

Estructura de datos tipo LISTAS

¿Qué son las listas?



- Estructura de datos que nos permite almacenar gran cantidad de valores (equivalente a los array en otros lenguajes de programación)
- En Python las listas pueden guardar **diferente tipo de valores** (en otros lenguajes no ocurre esto con los array) Python permite cargar en las listas distintos tipos de valores en cambio en Java no todos del mismo tipo
- Se pueden expandir dinámicamente añadiendo nuevos elementos (otra novedad respecto a los arrays en otros lenguajes)


Estructura de datos tipo LISTAS

Las listas en Python son:

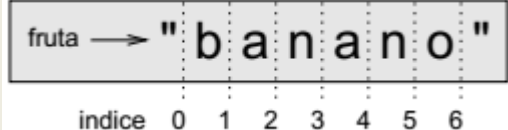
- **heterogéneas:** pueden estar conformadas por elementos de distintos tipo, incluidos otras listas.
- **mutables:** sus elementos pueden modificarse.
- Una lista en Python es una estructura de datos formada por una secuencia ordenada de objetos.
- Los elementos de una lista pueden accederse mediante su índice

Creación de la lista por asignación

```
nombreLista=[elem1, elem2, elem3.....]
```



The diagram shows three vertical green lines pointing from the elements 'elem1', 'elem2', and 'elem3' in the list definition to their respective indices: [0], [1], and [2].



The diagram shows a box containing the string "banana". Above the box, the word "fruta" is followed by an arrow pointing to the first character 'b'. Below the box, the word "indice" is followed by a sequence of indices 0 through 6, each aligned with a character in the string: 0 for 'b', 1 for 'a', 2 for 'n', 3 for 'a', 4 for 'n', 5 for 'o', and 6 for the final quote character.

Para crear una lista por asignación debemos indicar sus elementos encerrados entre corchetes `[]` y separados por coma.

```
lista1=[10, 5, 3]                # lista de enteros
lista2=[1.78, 2.66, 1.55, 89,4]   # lista de valores float
lista3=["lunes", "martes", "miercoles"] # lista de string
lista4=["juan", 45, 1.92]         # lista con elementos de distinto
tipo
```

Si queremos conocer la **cantidad de elementos** de una lista podemos llamar a la función ***len***:

```
lista1=[10, 5, 3]    # lista de enteros
print(len(lista1))    # imprime un 3
```

Índices

Como vimos en las cadenas los subíndices comienzan a numerarse a partir del cero

```
lista=["lunes", "martes", "miercoles", "Jueves"]
```

Si queremos ver el los elementos que tiene una lista

```
print (lista[:])
```

```
print (lista[2])
```

 Si queremos acceder al elemento 2 , en nuestro ejemplo obtendría
`miércoles`

```
print (lista[-2])
```

 Cuando ponemos índices negativos da la vuelta a la lista, es decir, empieza a contar ese índice desde el final, el ultimo es -1, el penúltimo -2 y así sucesivamente.

En nuestro caso sería `miércoles`

Si queremos acceder a una porción de una lista puedo poner, desde y hasta . El desde está incluido el hasta no se incluye por ejemplo:

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

 en este caso haría referencia a los elementos 1 y 2, es decir índice 0 y 1.

```
print (lista[:2])
```

 desde es por defecto es 0

```
print (lista[2:])
```

 hasta es por defecto es el último

Si tenemos esta lista ¿Qué mostrará si ejecutamos las dos últimas líneas de código?

```
lista=["Ana", "Luis", "Martín", "Lucía", "Adrián"]
```

Conocer el índice donde está un elemento

```
print(lista.index("Martín"))
```

nos devuelve el valor del índice donde se encuentra este elemento. Si hay dos valores iguales nos va a devolver el del primero que encuentre.

Comprobar si un elemento si encuentra en la lista

La función **in** nos devuelve True o False si el elemento se encuentra o no en la lista

```
print("Jorge" in lista)
```

En este caso nos devolverá False porque este elemento no está en la lista.

Agregar elementos

Una lista en Python es una estructura mutable (es decir puede ir cambiando durante la ejecución del programa)

El tipo de datos de lista de Python tiene tres métodos para agregar elementos:

append() : agrega un solo elemento a la lista.

insert() : inserta un solo elemento en una posición determinada de la lista.

extend() : agrega elementos de un iterable a la lista.

método append

La primera forma que veremos para que nuestra lista crezca es utilizar el **método append** que tiene la lista y pasar como parámetro el nuevo elemento al final de la lista.

```
lista=[10, 20, 30]  
lista.append(100)
```

Definir una lista vacía y luego solicitar la carga de 5 enteros por teclado y añadirlos a la lista. Imprimir la lista generada.

```
#definimos una lista vacía  
lista=[]  
#disponemos un ciclo de 5 vueltas  
for x in range(5):  
    valor=int(input("Ingrese un valor entero:"))  
    lista.append(valor)  
  
#imprimimos la lista  
print(lista)
```

método insert

Si queremos agregar un elemento en la posición determinada y no al final append no nos sirve, entonces tendremos que utilizar el **método insert**.

insert(posición, elemento)

`lista.insert(2,100)` agregamos un elemento en la posición 2

método extend

Por otro lado podemos querer agregar varios elementos, en este caso vamos a utilizar el **método extend**

`lista.extend([50,75,100])` entre los corchete agregamos los elementos que queremos agregar es como si estuviéramos agregando, concatenando otra lista

```
lista=[50,75]
lista.extend(100)
```

```
Traceback (most recent call last):
  File "C:/Users/Master/AppData/Local/Pro
, in <module>
    lista.extend(100)
TypeError: 'int' object is not iterable
>>> |
```

```
lista=[50,75]
lista.extend([100])
```


Eliminar elementos

método remove

Si queremos eliminar un elemento utilizaremos el **método remove** (elimina por valor)

```
lista.remove(75)
```

método pop

Una segunda opción si queremos eliminar elementos es utilizar el **método pop** (elimina por posición)

```
lista.pop()
```

 si no le indicamos nada por defecto borrará el último

```
lista.pop(1)
```

 borrará el elemento de la posición 1

Función del ()

Otra opción es la **función del** indicando la posición del elemento a eliminar

```
del(lista[1])
```

Concatenar listas

Para unir listas creando una nueva podemos utilizar el operador **+**

```
lista1=[10, 20, 30]  
lista2=["Ana", "Luis", "Martín", "Lucía", "Adrián"]  
lista3=lista1 + lista2
```

Otro operador que podemos utilizar es el *****. Este va a funcionar como **repetidor**

```
lista1=[10, 20, 30] * 3
```

 va a repetir lista1 3 veces

Listas paralelas

Podemos decir que dos listas son paralelas cuando hay una relación entre las componentes de igual subíndice (misma posición) de una lista y otra.

<i>nombres</i>	Juan	Ana	Marcos	Pablo	Laura
<i>edades</i>	12	21	27	14	21

Decimos que la lista nombres es paralela a la lista edades si en la componente 0 de cada lista se almacena información relacionada a una persona (Juan - 12 años)

Es decir hay una relación entre cada componente de las dos listas.

Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesen los datos almacenados en las estructuras de datos.

Listas: ordenamiento de sus elementos

Se debe crear y cargar una lista donde almacenar 5 sueldos. Desplazar el valor mayor de la lista a la última posición.

```
suellos=[]
for x in range(5):
    valor=int(input("Ingrese sueldo:"))
    sueldos.append(valor)
print("Lista sin ordenar")
print(sueldos)
for x in range(4):
    if sueldos[x]>sueldos[x+1]:
        aux=sueldos[x]
        sueldos[x]=sueldos[x+1]
        sueldos[x+1]=aux
print("Lista con el último elemento ordenado")
print(sueldos)
```

El algoritmo consiste en comparar si la primera componente es mayor a la segunda, en caso que la condición sea verdadera, intercambiamos los contenidos de las componentes.

Listas: ordenamiento de sus elementos

Se debe crear y cargar una lista donde almacenar 5 sueldos. Ordenar de menor a mayor la lista.

```
suellos=[]
for x in range(5):
    valor=int(input("Ingrese sueldo:"))
    sueldos.append(valor)
print("Lista sin ordenar")
print(sueldos)
for k in range(4):
    for x in range(4):
        if sueldos[x]>sueldos[x+1]:
            aux=sueldos[x]
            sueldos[x]=sueldos[x+1]
            sueldos[x+1]=aux
print("Lista ordenada")
print(sueldos)
|
```

¿Porque repetimos 4 veces el for externo?

Como sabemos cada vez que se repite en forma completa el for interno queda ordenada una componente de la lista. A primera vista diríamos que deberíamos repetir el for externo la cantidad de componentes de la lista, en este ejemplo la lista sueldos tiene 5 componentes.

Si observamos, cuando quedan dos elementos por ordenar, al ordenar uno de ellos queda el otro automáticamente ordenado (podemos imaginar que si tenemos una lista con 2 elementos no se requiere el for externo, porque este debería repetirse una única vez)

Listas: ordenamiento de sus elementos

Se debe crear y cargar una lista donde almacenar 5 sueldos. Ordenar de menor a mayor la lista.

```
for k in range(4):  
    for x in range(4-k):  
        if sueldos[x]>sueldos[x+1]:  
            aux=sueldos[x]  
            sueldos[x]=sueldos[x+1]  
            sueldos[x+1]=aux
```

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando continuamos comparándolos.

Podemos concluir que la primera vez debemos hacer para este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en uno la cantidad de comparaciones.

Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del anterior es el plantear un for interno con la siguiente estructura:

Es decir restarle el valor del contador del for externo.

Ordenamiento con listas paralelas

Cuando se tienen listas paralelas y se ordenan los elementos de una de ellas hay que tener la precaución de intercambiar los elementos de las listas paralelas.

Confeccionar un programa que permita cargar los nombres de 5 alumnos y sus notas respectivas. Luego ordenar las notas de mayor a menor. Imprimir las notas y los nombres de los alumnos.

```
alumnos=[]
notas=[]
for x in range(5):
    nom=input("Ingrese el nombre del alumno:")
    alumnos.append(nom)
    no=int(input("Ingrese la nota de dicho alumno:"))
    notas.append(no)
for k in range(4):
    for x in range(4-k):
        if notas[x]<notas[x+1]:
            aux1=notas[x]
            notas[x]=notas[x+1]
            notas[x+1]=aux1
            aux2=alumnos[x]
            alumnos[x]=alumnos[x+1]
            alumnos[x+1]=aux2
print("Lista de alumnos y sus notas ordenadas de mayor a menor")
for x in range(5):
    print(alumnos[x],notas[x])
```

Componentes de tipo lista

```
notas=[[4,5], [6,9], [7,3]]
```

Crear una lista por asignación. La lista tiene que tener cuatro elementos. Cada elemento debe ser una lista de 3 enteros. Imprimir sus elementos accediendo de diferentes modos

```
lista=[[1,2,3], [4,5,6], [7,8,9], [10,11,12]]
# imprimimos la lista completa
print(lista)
print("-----")
# imprimimos la primer componente
print(lista[0])
print("-----")
# imprimimos la primer componente de la lista contenida
# en la primer componente de la lista principal
print(lista[0][0])
print("-----")
# imprimimos con un for la lista contenida en la primer componente
for x in range(len(lista[0])):
    print(lista[0][x])
print("-----")
# imprimimos cada elemento entero de cada lista contenida en la lista
```


Otros Métodos

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
```

reverse()

Este método invierte el orden de los elementos de una lista

sort()

Este método ordena los elementos de una lista

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)
```