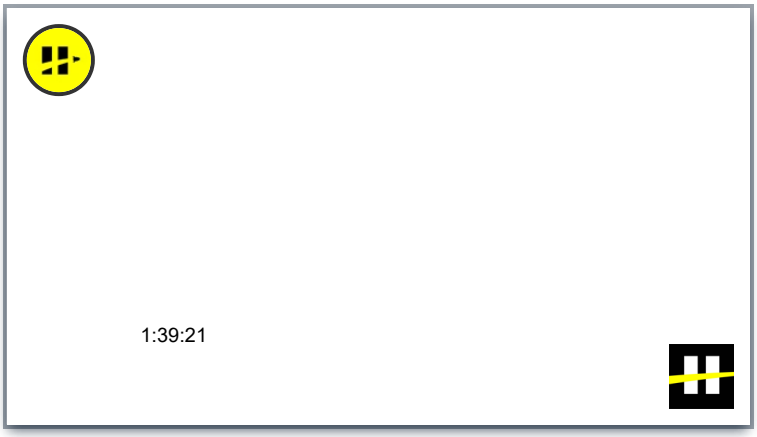
Tiempo de lectura
11 min

Grabación de la Clase 5



Principales Objetivos de Aprendizaje para esta Clase

- **Conocer los conceptos de Iteradores e Iterables**

Iteradores e Iterables

Permiten iterar colecciones de datos que sean iterables. Si tenemos una determinada colección de datos, en este caso una lista con varios valores, y queremos mostrar sus valores uno a uno por pantalla podría resolverse de la siguiente manera con un while.

Dejanos tu feedback! 👍

```
>>>     elemento = lista[i]
>>>     print(elemento)
>>>     i += 1
5
4
9
2
```

Contenido de la clase

Aunque es una forma válida, en Python existe una forma mucho más fácil de iterar una lista. Dicha forma es la siguiente.

```
>>> lista = [5, 4, 9, 2]
>>> for elemento in lista:
>>>     print(elemento)
5
4
9
2
```

Iterables

Una clase iterable es una clase que puede ser iterada. Dentro de Python hay gran cantidad de clases iterables como las listas, strings, diccionarios o archivos. Si tenemos una clase iterable, podemos usarla a la derecha del for de la siguiente manera.

```
for elemento in [clase_iterable]:
```

En el ciclo for, como se puede ver, la variable elemento irá tomando los valores de cada elemento presente en la clase iterable. De esta manera, ya no tenemos que ir accediendo manualmente con [] a cada elemento.
Anteriormente hemos visto un ejemplo iterando una lista, pero también podemos iterar una cadena, ya que es una clase iterable. Al iterar una cadena se nos devuelve cada letra presente en la misma. La sintaxis se asemeja bastante al lenguaje natural, sería algo así como decir “poner en c cada elemento presente en la cadena”.

Dejanos tu feedback! 👍

```
H
o
l
a
```

Contenido de la clase

Además de obtener el elemento dentro del iterable, también es posible obtener el índice en el que está posicionado utilizando la función **enumerate**

```
>>> cadena = "Hola"
>>> for i, c in enumerate(cadena):
>>>   print(i, c)
0 H
1 o
2 l
3 a
```

Para saber si una clase es iterable o no hay dos opciones. La primera sería consultar la documentación oficial de Python. La segunda es ver si la clase u objeto en cuestión hereda de Iterable. Con isinstance() podemos comprobar si una clase hereda de otra.

```
>>> from collections import Iterable
>>> cadena = "Hola"
>>> numero = 3
>>> print("cadena", isinstance(cadena, Iterable
>>> print("numero", isinstance(numero, Iterable
cadena True
numero False
```

Python nos ofrece también diferentes métodos que pueden ser usados sobre clases iterables como los que se muestran a continuación:

- **list() convierte a lista una clase iterable**
- **sum() para sumar**
- **join() permite unir cada elemento de una clase iterable con el primer argumento**

Dejanos tu feedback! 👍

Contenido de la clase


```
>>> print(list("Hola"))
['H', 'o', 'l', 'a']

>>> print(sum([1, 2, 3]))
6

>>> print("-".join("Hola"))
H-o-l-a
```

De la misma forma que iteramos una cadena o una lista, también podemos iterar un diccionario. El iterador del diccionario devuelve las claves o keys del mismo.

```
>>> mi_dict = {'a':1, 'b':2, 'c':3}
>>> for i in mi_dict:
>>>     print(i)
a
b
c
```

Iteradores

Se podría explicar la diferencia entre iteradores e iterables usando un libro como analogía. El libro sería nuestra clase iterable, ya que tiene diferentes páginas a las que podemos acceder. El libro podría ser una lista, y cada página un elemento de la lista. Por otro lado, el iterador sería un marcapáginas, es decir, una referencia que nos indica en qué posición estamos del libro, y que puede ser usado para "navegar" por él.
Es posible obtener un iterador a partir de una clase iterable con la función iter(). En el siguiente ejemplo podemos ver como obtenemos el iterador del libro.

```
>>> libro = ['página1', 'página2', 'página3',
>>> marcapaginas = iter(libro)
```

Llegados a este punto, nuestro marcapáginas almacena un iterador. Se trata de un objeto que

Dejanos tu feedback! 👍

Contenido de la clase


```
>>> print(next(marcapaginas))
>>> print(next(marcapaginas))
>>> print(next(marcapaginas))
>>> print(next(marcapaginas))
página1
página2
página3
página4
```

Algo parecido a esto es lo que sucede por debajo cuando usamos el for sobre una clase iterable. Se va accediendo secuencialmente a los elementos hasta que la excepción StopIteration es lanzada. Dicha excepción se lanza cuando hemos llegado al final, y no existen más elementos que iterar.

```
>>> print(next(marcapaginas))
-------------------------------------------
StopIteration                             Trace
~\AppData\Local\Temp/ipykernel_10044/139163631
----> 1 print(next(marcapaginas))
```

Una nota muy importante es que cuando el iterador es obtenido con iter() como hemos visto, apunta por defecto fuera de la lista. Es decir, si queremos acceder al primer elemento de la lista, deberemos llamar una vez a next().
Por otro lado, a diferencia de un marcapáginas de un libro, el iterador sólo puede ir hacia delante. No es posible retroceder.

Sentencia zip

Dadas dos listas, digamos lista1 y lista2, al pasarlas a **zip** como entrada, el elemento 1 de lista1 se asocia con el elemento 1 de lista2, el elemento 2 de lista1 se asocia con el elemento 2 de lista2, el elemento 3 de lista1 se asocia con el elemento 3 de lista2, y así sucesivamente. Es decir que el resultado será una tupla donde cada elemento tendrá todos y cada uno de los elementos i-ésimos de las listas pasadas como entrada.

Dejanos tu feedback! 👍

```
>>> type(c)
zip
>>> list(c)
[(1, 'Uno'), (2, 'Dos')]
```

Contenido de la clase

Añadiendo condicionales

Hemos visto cómo modificar todos los elementos de un iterable (como una lista) de diferentes maneras, pero ¿y si quisiéramos realizar la operación sobre el elemento sólo si una determinada condición se cumple? Debemos añadir un condicional if.
La expresión genérica sería la siguiente.

lista = [expresión for elemento in iterable if condición]

Por lo tanto la expresión sólo se aplicará al elemento si se cumple la condición. Veamos un ejemplo con una frase, de la que queremos saber el número de erres que tiene.

```
>>> frase = "El perro de san roque no tiene rab
>>> erres = [i for i in frase if i == 'r']
>>> print(errores)
['r', 'r', 'r', 'r']
>>> print(len(erres))
4
```

Lo que hace el código anterior es iterar cada letra de la frase, y si es una r, se añade a la lista. De esta manera el resultado es una lista con tantas letras r como la frase original tiene, y podemos calcular las veces que se repite con len().

Homework

Completa la tarea descrita en el archivo **README**

Dejanos tu feedback! 👍

Iteradores e iterables | Python - Prep Course

Hecho con 💛 por alumnos de Henry

Dejanos tu feedback! 👍