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
- ✓ 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

Cuestionario de tarea

¿Te gustaría comprobar tus conocimientos de la clase anterior?

Te compartimos a través del chat de Zoom el enlace a un breve **cuestionario de Kahoot**

Duración: **5 minutos**

Listas

Tipos compuestos

En esta segunda clase 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', '?']
```

CODERHOUSE

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’]
```

Listas y Strings

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]
[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 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'

CODERHOUSE

Contenido destacado

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

Ejemplo en vivo

A continuación veremos ejemplos de listas y tuplas.

Paso a paso

- Borrar valores por slicing
- Funciones de listas
- Append
- Pop

CODERHOUSE

Paso a paso

- Count+Index
- Tuplas
- Funciones de tuplas
- Anidación

CODERHOUSE

Borrar valores por Slicing

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 **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?

Contesta mediante el chat de Zoom

Funciones de lista

¿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 próximas clases.

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**(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 ítem ubicado en dicha posición.

Ejemplo:

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

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
```

Desafío de Listas

Duración: **10 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Desafío de Listas

Descripción de la actividad.

En esta actividad, podrás poner en práctica todo lo aprendido durante la sesión.

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

- ✔ Añade a la LISTA1 el int 456789 y luego el string “Hola Mundo”
- ✔ Luego añade a la LISTA2 el string “Hola y Adios” y luego el int 5555
- ✔ Genera una LISTA3 con todos los elementos de la LISTA1 sin considerar el último elemento
- ✔ Genera una LISTA4 con todos los elementos de la LISTA2 menos el primero y el último elemento
- ✔ Finalmente, genera una LISTA5 con los elementos de la LISTA4 y de la LISTA3

Break

¡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, 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.

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
```

Borras 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

CODERHOUSE

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)
```

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[0]
[1,2,3]
>>> 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().**

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

A partir de una variable, imprimir por pantalla

Duración: **10 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:
1. El último ítem de tupla
2. El número de ítems de tupla
3. La posición donde se encuentra el ítem 87 de tupla
4. Una lista con los últimos tres ítems de tupla
5. Un ítem que haya en la posición 8 de tupla
6. 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 el Workbook, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

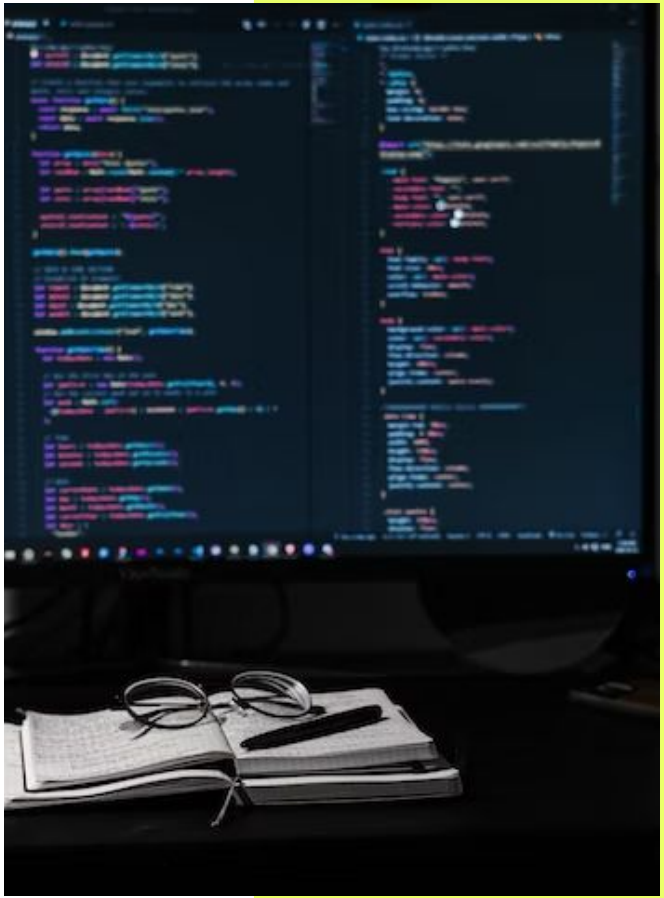
Hora de entrenar

#WorkingTime

Levante la mano quienes quedaron con ganas de seguir practicando.🙋‍♀️🙋🏻‍♂️

Bueno, con el propósito de que sigan robusteciendo sus perfiles profesionales y desarrollando habilidades, les invitamos a revisar el **Workbook del curso**, donde conseguirán un ejercicio para poner en práctica lo visto en la clase de hoy.

Este material es completamente de apoyo por lo que no requerirá ser entregado al profesor/a ni a los tutores o corregido por los mismos. No obstante sí recomendamos que puedan realizar intercambio de las respuestas y logros alcanzados en el workbook, con otros estudiantes y así nutrirse de tips y formas de trabajo.



Fuente de imagen: Kevin Canlas en Unsplash

CODERHOUSE

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 el Workbook para poder editarlo.

Actividad extra Nº 2

¡Prácticas iniciales!

1) Identifica el tipo de dato (int, float, string, list o touple) de los siguientes valores literales.

Dato	Tipo de datos
"Hola Mundo"	
[1, 10, 100]	
-25	
(8, 100, -12)	
1.167	
["Hola", "Mundo"]	
' '	
(1, -5, "Hola!")	

Actividad extra Nº 2

¡Prácticas iniciales!

2) Determina mentalmente (sin programar) el resultado que aparecerá por pantalla a partir de las siguientes variables:

- ✓ a = 10
- ✓ b = -5
- ✓ c = "Hola"
- ✓ d = [1, 2, 3]
- ✓ e= (4,5,6)

Actividad extra Nº 2

¡Prácticas iniciales!

Ejecutar	Resultado
print(a * 5)	
print(a - b)	
print(c + "Mundo")	
print(c * 2)	
print(c[-1])	
print(c[1:])	
print(d + d)	
print(e[1])	
print(e+(7,8,9))	

Actividad extra Nº 2

¡Prácticas iniciales!

3) El siguiente código pretende realizar una media entre 3 números, pero no funciona correctamente. ¿Eres capaz de identificar el problema y solucionarlo?

```
In [1]:
numero_1 = 9
numero_2 = 3
numero_3 = 6

media = numero_1 + numero_2 + numero_3 / 3
print("La nota media es", media)
La nota media es 14.0
```

Actividad extra Nº 2

¡Prácticas iniciales!

4) A partir del ejercicio anterior, desarrolla un programa para calcular la nota final. Para ello vamos a suponer que cada número es una nota y que queremos obtener la nota media. Cada nota tiene un valor porcentual:

- ✓ La primera nota vale un 15% del total
- ✓ La segunda nota vale un 35% del total
- ✓ La tercera nota vale un 50% del total

Ejemplos:

- ✓ nota_1 = 10
- ✓ nota_2 = 7
- ✓ nota_3 = 4

Actividad extra Nº 2

¡Prácticas iniciales!

5) La siguiente matriz (o lista con listas anidadas) debe cumplir una condición: en cada fila el cuarto elemento siempre debe ser el resultado de sumar los tres primeros. ¿Eres capaz de modificar las sumas incorrectas utilizando la técnica del slicing?

🖐 Ayuda: La función llamada sum(lista) devuelve una suma de todos los elementos de la lista

Actividad extra Nº 2

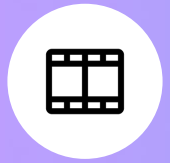
¡Prácticas iniciales!

Partirás de:

```
matriz = [
    [1, 5, 1],
    [2, 1, 2],
    [3, 0, 1],
    [1, 4, 4]
]
```

Debes llegar a:

```
matriz = [
    [1, 5, 1, 7],
    [2, 1, 2, 5],
    [3, 0, 1, 4],
    [1, 4, 4, 9]
]
```



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

CODERHOUSE

MATERIAL AMPLIADO

Recursos

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

Disponible en nuestro repositorio.

¿Aún quieres conocer más? Te recomendamos el siguiente material

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

- ✓ Listas|**Nicolás Perez**
- ✓ Tuplas|**Nicolás Perez**

Disponible en nuestro repositorio.

CODERHOUSE

¿Preguntas?

Resumen
de la clase hoy

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

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación