Listas y stings

August 26, 2016

```
In [1]: lenguajes = ["C","C++","Python","Java"]

In [2]: lenguajes

Out[2]: ['C', 'C++', 'Python', 'Java']

In [34]: enteros = [3, 187, 1232, 53, 21398]

In [4]: enteros

Out[4]: [3, 187, 1232, 53, 21398]

In [5]: l = ["String", 4, 4.5, True]

In [7]: type(True)

Out[7]: bool
```

1 ¿Como accedemos a elementos de una lista?

En python, las estructuras de datos tienen indexación en base cero. La pocision contiene el primer elemento, la posicion 1 es el segundo y cero es el primero.

```
In [10]: lenguajes[2] #python es el tercero por eso es 2

Out[10]: 'Python'

In [13]: lenguajes[0] #primer elemento

Out[13]: 'C'

In [14]: lenguajes [-1]  #muestra el ultimo elemento

Out[14]: 'Java'

In [15]: lenguajes[-2] #muestra el penultimo

Out[15]: 'Python'

In [18]: lenguajes[2] = "Python 3" #cambio de elementos en una lista
```

```
In [19]: lenguajes

Out[19]: ['C', 'C++', 'Python 3', 'Java']

In [21]: lenguajes [1 : 3] #muestra todo los elementos desde el segundo hasta el nu

Out[21]: ['C++', 'Python 3']

In [25]: lenguajes [0:4]

Out[25]: ['C', 'C++', 'Python 3', 'Java']
```

2 muestra todo los elementos desde el primero hasta el 4 (donde es desde 1 hasta 5 pero como 5 no esta pero no muestra no es erro)

```
In [28]: lenguajes[:4] #no es necesario poner 0 para mostrar todos

Out[28]: ['C', 'C++', 'Python 3', 'Java']

In [27]: lenguajes[:] #muestra todos

Out[27]: ['C', 'C++', 'Python 3', 'Java']

In [29]: lenguajes [0:4:2]

Out[29]: ['C', 'Python 3']
```

Muestra todo los elementos, pero los quiero en dos en dos osea coge el primero, tercero, etc

```
In [30]: lenguajes [0:4:1]

Out[30]: ['C', 'C++', 'Python 3', 'Java']

In [31]: lenguajes [0:4:3]

Out[31]: ['C', 'Java']

In [32]: for x in lenguajes:
             print(x)

C
C++
Python 3
Java


In [38]: for x in enteros:
             print(x**2)
```

```
9
34969
1517824
2809
457874404
```

```
In [39]: print(l)
```

```
['String', 4, 4.5, True]
```

```
In [41]: for asdf in l:
             print("\n" + str(asdf) + " cualquier cosa")
```

```
String cualquier cosa

4 cualquier cosa

4.5 cualquier cosa

True cualquier cosa
```

3 ciclo sobre sub indices de la lista

```
In [46]: a, b = 2, "asdf"  #se puede crear variables en una misma lista
```

```
In [45]: a,b
```

```
Out[45]: (2, 'asdf')
```

```
In [50]: for i, x in enumerate(lenguajes):
             print(i, x)
         #muestra la posicion y el elemento de la lista
         #enumerate saca el indice y elemento de una lista
```

```
0 C
1 C++
2 Python 3
3 Java
```

```
In [54]: len(lenguajes) #mismo que length en R
```

```
Out[54]: 4
```

```
In [55]: for i in range(len(lenguajes)):
             print(i, lenguajes[i])
```

```
0 C
1 C++
2 Python 3
3 Java
```

```
In [56]: #crear 10000 numeros aleatorios entre (0,9)
```

```
In [68]: import random   #importación de libreria o modulo

         N = 10000
         random_numbers = []  #creando una lista vacia
         for i in range(N):
             random_numbers.append(random.randint(0,9))

         #random_randint saca desde 0 hasta 9 saca aleatorios
         #.append es agregar un elemento en una lista
         #random_numbers.append es agregar elementos a la lista random_numbers

         #para usar una libreria o sus funciones es
         #libreria.funcion
```

```
In [62]: print(random_numbers)
```

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

```
In [63]: len(random_numbers)
```

```
Out[63]: 10000
```

```
In [64]: import random   #importación de libreria o modulo

         N = 100
         random_numbers2 = []  #creando una lista vacia
         for i in range(N):
             random_numbers2.append(random.randint(0,9))
             print(random_numbers2)
```

```
[6]
[6, 1]
[6, 1, 3]
[6, 1, 3, 0]
[6, 1, 3, 0, 3]
[6, 1, 3, 0, 3, 9]
[6, 1, 3, 0, 3, 9, 1]
[6, 1, 3, 0, 3, 9, 1, 4]
[6, 1, 3, 0, 3, 9, 1, 4, 2]
[6, 1, 3, 0, 3, 9, 1, 4, 2, 6]
```

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

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

4 imprime la lista en cada iterador del ciclo

4.0.1 ¿ como hallar la frecuencia de cada número del 0 al 9 ?

```
In [67]: count = []

In [71]: for x in range(0,10):
             frecuencia = random_numbers.count(x)
             count.append(frecuencia)
         # numero de veces que aparecen los números desde el  hasta 9 en la lista c
         #de radom_numbers

In [70]: count

Out[70]: [1065, 1027, 1029, 966, 982, 992, 937, 964, 1024, 1014]
```

5 concepto clave de python. Los “tipos de objetos” tienen metodos

5.0.1 (“randin” es un metodo del modulo “random”, “append” es un metodo de las listas, “count” tambien)

```
In [72]: help(list)

Help on class list in module builtins:

class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
```

```
|  __getitem__(...)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __iadd__(self, value, /)
|      Implement self+=value.
|
|  __imul__(self, value, /)
|      Implement self*=value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.n
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(...)
|      L.__reversed__() -- return a reverse iterator over the list
|
|  __rmul__(self, value, /)
|      Return self*value.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
```

```
 |  __sizeof__(...)
 |      L.__sizeof__() -- size of L in memory, in bytes
 |
 |  append(...)
 |      L.append(object) -> None -- append object to end
 |
 |  clear(...)
 |      L.clear() -> None -- remove all items from L
 |
 |  copy(...)
 |      L.copy() -> list -- a shallow copy of L
 |
 |  count(...)
 |      L.count(value) -> integer -- return number of occurrences of value
 |
 |  extend(...)
 |      L.extend(iterable) -> None -- extend list by appending elements from the it
 |
 |  index(...)
 |      L.index(value, [start, [stop]]) -> integer -- return first index of value.
 |      Raises ValueError if the value is not present.
 |
 |  insert(...)
 |      L.insert(index, object) -- insert object before index
 |
 |  pop(...)
 |      L.pop([index]) -> item -- remove and return item at index (default last).
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(...)
 |      L.remove(value) -> None -- remove first occurrence of value.
 |      Raises ValueError if the value is not present.
 |
 |  reverse(...)
 |      L.reverse() -- reverse *IN PLACE*
 |
 |  sort(...)
 |      L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

```
In [77]: help(random)
```

```
Help on module random:
```

```
NAME
    random - Random variable generators.

DESCRIPTION
        integers
        --------
               uniform within range

        sequences
        ---------
               pick random element
               pick random sample
               generate random permutation

        distributions on the real line:
        ------------------------------
               uniform
               triangular
               normal (Gaussian)
               lognormal
               negative exponential
               gamma
               beta
               pareto
               Weibull

        distributions on the circle (angles 0 to 2pi)
        ---------------------------------------------
               circular uniform
               von Mises

    General notes on the underlying Mersenne Twister core generator:

    * The period is 2**19937-1.
    * It is one of the most extensively tested generators in existence.
    * The random() method is implemented in C, executes in a single Python step,
      and is, therefore, threadsafe.

CLASSES
    _random.Random(builtins.object)
        Random
            SystemRandom

    class Random(_random.Random)
     |  Random number generator base class used by bound module functions.
     |
     |  Used to instantiate instances of Random to get generators that don't
```

```
|  share state.
|
|  Class Random can also be subclassed if you want to use a different basic
|  generator of your own devising: in that case, override the following
|  methods:  random(), seed(), getstate(), and setstate().
|  Optionally, implement a getrandbits() method so that randrange()
|  can cover arbitrarily large ranges.
|
|  Method resolution order:
|      Random
|      _random.Random
|      builtins.object
|
|  Methods defined here:
|
|  __getstate__(self)
|      # Issue 17489: Since __reduce__ was defined to fix #759889 this is no
|      # longer called; we leave it here because it has been here since random
|      # rewritten back in 2001 and why risk breaking something.
|
|  __init__(self, x=None)
|      Initialize an instance.
|
|      Optional argument x controls seeding, as for Random.seed().
|
|  __reduce__(self)
|      helper for pickle
|
|  __setstate__(self, state)
|
|  betavariate(self, alpha, beta)
|      Beta distribution.
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|      Returned values range between 0 and 1.
|
|  choice(self, seq)
|      Choose a random element from a non-empty sequence.
|
|  expovariate(self, lambd)
|      Exponential distribution.
|
|      lambd is 1.0 divided by the desired mean.  It should be
|      nonzero.  (The parameter would be called "lambda", but that is
|      a reserved word in Python.)  Returned values range from 0 to
|      positive infinity if lambd is positive, and from negative
|      infinity to 0 if lambd is negative.
|
```

```
|  gammavariate(self, alpha, beta)
|      Gamma distribution.  Not the gamma function!
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|
|      The probability distribution function is:
|
|                  x ** (alpha - 1) * math.exp(-x / beta)
|        pdf(x) =  --------------------------------------
|                    math.gamma(alpha) * beta ** alpha
|
|  gauss(self, mu, sigma)
|      Gaussian distribution.
|
|      mu is the mean, and sigma is the standard deviation.  This is
|      slightly faster than the normalvariate() function.
|
|      Not thread-safe without a lock around calls.
|
|  getstate(self)
|      Return internal state; can be passed to setstate() later.
|
|  lognormvariate(self, mu, sigma)
|      Log normal distribution.
|
|      If you take the natural logarithm of this distribution, you'll get a
|      normal distribution with mean mu and standard deviation sigma.
|      mu can have any value, and sigma must be greater than zero.
|
|  normalvariate(self, mu, sigma)
|      Normal distribution.
|
|      mu is the mean, and sigma is the standard deviation.
|
|  paretovariate(self, alpha)
|      Pareto distribution.  alpha is the shape parameter.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1, _int=<class 'int'>)
|      Choose a random item from range(start, stop[, step]).
|
|      This fixes the problem with randint() which includes the
|      endpoint; in Python this is usually not what you want.
|
|  sample(self, population, k)
|      Chooses k unique random elements from a population sequence or set.
```

```
|
|      Returns a new list containing elements from the population while
|      leaving the original population unchanged.  The resulting list is
|      in selection order so that all sub-slices will also be valid random
|      samples.  This allows raffle winners (the sample) to be partitioned
|      into grand prize and second place winners (the subslices).
|
|      Members of the population need not be hashable or unique.  If the
|      population contains repeats, then each occurrence is a possible
|      selection in the sample.
|
|      To choose a sample in a range of integers, use range as an argument.
|      This is especially fast and space efficient for sampling from a
|      large population:   sample(range(10000000), 60)
|
|  seed(self, a=None, version=2)
|      Initialize internal state from hashable object.
|
|      None or no argument seeds from current time or from an operating
|      system specific randomness source if available.
|
|      For version 2 (the default), all of the bits are used if *a* is a str,
|      bytes, or bytearray.  For version 1, the hash() of *a* is used instead.
|
|      If *a* is an int, all bits are used.
|
|  setstate(self, state)
|      Restore internal state from object returned by getstate().
|
|  shuffle(self, x, random=None)
|      Shuffle list x in place, and return None.
|
|      Optional argument random is a 0-argument function returning a
|      random float in [0.0, 1.0); if it is the default None, the
|      standard random.random will be used.
|
|  triangular(self, low=0.0, high=1.0, mode=None)
|      Triangular distribution.
|
|      Continuous distribution bounded by given lower and upper limits,
|      and having a given mode value in-between.
|
|      http://en.wikipedia.org/wiki/Triangular_distribution
|
|  uniform(self, a, b)
|      Get a random number in the range [a, b) or [a, b] depending on roundinc
|
|  vonmisesvariate(self, mu, kappa)
```

```
 |      Circular data distribution.
 |
 |      mu is the mean angle, expressed in radians between 0 and 2*pi, and
 |      kappa is the concentration parameter, which must be greater than or
 |      equal to zero.  If kappa is equal to zero, this distribution reduces
 |      to a uniform random angle over the range 0 to 2*pi.
 |
 |  weibullvariate(self, alpha, beta)
 |      Weibull distribution.
 |
 |      alpha is the scale parameter and beta is the shape parameter.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _random.Random:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  getrandbits(...)
 |      getrandbits(k) -> x.  Generates an int with k random bits.
 |
 |  random(...)
 |      random() -> x in the interval [0, 1).

class SystemRandom(Random)
 |  Alternate random number generator using sources provided
 |  by the operating system (such as /dev/urandom on Unix or
 |  CryptGenRandom on Windows).
 |
 |   Not available on all systems (see os.urandom() for details).
 |
```

```
|  Method resolution order:
|      SystemRandom
|      Random
|      _random.Random
|      builtins.object
|
|  Methods defined here:
|
|  getrandbits(self, k)
|      getrandbits(k) -> x.  Generates an int with k random bits.
|
|  getstate = _notimplemented(self, *args, **kwds)
|
|  random(self)
|      Get the next random number in the range [0.0, 1.0).
|
|  seed(self, *args, **kwds)
|      Stub method.  Not used for a system random number generator.
|
|  setstate = _notimplemented(self, *args, **kwds)
|
|  ----------------------------------------------------------------------
|  Methods inherited from Random:
|
|  __getstate__(self)
|      # Issue 17489: Since __reduce__ was defined to fix #759889 this is no
|      # longer called; we leave it here because it has been here since random
|      # rewritten back in 2001 and why risk breaking something.
|
|  __init__(self, x=None)
|      Initialize an instance.
|
|      Optional argument x controls seeding, as for Random.seed().
|
|  __reduce__(self)
|      helper for pickle
|
|  __setstate__(self, state)
|
|  betavariate(self, alpha, beta)
|      Beta distribution.
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|      Returned values range between 0 and 1.
|
|  choice(self, seq)
|      Choose a random element from a non-empty sequence.
|
```

```
|  expovariate(self, lambd)
|      Exponential distribution.
|
|      lambd is 1.0 divided by the desired mean.  It should be
|      nonzero.  (The parameter would be called "lambda", but that is
|      a reserved word in Python.)  Returned values range from 0 to
|      positive infinity if lambd is positive, and from negative
|      infinity to 0 if lambd is negative.
|
|  gammavariate(self, alpha, beta)
|      Gamma distribution.  Not the gamma function!
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|
|      The probability distribution function is:
|
|                  x ** (alpha - 1) * math.exp(-x / beta)
|        pdf(x) =  --------------------------------------
|                    math.gamma(alpha) * beta ** alpha
|
|  gauss(self, mu, sigma)
|      Gaussian distribution.
|
|      mu is the mean, and sigma is the standard deviation.  This is
|      slightly faster than the normalvariate() function.
|
|      Not thread-safe without a lock around calls.
|
|  lognormvariate(self, mu, sigma)
|      Log normal distribution.
|
|      If you take the natural logarithm of this distribution, you'll get a
|      normal distribution with mean mu and standard deviation sigma.
|      mu can have any value, and sigma must be greater than zero.
|
|  normalvariate(self, mu, sigma)
|      Normal distribution.
|
|      mu is the mean, and sigma is the standard deviation.
|
|  paretovariate(self, alpha)
|      Pareto distribution.  alpha is the shape parameter.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1, _int=<class 'int'>)
|      Choose a random item from range(start, stop[, step]).
```

```
|
|      This fixes the problem with randint() which includes the
|      endpoint; in Python this is usually not what you want.
|
|  sample(self, population, k)
|      Chooses k unique random elements from a population sequence or set.
|
|      Returns a new list containing elements from the population while
|      leaving the original population unchanged.  The resulting list is
|      in selection order so that all sub-slices will also be valid random
|      samples.  This allows raffle winners (the sample) to be partitioned
|      into grand prize and second place winners (the subslices).
|
|      Members of the population need not be hashable or unique.  If the
|      population contains repeats, then each occurrence is a possible
|      selection in the sample.
|
|      To choose a sample in a range of integers, use range as an argument.
|      This is especially fast and space efficient for sampling from a
|      large population:   sample(range(10000000), 60)
|
|  shuffle(self, x, random=None)
|      Shuffle list x in place, and return None.
|
|      Optional argument random is a 0-argument function returning a
|      random float in [0.0, 1.0); if it is the default None, the
|      standard random.random will be used.
|
|  triangular(self, low=0.0, high=1.0, mode=None)
|      Triangular distribution.
|
|      Continuous distribution bounded by given lower and upper limits,
|      and having a given mode value in-between.
|
|      http://en.wikipedia.org/wiki/Triangular_distribution
|
|  uniform(self, a, b)
|      Get a random number in the range [a, b) or [a, b] depending on roundinc
|
|  vonmisesvariate(self, mu, kappa)
|      Circular data distribution.
|
|      mu is the mean angle, expressed in radians between 0 and 2*pi, and
|      kappa is the concentration parameter, which must be greater than or
|      equal to zero.  If kappa is equal to zero, this distribution reduces
|      to a uniform random angle over the range 0 to 2*pi.
|
|  weibullvariate(self, alpha, beta)
```

```
 |      Weibull distribution.
 |
 |      alpha is the scale parameter and beta is the shape parameter.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Random:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Random:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _random.Random:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.

FUNCTIONS
    betavariate(alpha, beta) method of Random instance
        Beta distribution.

        Conditions on the parameters are alpha > 0 and beta > 0.
        Returned values range between 0 and 1.

    choice(seq) method of Random instance
        Choose a random element from a non-empty sequence.

    expovariate(lambd) method of Random instance
        Exponential distribution.

        lambd is 1.0 divided by the desired mean.  It should be
        nonzero.  (The parameter would be called "lambda", but that is
        a reserved word in Python.)  Returned values range from 0 to
        positive infinity if lambd is positive, and from negative
        infinity to 0 if lambd is negative.

    gammavariate(alpha, beta) method of Random instance
        Gamma distribution.  Not the gamma function!
```

```
    Conditions on the parameters are alpha > 0 and beta > 0.

    The probability distribution function is:

                x ** (alpha - 1) * math.exp(-x / beta)
      pdf(x) =  --------------------------------------
                  math.gamma(alpha) * beta ** alpha

gauss(mu, sigma) method of Random instance
    Gaussian distribution.

    mu is the mean, and sigma is the standard deviation.  This is
    slightly faster than the normalvariate() function.

    Not thread-safe without a lock around calls.

getrandbits(...) method of Random instance
    getrandbits(k) -> x.  Generates an int with k random bits.

getstate() method of Random instance
    Return internal state; can be passed to setstate() later.

lognormvariate(mu, sigma) method of Random instance
    Log normal distribution.

    If you take the natural logarithm of this distribution, you'll get a
    normal distribution with mean mu and standard deviation sigma.
    mu can have any value, and sigma must be greater than zero.

normalvariate(mu, sigma) method of Random instance
    Normal distribution.

    mu is the mean, and sigma is the standard deviation.

paretovariate(alpha) method of Random instance
    Pareto distribution.  alpha is the shape parameter.

randint(a, b) method of Random instance
    Return random integer in range [a, b], including both end points.

random(...) method of Random instance
    random() -> x in the interval [0, 1).

randrange(start, stop=None, step=1, _int=<class 'int'>) method of Random instan
    Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
```

```
    endpoint; in Python this is usually not what you want.

sample(population, k) method of Random instance
    Chooses k unique random elements from a population sequence or set.

    Returns a new list containing elements from the population while
    leaving the original population unchanged.  The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples.  This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.

    To choose a sample in a range of integers, use range as an argument.
    This is especially fast and space efficient for sampling from a
    large population:   sample(range(10000000), 60)

seed(a=None, version=2) method of Random instance
    Initialize internal state from hashable object.

    None or no argument seeds from current time or from an operating
    system specific randomness source if available.

    For version 2 (the default), all of the bits are used if *a* is a str,
    bytes, or bytearray.  For version 1, the hash() of *a* is used instead.

    If *a* is an int, all bits are used.

setstate(state) method of Random instance
    Restore internal state from object returned by getstate().

shuffle(x, random=None) method of Random instance
    Shuffle list x in place, and return None.

    Optional argument random is a 0-argument function returning a
    random float in [0.0, 1.0); if it is the default None, the
    standard random.random will be used.

triangular(low=0.0, high=1.0, mode=None) method of Random instance
    Triangular distribution.

    Continuous distribution bounded by given lower and upper limits,
    and having a given mode value in-between.

    http://en.wikipedia.org/wiki/Triangular_distribution
```

```
    uniform(a, b) method of Random instance
        Get a random number in the range [a, b) or [a, b] depending on rounding.

    vonmisesvariate(mu, kappa) method of Random instance
        Circular data distribution.

        mu is the mean angle, expressed in radians between 0 and 2*pi, and
        kappa is the concentration parameter, which must be greater than or
        equal to zero.  If kappa is equal to zero, this distribution reduces
        to a uniform random angle over the range 0 to 2*pi.

    weibullvariate(alpha, beta) method of Random instance
        Weibull distribution.

        alpha is the scale parameter and beta is the shape parameter.

DATA
    __all__ = ['Random', 'seed', 'random', 'uniform', 'randint', 'choice',...

FILE
    c:\users\usuario\anaconda3\lib\random.py
```

```
In [78]: l1 = [1,2,3,4]

In [79]: l2 = l1

In [80]: l2

Out[80]: [1, 2, 3, 4]

In [81]: l2[0] = 100

In [82]: l2

Out[82]: [100, 2, 3, 4]

In [83]: l1

Out[83]: [100, 2, 3, 4]
```

5.1 lo que hiso es que la variable l2 modifico indirectamente la variable l1

```
In [84]: id(l1)

Out[84]: 663419795528

In [85]: id(l1) == id (l2)
```

```
Out[85]: True

In [86]: l3 = [1,2,3,4]

In [87]: l4 = [1,2,3,4]

In [88]: l3 == l4

Out[88]: True

In [89]: id(l3) == id(l4)

Out[89]: False
```

6 Modificar listas

```
In [91]: lenguajes = ["C","C+","Python","Java"]

In [92]: lenguajes.append("Scheme")

In [93]: lenguajes

Out[93]: ['C', 'C+', 'Python', 'Java', 'Scheme']

In [96]: lenguajes2 = ["Pascal, FORTRAN"]

In [99]: lenguajes3 = lenguajes + lenguajes2

In [101]: lenguajes3  #une dos listas

Out[101]: ['C', 'C+', 'Python', 'Java', 'Scheme', 'Pascal, FORTRAN']

In [104]: lenguajes.extend(lenguajes2) #concatena dos listas

In [105]: lenguajes

Out[105]: ['C', 'C+', 'Python', 'Java', 'Scheme', 'Pascal, FORTRAN', 'Pascal, FORTR

In [108]: lenguajes.insert(2, "Haskell") #en el indice 2 (realidad 3) edita lo que

In [110]: lenguajes

Out[110]: ['C',
           'C+',
           'Haskell',
           'Haskell',
           'Python',
           'Java',
           'Scheme',
           'Pascal, FORTRAN',
           'Pascal, FORTRAN']

In [114]: del lenguajes[2] #elemina el elemento de la posicion 2
          lenguajes

Out[114]: ['C', 'C+', 'Java', 'Scheme', 'Pascal, FORTRAN', 'Pascal, FORTRAN']
```

7 contruir listas

```
In [115]: precios = [100.0, 59.99, 7.00, 15.00]

In [116]: precios_con_descuentos = []

In [118]: for precio in precios:
              nuevo_precio = precio * 0.9
              precios_con_descuentos.append(nuevo_precio)

In [119]: precios_con_descuentos

Out[119]: [90.0, 53.991, 6.3, 13.5]

In [120]: ceros = [0] * 17

In [121]: ceros

Out[121]: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

In [122]: len(ceros)

Out[122]: 17
```

8 Listas de listas

```
In [123]: m = [ [1,2,3], [4,5,6], [7,8,9]]

In [124]: for row in m:
              print(row)

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


In [127]: m[1][2] #muestra elemento tercero de la fila dos

Out[127]: 6

In [128]: m[1][1]=0

In [129]: m

Out[129]: [[1, 2, 3], [4, 0, 6], [7, 8, 9]]
```

9 Algunas funciones y metodos utiles para lista

```
In [130]: len(lenguajes)

Out[130]: 6
```

len es una funcion predefinida, no un metodo de una lista. entonces la sintasxis es siempre len(lista) y no lista.lent()

```
In [133]: max(lenguajes) #ultimo elmento de la lista o maximo si es numero

Out[133]: 'Scheme'

In [135]: lenguajes.count("Java") #cuenta numero de veces que hay java

Out[135]: 1

In [140]: lenguajes.reverse() #cambia de orden de mayor a menor la lista, pero modi

In [141]: lenguajes

Out[141]: ['Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme', 'Java', 'C+', 'C']

In [144]: sorted(lenguajes) #ordena de menor a mayor pero sin alterar lista

Out[144]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']

In [146]: lenguajes.sort()

In [147]: lenguajes

Out[147]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']
```

10 Tuplas

Son listas que se definen y jamas se modifican

```
In [151]: tpl = (1,100,"asdf") #como se crea una tupla

In [149]: tpl

Out[149]: (1, 100, 'asdf')

In [150]: tpl[0] = 8

---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-150-03b431f93bff> in <module>()
----> 1 tpl[0] = 8


TypeError: 'tuple' object does not support item assignment
```

```
In [154]: del tpl[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-154-b2f167b115da> in <module>()
----> 1 del tpl[0]


TypeError: 'tuple' object doesn't support item deletion
```

11 Strings

```
In [157]: nombre = "Juana Matallana"

In [159]: len(nombre) #nombre de caracters

Out[159]: 15

In [161]: nombre[14] #ultimo caracter de nombre

Out[161]: 'a'

In [163]: nombre[14]="e" # no se puede modificar
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-163-feb8e582bdd8> in <module>()
----> 1 nombre[14]="e" # no se puede modificar


TypeError: 'str' object does not support item assignment
```

```
In [166]: nombre[0:-1] #primer caracter hasta el ultimo pero sin mostrarlo

Out[166]: 'Juana Matallan'

In [168]: nombre[0:len(nombre)]

Out[168]: 'Juana Matallana'

In [171]: type(nombre)
```

```
Out[171]: str

In [173]: nombre.upper()  #poner todo en mayusculas

Out[173]: 'JUANA MATALLANA'

In [175]: nombre.lower()

Out[175]: 'juana matallana'

In [176]: nombre.capitalize()

Out[176]: 'Juana matallana'

In [177]: "asf".upper()

Out[177]: 'ASF'

In [178]: nombre[:5]

Out[178]: 'Juana'

In [181]: nombre[0:5]="sofia" #no se puede por ser stinr

        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)

        <ipython-input-181-b9a5acc120af> in <module>()
    ----> 1 nombre[0:5]="sofia" #no se puede por ser stinr


        TypeError: 'str' object does not support item assignment


In [182]: nombre = "sofia matallana"

In [183]: "100" + "32"

Out[183]: '10032'

In [185]: nombre.find("a") #busca la letra a y muestra la ubicaciónm de la primera

Out[185]: 4

In [186]: csv = "Juana,Matallana,Rodriguez"

In [189]: values = csv.split(",") #crea una lista en donde la coma es el separador

In [190]: values
```

```
Out[190]: ['Juana', 'Matallana', 'Rodriguez']

In [191]: sep = "|"

In [193]: sep.join(values) #coge la lista values y la unificatodo y agrega | pero s

Out[193]: 'Juana|Matallana|Rodriguez'

In [194]: "|".join(values)

Out[194]: 'Juana|Matallana|Rodriguez'

In [195]: values[2] = 5

In [196]: values

Out[196]: ['Juana', 'Matallana', 5]

In [199]: "se puede incluir valores y variables en un strin usando llaves, así: {}

Out[199]: 'se puede incluir valores y variables en un strin usando llaves, así: {}

In [200]: "se puede incluir valores y variables en un strin usando llaves, así: {}

Out[200]: 'se puede incluir valores y variables en un strin usando llaves, así: 43

In [201]: PI = 3.141592653589793

In [203]: "Tambien se puede especifica el formato de un valor: {:.2f}.format (PI)"

Out[203]: 'Tambien se puede especifica el formato de un valor: {:.2f}.format (PI)'

In [205]: "Tambien se puede especifica el formato de un valor: {:.2f}".format (PI)

Out[205]: 'Tambien se puede especifica el formato de un valor: 3.14'

In [206]: "Tambien se puede especifica el formato de un valor: {:.4f}".format (PI)

Out[206]: 'Tambien se puede especifica el formato de un valor: 3.1416'

In [207]: "Tambien se puede especifica el formato de un valor: {:.10f}".format (PI)

Out[207]: 'Tambien se puede especifica el formato de un valor: 3.1415926536'

In [208]: "4"+"5"

Out[208]: '45'

In [209]: nombre

Out[209]: 'sofia matallana'

In [210]: lenguajes
```

```
Out[210]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']

In [211]: s=lenguajes

In [212]: s[0:-1]

Out[212]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN']

In [213]: s[-1]

Out[213]: 'Scheme'

In [214]: s[:len(s)-1]

Out[214]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN']

In [215]: s[:]

Out[215]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']

In [216]: s[0:len(s)]

Out[216]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']

In [217]: lenguajes

Out[217]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']

In [218]: s

Out[218]: ['C', 'C+', 'Java', 'Pascal, FORTRAN', 'Pascal, FORTRAN', 'Scheme']

In [ ]:
```