# Taller Integrador - Python.

### Clase 12

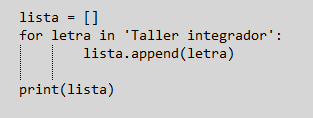
### Sábado 13/06/2020 (Chivilcoy)

### Martes 16/06/2020 CABA.

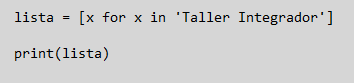
# Comprensión de listas.

La comprensión de listas nos permite resolver en pocas líneas de código, estructuras que normalmente nos llevarías varias líneas más.

Por ejemplo, si vamos queremos crear una lista con todas las letras de un mensaje, esto es lo que haríamos:

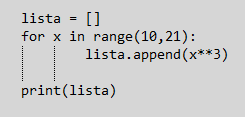


Si aplicamos la comprensión de listas de Python, lo haríamos de esta manera:

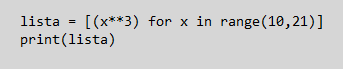


La sintaxis es muy sencilla, definimos la lista como veníamos haciendo, y dentro ponemos la variable donde vamos a guardar el recorrido del bucle, y a continuación el bucle en una sola línea.

OTRO EJEMPLO: queremos hacer una lista de los números del 10 al 20 elevados al cubo. La siguiente es la forma tradicional de resolverlo:

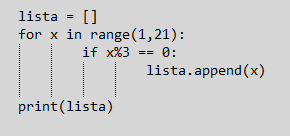


Mediante comprensión de listas lo resolveríamos de la siguiente manera:



OTRO EJEMPLO: Vamos a agregarle a nuestra lista una condición, por ejemplo, queremos crear una lista que contenga los múltiplos de 3 que están entre el 1 y el 20.

Método tradicional:



Mediante la comprensión de listas, lo que hacemos es agregar dicha condición, al final de nuestra lista:

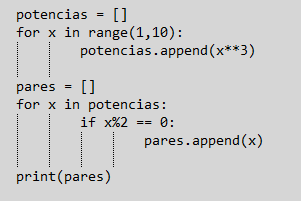
LISTA = [Variable a recorrer - Iteración - Condición]



OTRO EJEMPLO: un último ejemplo donde veremos cómo crear una lista a partir de otra, y ambas listas creadas mediante comprensión en una sola línea de código.

Problema: crear una lista que contenga los números pares, de una lista que contiene los números del 1 al 10 elevados al cubo.

Solución tradicional:



Solución mediante comprensión:

Lista = [Variable a recorrer – Iteración sobre lista - Condición]

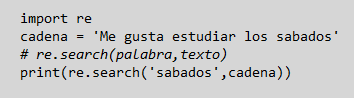


# Expresiones regulares.

Las expresiones regulares son aquellos patrones de datos que aparecen o se repiten en un documento.

El uso de estas expresiones nos permite hacer, desde búsquedas simples de una cadena dentro de un texto, hasta verificar el formato de un documento.

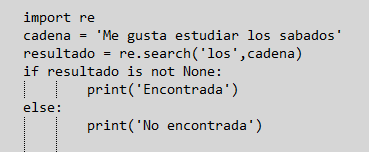
Por ejemplo, queremos buscar una palabra dentro de una cadena:



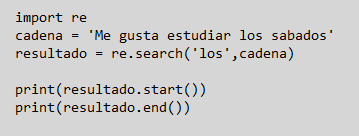
Lo que nos devuelve es un objeto del tipo Match, no nos sirve de mucho la información, así como viene:



En el caso de no encontrar la palabra o el patrón de caracteres que le pasamos, nos devolverá un tipo NONE, y podemos, mediante este resultado, armar una estructura simple para hacer más amigable dicho resultado:



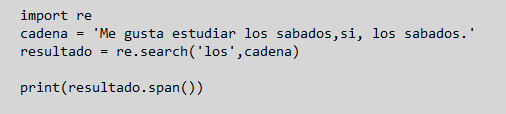
En el código anterior lo que hicimos fue crear un objeto llamado resultado, que tiene algunos métodos útiles, por ejemplo, los métodos START y END.



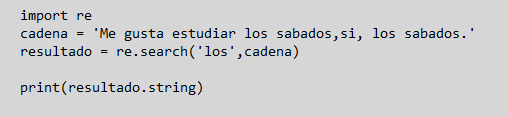
Estos métodos nos permiten encontrar donde empieza y donde termina el patrón buscado.

\*\* ¿Qué sucede si la palabra se repites dentro del texto?

METODO SPAN: nos devuelve una tupla con los valores de inicio y fin que obteníamos con los métodos anteriores.

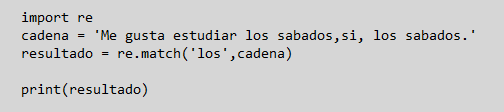


STRING: devuelve la cadena original sobre la que se realizó la búsqueda.

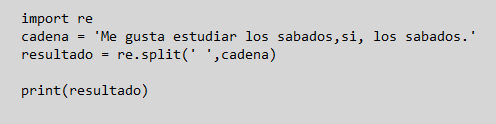


MATCH: busca un patrón, pero solo al principio de la cadena y no en toda, como hace SEARCH.

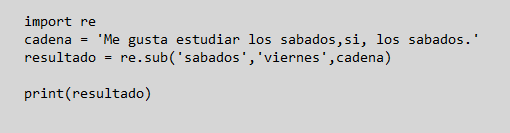
Devuelve un objeto si la encuentra o un NONE si no la encuentra.



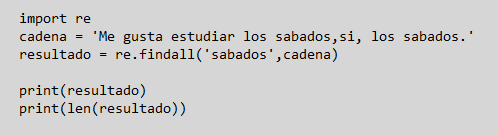
SPLIT: nos permite dividir una cadena en base a un patrón, por ejemplo, podemos armar una tupla en base a un texto donde cada elemento de la misma, sea una palabra de dicho texto.



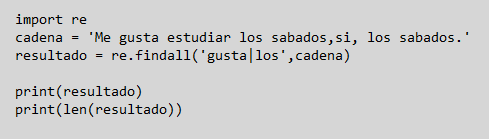
SUB: se utiliza para substituir un patrón por otro.



FINDALL: permite buscar todas las ocurrencias del patrón buscado dentro de una cadena.



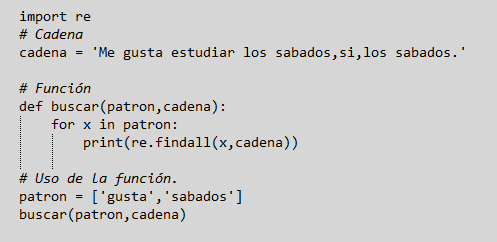
También admite el uso de OR:





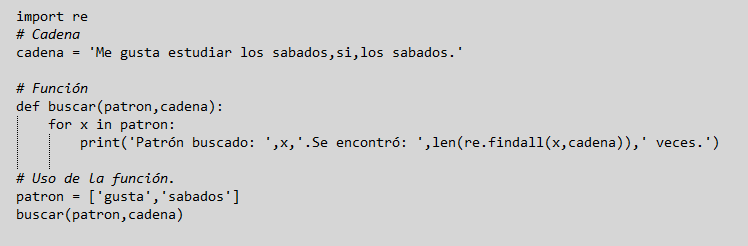
### Patrones con sintaxis repetida.

Podemos utilizar FINDALL en una función que reciba una lista de patrones a buscar y en base a dicha lista, efectuamos una búsqueda sobre nuestro texto.





Refinando un poquito la función, podemos ver la cantidad de veces que se repite cada patrón.

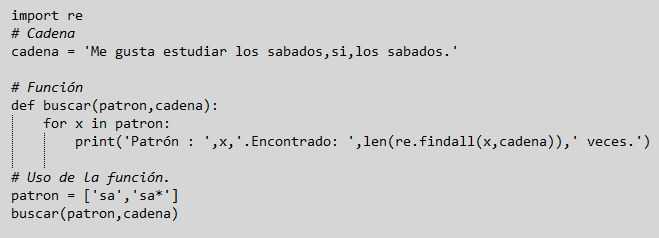




### Meta carácter \*.

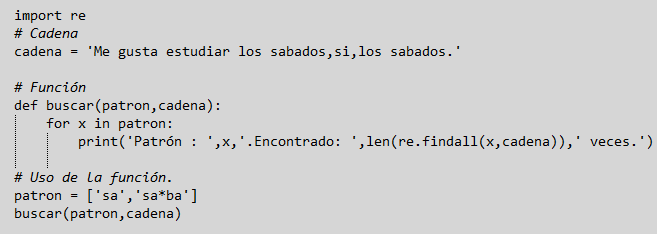
Un meta carácter es un comodín, y en particular el \* lo que hace es buscar ninguna o más repeticiones del carácter a su izquierda (más los caracteres a la izquierda de este).

Por ejemplo, queremos buscar en nuestra cadena de texto el patrón sa\* (es decir, buscaremos los patrones ‘s’ y ‘sa’).





Otra búsqueda que podríamos hacer mediante el uso del \* es la siguiente:



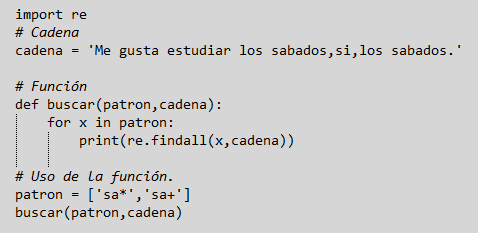
Lo que le estamos pidiendo a la función, es que busque los patrones que empiecen con ‘s’, ‘sa’ pero que a su vez, continúen con las letras ‘ba’.



**\*\*\* ¿Qué sucede si buscamos ‘ss’ o ‘s s’?**

### Meta carácter +.

El meta carácter + lo que hace es buscar una o más repeticiones del carácter a su izquierda (más los caracteres a la izquierda de este).





### Meta carácter ?

El meta carácter ? lo que hace es buscar una o ninguna repetición del carácter a su izquierda (más los caracteres a la izquierda de este).

