Esta clase va a ser

- grabada

Clase 02. PYTHON

Listas y tuplas

Temario

01

Números y cadenas de caracteres

- ✓ Números
- ✓ Cadenas de texto
- ✓ Variables
- ✓ String

02

Listas y tuplas

- ✓ Listas
- ✓ Listas (funciones)
- ✓ Tuplas

03

Operadores y expresiones

- ✓ Operadores
- ✓ Expresiones anidadas

Objetivos de la clase

- **Conocer** qué es una Lista
- **Analizar** similitud y diferencias de listas con string
- **Comprender** cómo asignar por slicing
- **Iniciar** los primeros pasos con funciones de listas
- **Definir** y trabajar con Tuplas

Listas

<list>

Listas

En esta segunda clase vamos a estar hablando de otro tipo de datos, llamado **lista**. Python es un lenguaje muy flexible, el cual implementa multitud de tipos distintos por defecto, y eso incluye también tipos compuestos de datos que 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.

CODERHOUSE

Listas

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", "cómo", "estás", "?"]
```

Listas

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 = ["cadena", 1000,  12.34, 0, "otra cadena"]
print(mi_lista)
```

Listas y cadenas

Las listas son muy parecidas a los strings, ya que funcionan exactamente igual respecto al índice y el slicing. Ejemplo:

```
datos = [1, -5, 123, 34, "Una cadena", "Otra cadena", "Pepito"]
print(datos[0])
1
print(datos[-1])
'Pepito'
print(datos[-2:])
['Otra cadena', 'Pepito']
```

Listas y cadenas

Otro punto en el 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]
print(datos)
[1, -5, 123, 34, 'Una cadena', 'Otra cadena', 0,
'Otra cadena distinta', 'Pepito', -873758, 123]
números = [1, 2, 3, 4]
números += [5, 6, 7, 8]
print(números)
[1, 2, 3, 4, 5, 6, 7, 8]
```

Listas y cadenas

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.

```
pares =  [0, 2, 4, 5, 8, 10]
pares[3] = 6
print(pares)
[0, 2, 4, 6, 8, 10]
```

Funciones de listas

CODERHOUSE

¿Qué son las funciones de lista?

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 relevantes.

Append

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(elemento)`
`mi_lista` sería la lista a la que se le desee agregar el elemento.

Ejemplo:

```
números = [1, 2, 3, 4]
numeros.append(5)
print(números)
[1, 2, 3, 4, 5]
```

Len

¿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.

```
números = [1, 2, 3, 4]
longitud = len(números)
print(longitud)
4
datos = [1, -5, 123, 34, "Una cadena", 'Otra cadena']
print(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.

```
números = [1, 2, 3, 4]
números.pop()
print(números)
[1, 2, 3]
```

```
datos = [1, -5, 123, 34, "Una cadena", "Otra cadena"]
datos.pop()
print(datos)
[1, -5, 123, 34, 'Una cadena']
```

Pop

Podemos especificar un **número entero** dentro de los paréntesis de pop. Ese número es el índice del elemento que queremos eliminar. Es decir, mi_lista.**pop(índice)**, la función eliminará el elemento ubicado en dicha posición. Le estamos pasando un argumento. Pop no puede recibir más de un argumento, solamente uno.

```
datos = [1, -5, 123, 34, "Una cadena", "Otra cadena"]
datos.pop(4)
print(datos)
[1, -5, 123, 34, 'Otra cadena']
```

count & index

CODERHOUSE

Count

Las listas pueden utilizar la función **count**.
Esta función cuenta el número de veces que se repite un elemento en una lista.

```
números = [1, 2, 1, 3, 1, 4, 1]
print(números.count(1))
4
```

Index

Las listas pueden utilizar la función **index**.
Esta función busca un elemento y nos dice en qué índice se encuentra.

```
números = [1, 2, 1, 3, 1, 4, 1, 5]
print(números.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 _, in <module>
ValueError: x is not in list
```

Tipos de datos vistos hasta ahora

`int` (números enteros)

`float` (números reales / decimales o flotantes)

`complex` (números complejos)

`str` (strings o cadenas)

`list` (listas)

List: Algunas funciones

`append` (agrega un elemento al final de la lista)

`pop` (elimina un elemento, el último o uno especificado)

`count` (cuenta la cantidad de veces de un elemento)

`index` (devuelve el índice de un elemento)

¿Preguntas?

Break

¡10 minutos y volvemos!

Desafío de Listas

Duración: **15 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Desafío de Listas

Descripción de la actividad:

En esta actividad, podrás poner en práctica lo aprendido. Usaremos Visual Studio Code.

Crea dos listas `lista_1` y `lista_2`, con cualquier elemento que quieras. Realiza los siguientes puntos usando las funciones integradas ya vistas y el método slice [:] Imprime la lista correspondiente luego de cada punto.

- ✔ Añade a la `lista_1` el <int> `456789` y luego el <str> `"Hola Mundo"`
- ✔ Luego añade a la `lista_2` el <str> `"Hola y adiós"`, y luego el <int> `5555`
- ✔ Genera una `lista_3` con todos los elementos de la `lista_1` sin considerar el último elemento [:]
- ✔ Genera una `lista_4` con todos los elementos de la `lista_2` menos el primero y el último elemento [:]
- ✔ Finalmente, genera una `lista_5` con los elementos de la `lista_4` y de la `lista_3`

Tuplas

<tuple>

Tuplas

Las tuplas son **colecciones de datos,** al igual que las listas, con la única diferencia de que las tuplas **son inmutables**.

Se utilizan para asegurarnos que una colección determinada de datos no se pueda modificar.
Python usa 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

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

```
mi_tupla = (1, 2, 3, 4)
otra_tupla = ("Hola", ", ", "¿", "cómo", "estás", "?")
```

Para declarar una tupla con un único valor hay que declararla de la siguiente forma:

```
tupla_valor_único = (2,)
```

Si no ponemos la coma al final, Python interpretará que asignamos el valor de un número entero (int) y no una tupla (tuple).

Tuplas: heterogeneidad

Al igual que las listas, l**as 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.

```
mi_variable = "Una variable"
datos = (1, "cadena", -0.4, mi_variable)
print(datos)
(1, 'cadena', -0.4, 'Una variable')
```

Tuplas: índices y slicing

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

```
datos = (1, "2", [3, "4"], (5, "6"))
print(datos[1])
'2'
más_datos = datos[-1]
print(más_datos)
(5, '6')
print(datos[2:])
([3, '4'], (5, '6'))
```

Tuplas: concatenación

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

Importante: Las tuplas no contienen la función **append()** 👀, pero puedes agregar elementos con la técnica de concatenación:

```
números = (1, 2, 3, 4)
números += (5, 6, 7, 8)
print(números)
(1, 2, 3, 4, 5, 6, 7, 8)
años = (2020, 2021, 2022)
años += (2023,)
print(años)
(2020, 2021, 2022, 2023)
```

Tuplas: 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.

```
mi_tupla = (1, 2, 3, 4)
mi_tupla[2] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment
```

Funciones de tuplas

Len

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

```
números = (1, 2, 3, 4)
print(len(números))
4
datos = (1, -5, 123.34, "Una cadena", "Otra cadena")
longitud_tupla = len(datos)
print(longitud_tupla)
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.

```
números = (1, 2, 1, 3, 1, 4, 1)
cuenta = números.count(1)
print(cuenta)
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.

```
números = (1, 2, 1, 3, 1, 4, 1, 5)
índice = números.index(5)
print(índice)
7
```

Si intentas buscar un valor fuera de la tupla, Python 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

Anidación

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

```
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)
```

Anidación

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

```
a = [1, 2, 3]
b = [4, 5, 6]
c = [7, 8, 9]
resultado = [a, b, c]
print(resultado)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(resultado[0])
[1, 2, 3]
print(resultado[0][1])
2
```

Transformación de colecciones

CODERHOUSE

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()**.

```
números = (1, 2, 3, 4)
números = list(números)
print(números)
[1, 2, 3, 4]
datos = [1, -5, 123.34, "Una cadena", "Otra cadena"]
datos = tuple(datos)
print(datos)
(1, -5, 123.34, 'Una cadena', 'Otra cadena')
```

¿Preguntas?

Desafío de tuplas

A partir de una variable, imprimir por pantalla

Duración: **7 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Desafío de tuplas

Descripción de la actividad.
A partir de una variable llamada `tupla`, 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 para iniciar el ejercicio:

```
tupla = (8, 15, 4, 39, 5, 89, 87, 19, 7, -755, 88, 123, 2, 11, 15, 9, 355)
```

#Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la Guía de Ejercicios Complementarios, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

Actividad extra Nº 2

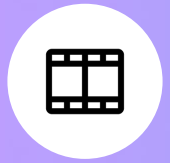
¡Prácticas iniciales!

Consigna

✓ Realiza los ejercicios 1, 2, 3, 4, y 5.

Formato

✓ Puedes completar estas consignas en un Google Docs o un link a su Colabs o descargando La Guía de actividades para poder editarla.



¿Quieres saber más?
Te dejamos material ampliado de la clase

CODERHOUSE

MATERIAL AMPLIADO

Recursos

- ✓ Tipo Listas
- ✓ Artículo: Funciones Listas
- ✓ Artículo: Tipo Tuplas

Disponible en nuestro repositorio.

¿Aún quieres conocer más? Te recomendamos el siguiente material

CODERHOUSE

Resumen
de la clase hoy

- ✓ Listas
- ✓ Tuplas
- ✓ Anidación
- ✓ Transformación de colecciones

Opina y valora esta clase

✨ Te da puntaje para el TOP 10

CODERHOUSE

Muchas gracias.

#DemocratizandoLaEducación