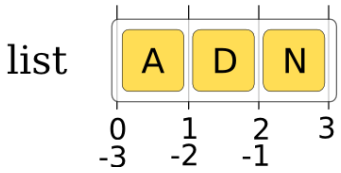
## ¿Por qué necesitamos LISTAS?

Puede suceder que tengas que leer, almacenar, procesar y, finalmente mostrar por pantalla los datos. Entonces las listas en Python tiene una relevancia, es una colección ordenada, algo similar al arreglo unidimensional, pero mucho más versatil.




Python nos permite definir secuencias de valores de cualquier tipo los cuales deben estar encerrados **entre corchetes y separados por comas**. Las listas se almacenan del mismo modo que las **cadenas**: mediante referencias a las secuencia de elementos.

Muchos de los operadores y funciones que trabajan sobre cadenas también lo hacen sobre listas. Hablaremos sobre los métodos de cadena muy pronto, por ahora veremos un breve resumen:

**Las cadenas de Python NO PUEDEN ser modificadas** y a cada carácter se le asigna un número, denominado **índice**, empezando desde 0. Las cadenas de texto soportan una gran cantidad de **MÉTODOS** para transformaciones básicas y búsqueda. Vamos a enfocarnos en métodos de cadenas:

```
>>> pal='inteligencia artificial'
>>> pal.capitalize()
'Inteligencia artificial'
>>> pal.index('g')
6
>>> pal.split(' ')
['inteligencia', 'artificial']
```

*Ejemplo*: Devorador de vocales

```
Ingrese una palabra: politecnica de cordoba
pltcnc d crdb
```

Fig.: Pantalla de salida del devorador de vocales

```
pal=(input('Ingrese una palabra: '))
sin_voc=''
for c in pal:
    if c.upper() not in 'AEIOU':
        sin_voc +=c
print(sin_voc)
```

Fig.: Código Fuente en Python

**Accediendo a los Elementos de la lista:**

Los elementos de la lista están indexados, pueden ser de cualquier tipo de datos y es dinámico.

```
>>> aula1=['Ramiro',4,4.8,True]
>>> aula1
['Ramiro', 4, 4.8, True]
```

El primero elemento tiene índice [0], pero aquí también los índices negativos (comenzar desde el final) son validos y pueden ser muy útiles para las operaciones con listas.

Entonces, dentro de los corchetes se encuentra el índice, que permite acceder a los elementos de la lista ordenada.

```
>>> aula1[2]
4.8
```

```
>>> aula1[-1]
True
```

**Rango de Indices**

Se puede especificar un rango de índices especificando dónde comenzar y donde terminar el rango.

Al especificar un rango, el valor de **retorno será una «nueva lista con los elementos especificados»**.

```
>>> aula1=['Ramiro',2,4.8,True]
>>> aula1[0:]
['Ramiro', 2, 4.8, True]
>>> aula1[:3]
['Ramiro', 2, 4.8]
```

En la 2da línea de instrucción, la búsqueda comenzará en el índice 0 y terminará en el último elemento al no especificar el índice.

En la 3era línea de instrucción, al omitir el valor inicial, el rango comenzará en el primer elemento y terminará en el índice 3 (NO INCLUIDO).

```
>>> aula1[0][3]
'i'
```

La siguiente instrucción permitio acceder al elemento de la posición 0 (’Ramiro’) y del mismo nos permitio acceder a su posición 3 (’i’)

## Diferencia entre método y función

A veces hacemos uso de la palabra sin distinción, pero es importante aclarar estos conceptos.

Una función no es más que una porción o bloque de código reutilizable que se encarga de realizar una determinada tarea. Lo estaremos abordando más adelante en los módulos de la Netacad, por lo traemos al presente.


Un método se comporta como una función y se parece a una función, pero difiere en la forma en que actúa y en su estilo de invocación.

**Los métodos son acciones o funciones que puede realizar un objeto** y Python pone a nuestra disposición un conjunto de métodos ya creados. Estos métodos dependen del tipo de objeto con el que estamos trabajando, como por ejemplo listas.

Si no nos sirve ninguno para nuestro objetivo, siempre podemos crearlo nosotros mismos, como ocurre con las funciones.

Para **acceder a los métodos** y propiedades de un objeto, Python utiliza una sintaxis muy simple: el nombre del objeto, seguido de punto y la propiedad o método al cual se desea acceder. Ejemplo:

```
>>> pal='cordoba capital'
>>> pal.title()
'Cordoba Capital'
>>> pal.count('a')
3
```

## Métodos vs funciones

¿Cuál es entonces la diferencia entre métodos y funciones?. La principal diferencia es que **un método es parte de una clase**, es decir, es **parte de la funcionalidad que le damos a un objeto.** Por tanto, siempre va a estar asociado a un objeto. Sin embargo, **las funciones en Python, están definidas por si mismas** y **no pertenecen a ninguna clase.**

*Para Leer:*

- https://empresas.blogthinkbig.com/python-para-todos-metodo-vs-funcion/
- Sección: 3.1.4.7 de la Netacad - INET-CISCO: Funciones y Métodos.

**Métodos de Listas:**

✓ append() e insert(): añade elementos a una lista. Ej.: a.append(4); a; //[1,2,3,4]

```
>>> aula1
['Ramiro', 4, 4.8, True]
>>> aula1.insert(2,'Nov')
>>> aula1
['Ramiro', 4, 'Nov', 4.8, True]
>>> aula1.insert(2,[34,65])
>>> aula1
['Ramiro', 4, [34, 65], 'Nov', 4.8, True]
>>> aula1.append('Jose')
>>> aula1
['Ramiro', 4, [34, 65], 'Nov', 4.8, True, 'Jose']
```

✓ del(): Se puede eliminar elemento de la lista mediante esta función, para ello se tiene que posicionar en el elemento a eliminar, el cual luego, desaparecerá de la lista y la longitud del mismo se reducirá automáticamente.

```
>>> numeros=[10,5,7,2,1]
>>> numeros
[10, 5, 7, 2, 1]
>>> del numeros[1:3]
>>> numeros
[10, 2, 1]
```

La sentencia **del** no produce una copia de la lista sin la celda borrada, sino que modifica directamente la lista sobre la que opera.

- ✓ len(): longitud de la lista. ESta función toma el nombre de la lista **como un argumento y devuelve el número de elementos almacenados actualmente**.
- ✓ index(): devuelve la posición donde se encuentra un elemento.

```
>>> aula1=[23,45,67,45,58,21]
>>> aula1.index(45)
1
>>> len(aula1)
6
```

- ✓ sort(): Ordena los elementos de una lista.
- ✓ reverse(): Invierte la lista.

```
>>> lista=[12,4,120,32,2,54]
>>> lista.sort()
>>> lista
[2, 4, 12, 32, 54, 120]
>>> lista.reverse()
>>> lista
[120, 54, 32, 12, 4, 2]
```

*Para Leer:*

- https://www.w3schools.com/python/python_ref_list.asp
- Sección: 3.1.4.8 de la Netacad - INET-CISCO: Métodos de listas

**Codificando con Listas en Python**

```
from random import randint
S=randint(20,30)
lista=[S]
while(S!=1):
    if(S%2 == 0):
        S=S/2;
    else:
        S=(S*3)+1
    lista.append(int(S))
print(lista)
print("Longitud de la secuencia: ", len(lista))
print("Número mayor de la secuencia: ", max(lista))
```

Resultado por pantalla

```
[23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1]
Longitud de la secuencia:  16
Número mayor de la secuencia:  160
```

**Listas avanzadas mediante**

## «Comprensión de lista»

Observe el siguiente código:

```
fila=[]
for i in range(8):
    fila.append(i+1)
print(fila)
```

Resultado por pantalla:

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

Otra forma de realizarlo, mediante «Comprensión de lista»:

```
fila=[i+1 for i in range(8)]
```

*Para Leer:*

- Sección: 3.1.7.1 de la Netacad - INET-CISCO: Listas Avanzadas