

3.9. Tipo listas

En Python tiene varios tipos de datos *compuestos* y dentro de las secuencias, están los tipos de *cadenas de caracteres*. Otro tipo muy importante de secuencia son las *listas*.

Entre las *secuencias*, el más versátil, es la *lista*, para definir una, usted debe escribir es entre corchetes, separando sus elementos con comas cada uno.

La lista en Python son variables que almacenan `arrays`, internamente cada posición puede ser un tipo de datos distinto.

```
>>> factura = ["pan", "huevos", 100, 1234]
>>> factura
['pan', 'huevos', 100, 1234]
```

Las listas en Python son:

- **heterogéneas**: pueden estar conformadas por elementos de distintos tipo, incluidos otras listas.
- **mutables**: sus elementos pueden modificarse.

Una lista en Python es una estructura de datos formada por una secuencia ordenada de objetos.

Los elementos de una lista pueden accederse mediante su índice, siendo 0 el índice del primer elemento.


```
>>> factura[0]
'pan'
>>> factura[3]
1234
```

La función `len()` devuelve la longitud de la lista (su cantidad de elementos).

```
>>> len(factura)
4
```

Los índices de una lista inicia entonces de **0** hasta el tamaño de la lista menos uno (`len(factura) - 1`):

```
>>> len(factura) - 1
3
```

Pueden usarse también índices negativos, siendo **-1** el índice del último elemento  v: 3.7 ▼

```
>>> factura[-1]
1234
```

Los índices negativos van entonces de **-1** (último elemento) a `-len(factura)` (primer elemento).

```
>>> factura[-len(factura)]
'pan'
```

A través de los índices, pueden cambiarse los elementos de una lista en el lugar.

```
>>> factura[1] = "carne"
>>> factura
['pan', 'carne', 100, 1234]
```

De esta forma se cambia el valor inicial de un elemento de la lista lo cual hacen una la lista *mutable*

3.9.1. Métodos

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

3.9.1.1. 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]
```

3.9.1.2. 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
```

 v: 3.7 ▼

3.9.1.3. 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]
```

3.9.1.4. 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
>>> 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
```

 v: 3.7 ▼

3.9.1.5. 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]
```

3.9.1.6. 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]
```

3.9.1.7. 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
```

 v: 3.7 ▼

3.9.1.8. 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]
```

3.9.1.9. 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]
```

3.9.2. Convertir a listas

Para convertir a *tipos listas* debe usar la función `list()` la cual [esta integrada](#) en el interprete Python.

Truco

Para más información consulte las funciones integradas para [operaciones de secuencias](#).

3.9.3. Ejemplos

 v: 3.7 ▼

A continuación, se presentan algunos ejemplos de su uso:

Definir una colección ordenada/arreglos o vectores

```
1 lista = [2, "CMS", True, ["Plone", 10]]
2 print(lista, type(lista))
```

Acceder a un elemento específico de una lista

```
1 l2 = lista[1]
2 print(l2)
```

Acceder a un elemento en una lista anidada

```
1 l3 = lista[3][0]
2 print(l3)
```

Definir nuevo valor de un elemento de lista

```
1 lista[1] = 4
2 print(lista)
3 lista[1] = "CMS"
```

Obtener un rango de elemento específico

```
1 # Obtener un rango de elemento específico
2 l3 = lista[0:3]
```

Obtener un rango con saltos de elementos específicos

```
1 # Obtener un rango con saltos de elementos específicos
2 l4 = lista[0:3:2]
```

Iterar sobre cualquier secuencia

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

Iterar sobre una cadenas de caracteres

```
>>> vocales = "aeiou"
>>> for letra in "hermosa":
...     if letra in vocales:
...         print(letra)
```

```
...  
e o a
```

Iterar sobre una lista

 v: 3.7 ▼

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  
tu?
```

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.
```

3.9.4. Ayuda integrada


Usted puede consultar toda la documentación disponible sobre las **listas** desde la [consola interactiva](#) de la siguiente forma:

```
>>> help(list)
```

Importante

Usted puede descargar el código usado en esta sección haciendo clic [aquí](#).

Truco

Para ejecutar el código `tipo_listas.py`, abra una consola de comando, acceda al  **v: 3.7** ▼ donde se encuentra el mismo, y ejecute el siguiente comando:

```
$ python tipo_listas.py
```

Ver también

Consulte la sección de [lecturas suplementarias](#) del entrenamiento para ampliar su conocimiento en esta temática.

¿Cómo puedo ayudar?

¡Mi soporte está aquí para ayudar!

Mi horario de oficina es de lunes a sábado, de 9 AM a 5 PM. [GMT-4 - Caracas, Venezuela](#).

La hora aquí es actualmente 7:35 PM GMT-4.

Mi objetivo es responder a todos los mensajes dentro de un día hábil.

[Contáctenos en la sección de soporte](#)



What do you think?

4 Respuestas



Upvote



Funny



Love



Surprised



Angry

v: 3.7 ▼



Sad

0 Comentarios

1 Acceder ▼

G

Sé el primero en comentar...

INICIAR SESIÓN CON

O REGISTRARSE CON DISQUS



Nombre