Listas

Las **Listas** y Tuplas pueden contener múltiples valores, lo cual hace más fácil escribir programas que manejen grandes cantidades de datos. Y como las **Listas** en sí pueden contener otras **Listas**, pueden utilizarse para organizar datos en estructuras jerárquicas.

Una **Lista** es un valor que puede contener múltiples valores ordenados secuencialmente. El termino **lista** se refiere a la **lista** misma, no a los valores dentro del valor **lista**. Un valor **lista** se ve de la siguiente manera:

['gato', 'perro', 'ratón', 'elefante']

Al igual que como delimitamos cadenas utilizando comillas ‘’, las **Listas** se delimitan utilizando corchetes []. Los valores dentro de una **lista** se llaman Elementos. Y los Elementos van separados con comas. Por ejemplo:

```
>>> [1, 2, 3]
>>> ['gato', 'perro', 'ratón', 'elefante']
>>> ['hola', 3.1415, True, None, 42]
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
```

A la variable spam le estamos asignando un solo valor: el valor **lista**. Pero la **lista** en sí contiene otros valores. El valor [] es una **lista** vacía, similar a como ‘’ sería una cadena vacía.

Obteniendo valores de una Lista utilizando Índices

Si tenemos una **lista** con ['gato', 'perro', 'ratón', 'elefante'] guardada en una variable spam. La expresión spam[0] se evalúa a ‘gato’, spam[1] se evalúa a ‘perro’ y así. El entero dentro de los corchetes se lo llama **índice**.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> spam[0]
>>> spam[1]
>>> spam[2]
>>> spam[3]
>>> ['gato', 'perro', 'ratón', 'elefante'][3]
>>> 'Hola ' + spam[0]
>>> 'El ' + spam[1] + ' se comió al ' + spam[0] + '.'
```

Las **Listas** también pueden contener otras **Listas**. Para acceder a los valores de esas **Listas** hay que utilizar más de un **índice**.

```
>> spam = [['gato', 'perro'], [10, 20, 30, 40, 50]]
>>> spam[0]
>>> spam[0][1]
>>> spam[1][4]
```

El primer **índice** determina qué **lista** acceder, y el segundo qué elemento de esa **lista** queremos obtener.

Índices Negativos

Lo **índices** empiezan empiezan en 0 pero también pueden ser negativos. Un **índice** -1 se refiere al último elemento de la **lista**, un -2 se refiere al ante último elemento y así.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> spam[-1]
>>> spam[-3]
>>> 'El ' + spam[-1] + ' tiene miedo del ' + spam[-2] + '.'
```

Creando SubListas con Slices/Rangos

Un **índice** nos permite conseguir un valor de la **lista**, mientras que una **slice/rango** nos permite conseguir múltiples valores. Al igual que el **índice** un **rango** se escribe entre corchetes, pero tiene 2 números enteros separados por dos puntos.

- spam[2] es una **lista** con un **índice** (un entero).
- spam[1:4] es una **lista** con un **rango** (dos enteros).

En un **rango**, el primer entero es el **índice** donde comienza el **rango**. El segundo entero es donde termina. El **rango** incluye todos los valores hasta el segundo entero, pero sin incluirlo. Un **rango** se evalúa a una nueva **lista**.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> spam[0:4]
>>> spam[1:3]
>>> spam[0:-1]
```

Si no escribimos el primer entero de un **rango** es lo mismo que escribir 0. Y si no escribimos el segundo entero será como escribir la longitud de la **lista**.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> spam[:2]
>>> spam[1:]
>>> spam[:]
```

Obtener la longitud de una lista con len()

La función len() devuelve el número de elementos que hay dentro de una **lista** cuando le pasamos una en su argumento.

```
>>> spam = ['gato', 'perro', 'ratón']
>>> len(spam)
```

Cambiar los Valores de una Lista utilizando índices

Normalmente cuando asignamos el valor a una variable lo hacemos como spam = 42. Pero con **índices** podemos asignar un valor a una posición en particular de una **lista**.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> spam[1] = 'Joaquin'
>>> spam
>>> spam[2] = spam[1]
>>> spam
>>> spam[-1] = 12345
>>> spam
```

Concatenación y Replicación de Listas

El operador + al que utilizamos para concatenar cadenas y crear una nueva conformada por ambas puede unir 2 **Listas** en una nueva con todos los valores de las mismas. Y el operador * también puede utilizarse con un entero para replicar los valores de una **lista**.

```
>>> [1, 2, 3] + ['A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
```

Remover valores de una Lista utilizando el comando del

El comando del elimina un elemento de la **lista**. Todos los valores que figuraban después se moverán una posición hacia atrás.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> del spam[2]
>>> spam
>>> del spam[2]
>>> spam
```

El comando del también sirve para eliminar variables, pero por lo general no se lo utiliza para ese fin.

Trabajando con Listas

Ejemplo

Ciclos de repetición con Listas

```
for i in [0, 1, 2, 3]:
   print(i)
utiles = ['biromes', 'abrochadoras', 'lanzallamas', 'carpetas']
for i in range(len(utiles)):
   print('El Indice ' + str(i) + ' en útiles vale: ' + utiles[i])
```

Los operadores *in* y not *in*

Podemos determinar si un valor está o no en una **lista** utilizando los operadores **in** y **not in**. Estos operadores devuelven valores Booleanos (True o False).

```
>>> 'hola' in ['aló', 'qué tal?', 'hola', 'buenas']
>>> spam = ['aló', 'qué tal?', 'hola', 'buenas']
>>> 'gato' in spam
>>> 'hola' not in spam
>>> 'gato' not in spam
```

Truco de asignación múltiple

Este truco nos permite asignar múltiples variables con valores de una **lista** en una línea de código.

```
>>> gato = ['gordo', 'naranja', 'ruidoso']
>>> tamaño = gato[0]
>>> color = gato[1]
>>> caracter = gato[2]
```

Utilizando el truco:

```
>>> gato = ['gordo', 'naranja', 'ruidoso']
>>> tamaño, color, caracter = gato
```

La cantidad de variables debe ser igual a la longitud de la **lista** o Python devolverá un valor del tipo ValueError .

Métodos de Listas: index(), append(), insert(), remove() y sort()

Un **método** es lo mismo que una función, pero solo puede llamarse a partir de un tipo de variable en particular. A continuación, veremos **método**s que pueden llamarse cuando trabajamos con variables del tipo **Lista**.

El **método** index() toma un valor, lo busca en la **lista** y devuelve en qué posición lo encontró.

```
>>> spam = ['aló', 'qué tal?', 'hola', 'buenas']
>>> spam.index('hola')
>>> spam.index('qué tal?')
>>> spam.index('hola hola hola')
```

Si encuentra varias veces el valor, solo devolverá la posición del primero que encontró.

Para agregar nuevos valores al final de una **lista** se puede utilizar el **método** append()

```
>>> spam = ['gato', 'perro', 'ratón']
>>> spam.append('alce')
>>> spam
```

El **método** insert() en cambio toma 2 valores: uno para determinar en qué posición de la **lista** se insertará y el otro será el valor a insertar.

```
>>> spam = ['gato', 'perro', 'ratón']
>>> spam.insert(1, 'gallina')
>>> spam
```

El **método** remove() busca un valor en la **lista** y elimina la primera coincidencia que encuentre.

```
>>> spam = ['gato', 'perro', 'ratón', 'elefante']
>>> spam.remove('ratón')
>>> spam
```

Tratar de remover un valor que no existe devolverá un error ValueError.

```
>>> spam.remove('gallina')
```

Listas con números o **Listas** con cadenas pueden ordenarse utilizando el **método** sort().

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
>>> spam = ['hormigas', 'gatos', 'perros', 'tejones', 'elefantes']
>>> spam.sort()
>>> spam
```

También puede utilizarse la keyword reverse=True para ordenarlas al revés.

```
>>> spam.sort(reverse=True)
>>> spam
```

Método copy()

Si tratamos de copiar una lista de la manera normal:

```
>>> lista = [1, 2, 3]
>>> nuevaLista = lista
>>> nuevaLista
```

Nos encontraremos con un problema cuando las modifiquemos:

```
>>> nuevaLista.append(4)
>>> nuevaLista
>>> lista
```

Cualquier cambio que hagamos a una de estas listas también ocurrirá en la otra. Esto se debe a que, en Python, cuando asignamos una lista a una variable no se crea una copia nueva de la lista, sino que se asigna la misma dirección de memoria de la primera lista a la segunda. Para evitar que ocurra esto es necesario utilizar el método copy().

```
>>> lista = [1, 2, 3]
>>> nuevaLista = lista.copy()
>>> nuevaLista
>>> nuevaLista.append(4)
>>> nuevaLista
>>> lista
```