Lección 07: Listas

Índice

Anterior

Definición

Hay cuatro estructuras de datos compuestas en Python: *Listas*, *Tuplas*, *Conjuntos* y *Diccionarios*.

Las listas : - Se utilizan para agrupar valores. - Se pueden modificar o cambiar los valores, son mutables. - Se pueden ordenar. - Permite valores duplicados. - Puede contener valores de distintos tipos, es decir cualquier tipo de objeto. - Pueden estar vacías.

> **Nota:** Una lista de caracteres es distinto de una cadena de caracteres. Entre otras cosas, las cadenas de caracteres son inmutables y solo pueden contener caracteres, las listas son mutables y pueden contener cualquier tipo de objeto.

Se pueden crear usando la función list() o con corchetes [].

```
# Creación de listas vacías

mi_lista = [] # Esto es una lista vacía
mi_otra_lista = list() # Esto es otra lista vacía

print(mi_lista)
print(mi_otra_lista)

print(type(mi_lista))
print(type(mi_otra_lista))

# Creación de listas con valores iniciales

mi_lista = [3,4,5,6, 6, 6, 6] # Pueden tener elementos repetidos
mi_otra_lista = ["queso", "plátano", " ", 35, 44, 2.2, None, "uno"] # Puenden contener eleme

print(mi_lista)
print(mi_otra_lista)

print(type(mi_lista))
print(type(mi_otra_lista))
```

Se puede utilizar la función *len()* para saber el número de elementos de una lista.

```
# Número de elementos de una lista
```

```
print("Lista:", mi_lista)
print(f"Número de elementos: {len(mi_lista)}")

print("Lista:", mi_otra_lista)
print(f"Número de elementos: {len(mi_otra_lista)}")
```

Accediendo a los elementos de una lista

- Se puede acceder a los elementos de una lista mediante el índice de cada uno de ellos.
- Al igual que en las cadenas el índice del primer elemento es 0.
- Se pueden utilizar índices negativos.

```
# Accediendo a los elementos de una lista

primer_elemento = mi_lista[0]
ultimo_elemento = mi_lista[-1] # Se pueden utilizar índices negativos

print(f"Lista: {mi_lista}")
print(f"El primer elemento de la lista es {primer_elemento}")
print(f"El último elemento de la lista es {ultimo_elemento}")

tercer_elemento = mi_otra_lista[2]

print(f"Lista: {mi_otra_lista}")
print(f"El tercer elemento de la lista es {tercer_elemento}")
```

- Se pueden desempaquetar los elementos.
- Se pueden cambiar l os elementos de una lista.

```
# Desempaquetando elementos de una lista
print(f"Lista inicial: {mi_otra_lista}")
primero, segundo, tercero, cuarto, quinto, sexto, septimo, octavo = mi_otra_lista
print(f"El tercer elemento de la lista es {tercero}")

# Cambiando elementos de una lista

mi_otra_lista[2]="globo"
print(f"Lista cambiada: {mi_otra_lista}")
primero, segundo, tercero, cuarto, quinto, sexto, septimo, octavo = mi_otra_lista
print(f"El tercer elemento de la lista es {tercero}")
```

Accediendo a partes de una lista - Slicing

- Se puede acceder u obtener un subconjunto de los elementos de una lista mediante slicing. Igual que con las cadenas, utilizando [*inicial*:*final*:*pasos*].

- Por defecto, *inicial* será 0 (el primer elemento), *final* será -1 (el último elemento) y *pasos* será 1.
- Se pueden utilizar índices negativos.
- Acceder a los elementos por slicing no implica cambiar la lista.

```
# Accediendo a partes de una lista - Slicing

print(f"La lista completa es: {mi_otra_lista}")

print(f"Dos primeros elementos: {mi_otra_lista[:2]}")
print(f"Todos los elementos desde el tercero hasta el final: {mi_otra_lista[2:]}")
print(f"Uno de cada dos elementos desde el primero: {mi_otra_lista[::2]}")
print(f"Lista invertida: {mi_otra_lista[::-1]}")

# Se puede cambiar una parte de la lista
mi_otra_lista[:2] = ["uno", "dos"]
print(f"La lista modificada es: {mi_otra_lista}")
```

Comprobando si un elemento está en la lista

- Se puede utilizar el operador *in* para comprobar si un elemento está en la lista.

```
# Comprobando si un elemento está en la lista

globo_en_lista = "globo" in mi_otra_lista
print(globo_en_lista)

none_en_lista = None in mi_otra_lista
print(none_en_lista)

num_en_lista = 35 in mi_otra_lista
print(num_en_lista)

num_en_lista = 2.2 in mi_otra_lista
print(num_en_lista)

cabra_en_lista = "cabra" in mi_otra_lista
print(cabra_en_lista)
```

Concatenar listas

- Se pueden concatenar listas utilizando el operador *+*.

```
# Concatenar listas

mi_lista_completa = mi_lista + mi_otra_lista
```

```
print(f"Mi lista completa es: {mi_lista_completa}")
```

Añadir elementos a la lista

- Para añadir elementos a una lista se puede utilizar el método *append()*

```
# Añadir elementos a una lista

print(f"Mi lista: {mi_lista}")
mi_lista.append("hola")
print(f"Mi lista con elemento añadido: {mi_lista}")

# Una lista puede tener como elementos otras listas

mi_lista.append(["elemento1", "elemento2"])
print(f"Mi lista con elemento añadido: {mi_lista}")
```

Insertar elementos en una lista

- Se pueden insertar elementos en una posición determinada con el método *insert(item)*

```
# Insertar elementos

print(f"Mi lista: {mi_lista}")
mi_lista.insert(1, "segundo_insertado") #Insertando un elemento en segunda posición
print(f"Mi lista con elemento añadido: {mi_lista}")
```

Eliminando elementos de una lista

Hay varios métodos para eliminar elementos de una lista: - *.remove(item)*: Elimina el elemento *item*. Si hay varios elementos iguales, elimina el primero de ellos. - *.pop(index)*: Elimina el elemento en la posición *index* o el útimo elemento si no se especifica. Retorna el elemento eliminado. - *.clear()*: Elimina todos los elementos de una lista. - *del*: Elimina una lista, o cualquier otro tipo de objeto.

```
# Eliminar elementos de una lista

print(f"Mi lista: {mi_lista_completa}")
mi_lista_completa.remove("dos")
print(f"Mi lista: {mi_lista_completa}")
mi_lista_completa.remove("uno") # Si hay varios elementos iguales, elimina el primero de el
print(f"Mi lista: {mi_lista_completa}")

mi_lista_completa.pop()
print(f"Mi lista: {mi_lista_completa}")
```

```
elemento = mi_lista_completa.pop(7)
print(f"Mi lista: {mi_lista_completa}. Se ha eliminado el elemento: {elemento}")

mi_lista_completa.clear()
print(f"Mi lista: {mi_lista_completa}")

# Eliminando una lista
del mi_lista_completa
```

Copiando listas

Para obtener una copia independiente de una lista es necesario utilizar el método *copy()*.

> **Nota**: Reasignar una lista a otra lista con el operador = no implica una copia de una lista. Por ejemplo, con *lista2=lista1* lista2 será una referencia de lista1. cualquier cambio en lista2, cambiará también lista1.

```
# Copiando listas
lista1 = [1,2,5]
lista2 = lista1 # El operador asignación no realiza una copia.
lista2.append(6)
print(lista1)

lista3 = lista1.copy()
lista3.append("copia independiente")
print(lista3)
print(lista1)
```

Otros métodos interesantes

- *.reverse()*: Invierte el orden de los elementos de una lista, modificando la lista.
- *.sort()*: Ordena los elementos de una lista en orden ascendente, modificando la lista.
- *.index(item)*: Devuelve el índice del elemento *item* en una lista.
- *.count(item)*: Devuelve el número de veces que el elemento *item* está en una lista.

Siguiente