Clase 02 - Listas y Tuplas

# **Listas**

## **Tipos compuestos**

En esta segunda lección vamos a estar hablando de otro tipo de datos, llamado **Lista** o **Array**. Python es un lenguaje muy flexible, el cual implementa multitud de tipos distintos por defecto y eso incluye también tipos compuestos de datos, los cuales se utilizan para agrupar distintos **elementos** o **ítems**, por ejemplo variables, o valores, de una forma **ordenada**, es decir, mantienen el orden en el que se definieron.

## Listas en Python

El más versátil de los tipos compuestos, es la Lista, la cual se describe como una lista de ítems separados por coma y contenido entre dos corchetes.

*Ejemplo*:

**mi\_lista = [-11, 20 , 3, 41]**

**otra\_lista = ['Hola', 'como', 'estas', '?']**

## **Heterogéneas**

En otros lenguajes, las listas tienen como restricción que permite tener un solo tipo de dato. Pero en Python, no tenemos esa restricción. Podemos tener una lista heterogénea que contenga números, variables, strings, o incluso otras listas, u otros tipos de datos que veremos más adelante.

**>>> mi\_lista = [-11, 20 , 3, 41]**

**>>> otra\_lista = ['Hola', 'como', 'estas', '?']**

## **Listas y Strings**

Las listas son muy parecidas a los string, ya que funciona exactamente igual con el índice y el slicing.

*Ejemplo*:

>>> datos = [1, -5, 123 , 34, 'Una cadena', 'Otra cadena', 'Pepito’]

>>> datos[0]

1

>>> datos[-1]

‘Pepito’

>>> datos[-2:]

[‘Otra cadena’, ‘Pepito’]

Otra cosa en la que se parecen las listas a los strings, es que en ambos se puede concatenar, en este caso se **concatenan** listas.

*Ejemplo*:

>>> datos = [1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’]

>>> datos + [0, ‘Otra cadena distinta’, ‘Pepito’, -873758,123]

[1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’, 0, ‘Otra cadena distinta’, ‘Pepito’, -873758,123]

>>> numeros = [1,2,3,4]

>>> numeros + [5,6,7,8]

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

Sin embargo, hay una diferencia entre listas y string, los strings son **inmutables**, pero, las listas son **mutables**, esto significa que si podemos reasignar sus ítems haciendo referencia con el índice.  
  
*Ejemplo*:

>>> pares = [0,2,4,5,8,10]

>>> pares[3] = 6

[0,2,4,6,8,10]

## **Asignación por slicing**

Como vimos, las listas son **mutables** por lo cual, podemos hacer algo que en python se denomina **asignación por slicing**. Esto se logra cuando modificamos cierta parte de la lista, y le damos otro valor.

*Ejemplo*:

letras = ['a', 'b', 'c', 'd', 'e', 'f']

letras[:3] = ['A', 'B', 'C']

'A', 'B', 'C', 'd', 'e', 'f'

Python no exige que sean los mismos valores los que se pueden reasignar.

## **Ejemplo en vivo**

Importante: a continuación veremos ejemplos de listas y tuplas.

Pasos a tener en cuenta

* Borrar valores por slicing
* Funciones de listas
* Append
* Pop
* Count + Index
* Tuplas
* Funciones de tuplas
* Anidación

### **Borrar valores por slicing**

Otra funcionalidad que podemos utilizar gracias a la mutabilidad de las listas y al slicing es borrar los ítems que queramos de una lista.

*Ejemplo*:

>>> letras = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]

>>> letras[:3] = [ ]

[‘d’, ‘e’, ‘f’]

De esta forma le decimos que **los 3 primeros valores son una lista vacía, entonces lo “borra”.**

### **Borrar valores**

¿Y si quisiéramos borrar todos los valores de una lista? En Python podemos hacerlo de una forma muy sencilla, la cual sería **reasignar los ítems de dicha lista a una lista vacía**:

*Ejemplo*:

>>> letras = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]

>>> **letras = [ ]**

**[]**

# **Funciones de listas**

## **¿Qué son?**

En las listas, hay funciones que son muy interesantes e importantes, las funciones integradas. Las listas en Python tienen muchas funciones para utilizar, entre todas ellas vamos a nombrar las más importantes.

👉 Hablaremos de las funciones en Python en una clase en el futuro.

## **APPEND**

La primera función de las listas de la que estaremos hablando es APPEND. Esta función permite agregar un nuevo ítem al final de una lista. La misma se escribe mi\_lista**.append**(ítem\_a\_agregar)  
  
*Ejemplo*:

>>> numeros = [1,2,3,4]

>>> numeros.append(5)

[1,2,3,4,5]

**mi\_lista** sería la lista a la que se le desee agregar el ítem, e **ítem\_a\_agregar** sería el ítem que deseemos agregar a la lista.

No solo acaba ahí. En la función append también podemos **realizar operaciones aritméticas en nuestro ítem.**

*Ejemplo*:

>>> numeros = [1,2,3,4]

>>> numeros.append(3\*2)

[1,2,3,4,6]

>>> numeros.append(3\*\*2+1-12+5\*)

[1,2,3,4,6,13]

## **Longitud de la lista**

¿Se acuerdan cuando hablamos de **len** en string? En listas, se puede usar exactamente la misma función para poder saber la longitud de una lista, es decir, la cantidad de ítems dentro de la misma.

*Ejemplo*:

>>> numeros = [1,2,3,4]

>>> len(numeros)

4

>>> datos = [1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’]

>>> len(datos)

5

# **Pop**

Si append permite agregar un ítem al final de una lista, **pop hace todo lo contrario**, elimina el último ítem de una lista, sin modificar el resto de la lista.

Se escribe como mi\_lista.**pop().**

*Ejemplo*:

>>> numeros = [1,2,3,4]

>>> numeros.pop()

[1,2,3]

>>> datos = [1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’]

>>> datos.pop()

[1, -5, 123,34, ‘Una cadena’]

Si especificamos algo entre el paréntesis al decir mi\_lista.**pop(**algo**)**,

Pop eliminará el primer ítem de ese valor que encuentre.

*Ejemplo*:

>>> numeros = [1,2,1,3,4,1]

>>> numeros.pop(1)

[2,1,3,4,1]

# **Count + Index**

Las listas pueden utilizar la función **count.**

Esta función cuenta el número de veces que nuestro ítem se repite en una lista.

*Ejemplo*:

>>> numeros = [1,2,1,3,1,4,1]

>>>numeros.count(1)

4

## **Index**

Las listas pueden utilizar la función index.

Esta función busca nuestro ítem y nos dice en qué índice se encuentra.

*Ejemplo*:

>>> numeros = [1,2,1,3,1,4,1,5]

>>> numeros.index(5)

7

Si se intenta buscar un valor fuera de la lista, devolverá un error y que no se encontró el valor

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: list.index(x): x not in list

# **Tuplas**

## **Tipos compuestos**

Las tuplas son unas **colecciones de datos parecidas a las listas**, una de las diferencias es que estas son inmutables. Se utilizan para asegurarnos que una colección determinada de datos no se pueda modificar.

Python utiliza tuplas en algunas funciones para devolver **resultados inmutables**, por eso, conviene saber identificarlas. A su vez, dependiendo de lo que queramos hacer, las tuplas pueden ser más rápidas que las listas.

## **Tuplas en Python**

Una tupla se declara muy similar a una lista, con la única diferencia que utiliza paréntesis en lugar de corchetes.

*Ejemplo*:

>>> mi\_tupla = (1,2,3,4)

>>> otra\_tupla = (“Hola”, “como”, “estas”, “?”)

Para declarar una tupla con un único valor hay que declararla de la siguiente forma:  
>>> tupla\_vacia = (2,)  
  
De lo contrario, tupla\_vacia recibirá el valor 2 y no será una tupla, si no, **un int**

## **Heterogéneas**

Al igual que las listas**, las tuplas no tienen la restricción sobre el tipo de datos de los ítems**. Podemos tener una tupla que contenga números, variables, strings, o incluso otras listas, u otros tipos de datos que veremos más adelante.

*Ejemplo*:

>>> mi\_var = ‘Una variable’

>>> datos = (1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’, mi\_var)

## **Tuplas**

Como las listas, las tuplas funcionan exactamente igual con el índice y el slicing.

*Ejemplo*:

>>> datos = (1, -5, 123 , 34, ‘Una cadena’, ‘Otra cadena’, ‘Pepito’)

>>> datos(0)

1

>>> datos(-1)

‘Pepito’

>>> datos( 2: )

(‘Otra cadena’, ‘Pepito’)

## **Concatenación**

Otra cosa en la que se parecen las tuplas a las listas, es que en ambos casos se puede concatenar.

Importante: NO FUNCIONA APPEND 👀 pero puedes agregar cosas

>>> datos = (1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’)

>>> datos + (0, ‘Otra cadena distinta’, ‘Pepito’, -873758,123)

(1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’, 0, ‘Otra cadena distinta’, ‘Pepito’, -873758,123)

>>> numeros = (1,2,3,4)

>>> numeros + (5,6,7,8)

(1,2,3,4,5,6,7,8)

## **Mutabilidad**

Como vimos, hay una diferencia entre listas y tuplas, las listas son mutables (podían reasignar sus ítems), en cambio, las tuplas son inmutables, esto significa que no podemos reasignar sus ítems haciendo referencia con el índice.  
  
*Ejemplo*:

>>>mi\_tupla = (1,2,3,4)

>>> mi\_tupla[2] = 5

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

mi\_tupla[2] = 5

## **Borrar valores en tuplas**

Igual que en las listas, podremos borrar todos los valores de una tupla simplemente indicando que la variable ahora contendrá una tupla vacía:

*Ejemplo*:

>>> letras = (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’)

>>> letras = ()

()

Nota: Esta también es la forma de instanciar una tupla vacía en Python. 😎

# 

# 

# **Funciones de tuplas**

## **Longitud de la tupla**

Al igual que listas, las tuplas pueden utilizar la función **len**.

*Ejemplo*:

>>> numeros = (1,2,3,4)

>>> len(numeros)

4

>>> datos = (1, -5, 123.34, ‘Una cadena’, ‘Otra cadena’)

>>> len(datos)

5

## **Count**

Al igual que las listas, las tuplas pueden utilizar la función count. Esta función cuenta el número de veces que nuestro ítem se repite en una tupla.

*Ejemplo*:

>>> numeros = (1,2,1,3,1,4,1)

>>>numeros.count(1)

4

## **Index**

Al igual que las listas, las tuplas pueden utilizar la función index. Esta función busca nuestro ítem y nos dice en qué índice se encuentra.

*Ejemplo***:**

>>> numeros = (1,2,1,3,1,4,1,5)

>>> numeros.index(5)

7

Si se intenta buscar un valor fuera de la tupla, devolverá un error y que no se encontró el valor.  
Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: tuple.index(x): x not in tuple

# **Anidación**

## **Anidadas**

En Python, una tupla y una lista pueden ser Anidadas esto significa, que pueden contener una lista o una tupla dentro de sí respectivamente.

*Ejemplo***:**

>>> datos = [155, [2,3,4] , ‘Una cadena’ , ‘Otra cadena’ ]

>>> otros\_datos = (2, (5,7,8) , 1 , 8)

>>> lista\_con\_tupla = [1, (2,3,4), ‘Una cadena’, ‘Otra cadena’]

>>> tupla\_con\_lista = (2, [5,7,8], 1, 8)

A continuación mostraremos un ejemplo de cómo acceder a los datos anidados:

*Ejemplo***:**

>>> a = [1,2,3]

>>> b = [4,5,6]

>>> c = [7,8,9]

>>> resultado = [ a ,b , c]

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

>>> resultado[0]

[1,2,3]

>>> resultado[0][1]

2

# **Transformación de colecciones**

En Python, podemos convertir una lista a una tupla haciendo uso de la función tuple() y a su vez, podemos hacer lo mismo, pero a la inversa, es decir, convertir una tupla a lista usando la función list().

*Ejemplo*:

>>> numeros = (1,2,3,4)

>>> list ( numeros )

[1,2,3,4]

>>> datos = [1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’]

>>> tuple(datos)

(1, -5, 123,34, ‘Una cadena’, ‘Otra cadena’)

Transformar una colección a otra