del Primer Valor:**

- La instrucción `indice = frutas.index("Plátano")` utiliza el método `index()` para encontrar el índice de la primera ocurrencia del valor "Plátano" en la lista `frutas`. En este caso, el índice de la primera aparición de "Plátano" es 1.

```
frutas = ["Manzana", "Plátano", "Pera", "Plátano"]
indice = frutas.index("Plátano")
print(frutas[indice])
```

Plátano

## Pertenencia de un elemento en una lista

El siguiente código verifica si un valor está presente en una lista.

- **Verificación de Existencia:**

- La instrucción `"Uva" in frutas` utiliza el operador `in` para comprobar si el valor "Uva" está presente en la lista `frutas`.

```
frutas = ["Manzana", "Plátano", "Pera", "Plátano"]
"Uva" in frutas
```

False

```
frutas = ["Manzana", "Plátano", "Pera", "Plátano"]
"Uva" not in frutas
```

True

## Número de ocurrencias en una lista

El siguiente código cuenta cuántas veces aparece un valor específico en una lista.

- **Conteo de Elementos:**

- La instrucción `frutas.count("Plátano")` utiliza el método `count()` para contar cuántas veces aparece el valor "Plátano" en la lista `frutas`.

```
frutas = ["Manzana", "Plátano", "Pera", "Plátano"]
conteo_platano = frutas.count("Plátano")
print(conteo_platano)
```

2

## Ordenar una lista

El siguiente código ordena una lista alfabéticamente sin modificar la lista original.

- **Ordenamiento de la Lista:**

- La instrucción `sorted(frutas)` utiliza la función `sorted()` para crear una nueva lista ordenada alfabéticamente, sin modificar la lista original `frutas`.

```
# Sin modificar la lista original, ordenando de la A a la Z
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
frutas_ordenadas = sorted(frutas)
print(frutas_ordenadas)
```

['Manzana', 'Pera', 'Plátano', 'Uva']

El siguiente código ordena una lista alfabéticamente en orden inverso sin modificar la lista original.

- **Ordenamiento Inverso de la Lista:**

- La instrucción `sorted(frutas, reverse=True)` utiliza la función `sorted()` con el parámetro `reverse=True` para crear una nueva lista ordenada alfabéticamente de la Z a la A, sin modificar la lista original `frutas`.

```
# Sin modificar la lista original, ordenando de la Z a la A
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
frutas_ordenadas_invertida = sorted(frutas, reverse = True)
print(frutas_ordenadas_invertida)
```

```
['Uva', 'Plátano', 'Pera', 'Manzana']
```

El siguiente código ordena una lista alfabéticamente en orden ascendente, modificando la lista original.

- **Ordenamiento de la Lista:**

- La instrucción *frutas.sort()* utiliza el método `sort()` para ordenar los elementos de la lista *frutas* en orden alfabético ascendente (de la A a la Z). A diferencia de la función `sorted()`, el método `sort()` modifica la lista original en lugar de crear una nueva.

```
# Modificar la lista original, ordenando de la A la Z
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
frutas.sort()
print(frutas)
```

```
['Manzana', 'Pera', 'Plátano', 'Uva']
```

El siguiente código ordena una lista alfabéticamente en orden descendente (de la Z a la A), modificando la lista original.

- **Ordenamiento de la Lista en Orden Descendente:**

- La instrucción *frutas.sort(reverse = True)* utiliza el método `sort()` con el parámetro `reverse` establecido en `True`. Esto ordena los elementos de la lista *frutas* en orden alfabético descendente (de la Z a la A). Al igual que el método `sort()` sin parámetros, `sort(reverse = True)` modifica la lista original.

```
# Modificar la lista original, ordenando de Z a la A
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
frutas.sort(reverse = True)
print(frutas)
```

```
['Uva', 'Plátano', 'Pera', 'Manzana']
```

## Iterar sobre una lista

El siguiente código itera sobre una lista de frutas y imprime cada elemento en una nueva línea.

- **Bucle For:**

- La instrucción *for fruta in frutas*: establece un bucle **for** que recorre cada elemento de la lista *frutas*. En cada iteración del bucle, la variable *fruta* toma el valor del siguiente elemento de la lista.

```
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
for fruta in frutas:
    print(fruta)
```

```
Uva
Manzana
Plátano
Pera
```

## Iterar sobre una lista usando enumerate

El siguiente código itera sobre una lista de frutas y imprime tanto el índice como el elemento correspondiente en cada iteración.

- **Definición de la Lista:**

- Se define una lista llamada *frutas* que contiene los elementos “Uva”, “Manzana”, “Plátano” y “Pera”.

- **Uso de `enumerate()`:**

- La función *enumerate(frutas)* se utiliza para obtener tanto el índice como el valor de cada elemento en la lista *frutas*. *enumerate* devuelve un objeto iterable de tuplas, donde cada tupla contiene el índice y el valor correspondiente.

- **Bucle For con `enumerate()`:**

- La instrucción *for i, fruta in enumerate(frutas)*: establece un bucle **for** que recorre cada tupla generada por *enumerate*. En cada iteración, *i* toma el índice del elemento, y *fruta* toma el valor del elemento.

- **Impresión del Índice y el Elemento:**

- Dentro del bucle, la instrucción `print(i, fruta)` imprime el índice y el valor del elemento correspondiente. El índice y el valor están separados por una coma en la salida.

```
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
for i, fruta in enumerate(frutas):
    print(i, fruta)
```

```
0 Uva
1 Manzana
2 Plátano
3 Pera
```

## Iterar sobre múltiples listas

El siguiente código itera simultáneamente sobre dos listas de frutas y imprime los elementos correspondientes de cada lista en cada iteración.

- **Definición de las Listas:**

- Se definen dos listas:
  - \* `frutas1`: ["Fresa", "Arándano", "Frambuesa"]
  - \* `frutas2`: ["Manzana", "Kiwi", "Pera"]

- **Uso de `zip()`:**

- La función `zip(frutas1, frutas2)` se utiliza para combinar las dos listas en un solo iterable. `zip` crea pares de elementos de las listas, emparejando el primer elemento de `frutas1` con el primer elemento de `frutas2`, el segundo con el segundo, y así sucesivamente.

- **Bucle For con `zip()`:**

- La instrucción `for fruta1, fruta2 in zip(frutas1, frutas2):` establece un bucle `for` que recorre cada par de elementos generados por `zip`. En cada iteración, `fruta1` toma el valor de un elemento de `frutas1` y `fruta2` toma el valor del elemento correspondiente de `frutas2`.

```
frutas1 = ["Fresa", "Arándano", "Frambuesa"]
frutas2 = ["Manzana", "Kiwi", "Pera"]

for fruta1, fruta2 in zip(frutas1, frutas2):
    print(fruta1, fruta2)
```

Fresa Manzana  
Arándano Kiwi  
Frambuesa Pera

## Copiar una lista

El siguiente código demuestra cómo hacer una copia superficial de una lista y cómo las modificaciones en la lista original no afectan a la copia.

- **Definición de la Lista:**

- Se define la lista *frutas* con los elementos: ["Uva", "Manzana", "Plátano", "Pera"].

- **Copia de la Lista:**

- La instrucción *frutas\_copia = frutas.copy()* crea una copia superficial de la lista *frutas*. Esto significa que *frutas\_copia* es una nueva lista con los mismos elementos que *frutas* en el momento de la copia.

- **Impresión Inicial:**

- *print(frutas)* imprime la lista original.
- *print(frutas\_copia)* imprime la copia de la lista.

- **Modificación de la Lista Original:**

- La instrucción *frutas[0] = "Sandía"* modifica el primer elemento de la lista original *frutas*. El nuevo primer elemento es "Sandía".

- **Impresión Después de la Modificación:**

- *print(frutas)* imprime la lista original después de la modificación.
- *print(frutas\_copia)* imprime la copia de la lista, que sigue siendo igual a la lista original antes de la modificación.

```
frutas = ["Uva", "Manzana", "Plátano", "Pera"]
frutas_copia = frutas.copy()
print(frutas)
print(frutas_copia)
frutas[0] = "Sandía"
print(frutas)
print(frutas_copia)
```

```
['Uva', 'Manzana', 'Plátano', 'Pera']
['Uva', 'Manzana', 'Plátano', 'Pera']
['Sandía', 'Manzana', 'Plátano', 'Pera']
['Uva', 'Manzana', 'Plátano', 'Pera']
```

## Funciones matemáticas con listas

El siguiente código muestra cómo sumar todos los valores de una lista utilizando la función `sum()` en Python.

- **Definición de la Lista:**

- Se define la lista *numeros* con los valores: [2, 4, 5, 1, 8].

- **Cálculo de la Suma:**

- La instrucción *suma = sum(numeros)* utiliza la función `sum()` para calcular la suma de todos los elementos de la lista *numeros*. La función `sum()` toma como argumento una secuencia (en este caso, la lista *numeros*) y devuelve la suma de sus elementos.

```
# Suma de todos los valores de una lista
numeros = [2,4,5,1,8]
suma = sum(numeros)
print(suma)
```

20

El siguiente código muestra cómo encontrar el valor mínimo de una lista utilizando la función `min()` en Python.

- **Cálculo del Valor Mínimo:**

- La instrucción *minimo = min(numeros)* utiliza la función `min()` para encontrar el valor mínimo de todos los elementos de la lista *numeros*. La función `min()` toma como argumento una secuencia (en este caso, la lista *numeros*) y devuelve el valor más pequeño en esa secuencia.

```
# Valor mínimo de los valores de una lista
numeros = [2,4,5,1,8]
minimo = min(numeros)
print(minimo)
```

1

El siguiente código muestra cómo encontrar el valor máximo de una lista utilizando la función `max()` en Python.

- **Cálculo del Valor Mínimo:**

- La instrucción `maximo = max(numeros)` utiliza la función `max()` para encontrar el valor máximo de todos los elementos de la lista `numeros`. La función `max()` toma como argumento una secuencia (en este caso, la lista `numeros`) y devuelve el valor más grande en esa secuencia.

```
# Valor máximo de los valores de una lista
numeros = [2,4,5,1,8]
maximo = max(numeros)
print(maximo)
```

8

El siguiente código muestra cómo encontrar la longitud de una lista utilizando la función `len()` en Python.

- **Cálculo de la Longitud:**

- La instrucción `longitud = len(numeros)` utiliza la función `len()` para obtener la cantidad de elementos en la lista `numeros`. La función `len()` devuelve el número total de elementos en la lista.

```
# Longitud de una lista
numeros = [2,4,5,1,8]
longitud = len(numeros)
print(longitud)
```

5

## Lista de listas

El siguiente código muestra cómo crear una lista de listas y luego imprimirla.

- **Definición de Listas:**

- Se definen tres listas individuales:
  - \* `lista1` con los valores: [1, 2, 3]
  - \* `lista2` con los valores: [4, 5, 6]
  - \* `lista3` con los valores: [7, 8, 9]

- **Creación de una Lista de Listas:**

- La instrucción `lista_de_listas = [lista1, lista2, lista3]` crea una lista que contiene las tres listas previamente definidas como sus elementos.



```

lista1 = [1,2,3]
lista2 = [4,5,6]
lista3 = [7,8,9]
lista_de_listas = [lista1, lista2, lista3]
print(lista_de_listas)

```

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

- **Impresión de Elementos Específicos:**

- `print(lista_de_listas[0])` imprime el primer elemento de `lista_de_listas`, que es `lista1`.
- `print(lista_de_listas[1])` imprime el segundo elemento de `lista_de_listas`, que es `lista2`.
- `print(lista_de_listas[2])` imprime el tercer elemento de `lista_de_listas`, que es `lista3`.

```

print(lista_de_listas[0])
print(lista_de_listas[1])
print(lista_de_listas[2])

```

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

- **Impresión de Elementos Específicos:**

- `print(lista_de_listas[0][0])` imprime el primer elemento del primer sub-lista (`lista1`), que es el número 1.
- `print(lista_de_listas[0][2])` imprime el tercer elemento del primer sub-lista (`lista1`), que es el número 3.
- `print(lista_de_listas[1][0])` imprime el primer elemento del segundo sub-lista (`lista2`), que es el número 4.
- `print(lista_de_listas[2][2])` imprime el tercer elemento del tercer sub-lista (`lista3`), que es el número 9.

```

print(lista_de_listas[0][0])
print(lista_de_listas[0][2])
print(lista_de_listas[1][0])
print(lista_de_listas[2][2])

```

1

3

4

9

- **Impresión de un slice:**

- `print(lista_de_listas[0][0:2])` imprime un slice del primer sub-lista (*lista1*), que incluye los elementos en los índices 0 y 1.
- En este caso, el slice `[0:2]` toma los elementos desde el índice 0 hasta el índice 2 (sin incluir el índice 2).

```
print(lista_de_listas[0][0:2])
```

`[1, 2]`