



Clase 2. Python

Listas y Tuplas

***RECUERDA PONER A GRABAR LA
CLASE***





OBJETIVOS DE LA CLASE

- Identificar qué es una Lista y sus funciones.
- Reconocer similitudes y diferencias de listas con string.
- Borrar valores por slicing.
- Identificar qué es una Tupla y sus funciones.
- Reconocer similitudes y diferencias de Tuplas.
- Borrar valores de tuplas.

CRONOGRAMA DEL CURSO

Clase 1



Números y cadenas de caracteres



SCRIPT + SLICING



MI PRIMER PROGRAMA EN PYTHON

Clase 2



Listas y tuplas



DESAFÍO DE LISTAS



DESAFÍO DE TUPLAS



¡PRÁCTICAS INICIALES!

Clase 3



Operadores y expresiones



OPERADORES RELACIONALES



OPERADORES LÓGICOS



EXPRESIONES ANIDADAS

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 = [1,2,3,4]
>>> otra_lista = ["Hola", "como", "estas", "?"]
```



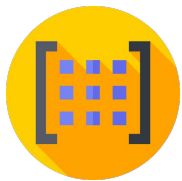
Heterogéneas

En otros lenguajes, las listas tienen como restricción que sólo permite tener un sólo 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.

Ejemplo:

```
>>> mi_var = 'Una variable'  
>>> datos = [1, -5, 123,34, 'Una cadena', 'Otra cadena',  
mi_var]
```

CODER HOUSE



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']
```

CODER HOUSE



Listas y Strings

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]
```



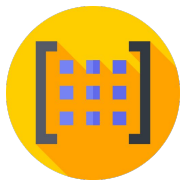
Listas y Strings

Sin embargo, hay una diferencia entre listas y string, los strings son **inmutables**, pero las listas son **mutables**, esto significa que sí 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]
```

CODER HOUSE



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



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”**.

CODER HOUSE

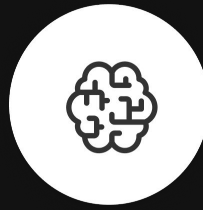


Borrar valores

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

Ejemplo:

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



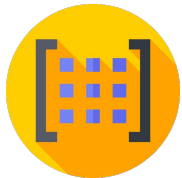
¡PARA PENSAR!

¿Crees que esta forma nos sirve para instanciar una lista vacía de Python?

¿VERDADERO O FALSO?
CONTESTA LA ENCUESTA DE ZOOM



FUNCIONES DE LISTAS



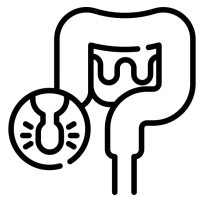
¿Qué son?

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

CODER HOUSE

APPEND



Append

La primer 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)`

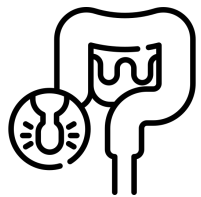
Ejemplo:

```
>>> 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.



Append



No sólo 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



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']
```



Pop



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



Count

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
```

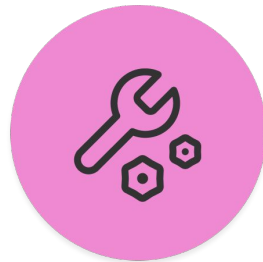
Si se intenta buscar un valor fuera de la lista,
devolverá un error y que no se encontró el valor

Traceback (most recent call last):

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

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

CODER HOUSE



Desafío de Listas

Tiempo estimado: 10 minutos

DESAFÍO DE LISTAS

Desafío
generico



Tiempo estimado: 10 minutos

Dadas dos listas LISTA1 y LISTA2 debes realizar las siguientes tareas:

- Añade a la LISTA1 el int 1234 y luego el string “Hola”
- Añade a la LISTA2 el string “Adios” y luego el int 1234
- Genera una LISTA3 con todos los elementos de la LISTA1 menos el último
- Genera una LISTA4 con todos los elementos de la LISTA2 menos el primero y el último
- Genera una LISTA5 con los elementos de la LISTA4 y de la LISTA3

```
lista1 = [1, 12, 123]
```

```
lista2 = ["Bye", "Ciao", "Agur", "Adieu"]
```



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

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.**

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)
```



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
```

TypeError: 'tuple' object does not support item assignment



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.



CODER HOUSE

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 = [1, [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)
```



Anidadas

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[1]
[4,5,6]
>>> resultado[1][2]
6
```

TRANSFORMACIÓN DE COLECCIONES



Transformar una colección a otra

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)
```

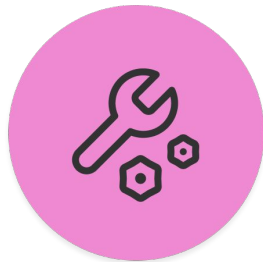
```
>>> list(numeros)
```

```
[1,2,3,4]
```

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

```
>>> tuple(datos)
```

```
(1, -5, 123,34, 'Una cadena', 'Otra cadena')
```



DESAFÍO DE TUPLAS

Tiempo estimado: 10 minutos



DESAFÍO DE TUPLAS

Tiempo estimado: 10 minutos

Nos agruparemos de la 3 estudiantes en Break Out rooms, a cargo del tutor. A partir de una variable llamada tupla debes imprimir por pantalla de forma ordenada, lo siguiente:

- El último ítem de tupla
- El número de ítems de tupla
- La posición donde se encuentra el ítem 87 de tupla
- Una lista con los últimos tres ítems de tupla
- Un ítem que haya en la posición 8 de tupla
- El número de veces que el ítem 7 aparece en tupla

Copia esta tupla:

tupla = (5, 12, 7, 37, 8, 86, 19, 7, -783, 87, 188, 7, 9, 12, 7, 3982)



¡PRÁCTICAS INICIALES!

Breve resumen de la consigna del desafío.

¡PRÁCTICAS INICIALES!

Formato: Documento de Word, Google Docs o PDF con el nombre “Funciones+Apellido”.

Sugerencia: Haz una copia del documento para trabajar.

Desafío
entregable



>> Consigna:

Realizar los ejercicios del siguiente documento disponible en la carpeta:



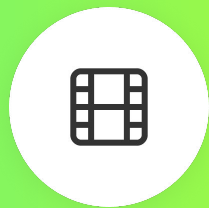
[Desafío entregable 1 \(Clase 2\)](#)

>Aspectos a incluir en el entregable:

Copia del documento con tus respuestas.

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***

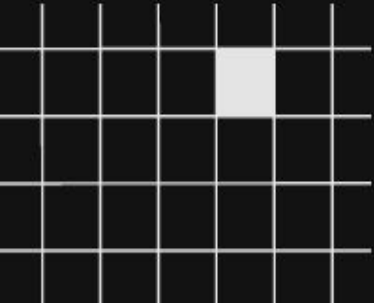


- Artículo: [Tipo Listas](#)
- Artículo: [Funciones Listas](#)
- Artículo: [Tipo Tuplas](#)



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Listas
 - Tuplas
 - Anidación
 - Transformación de colecciones
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE