-Jupyter no hace falta poner print.

-Las funciones built-in son las propias de Python.

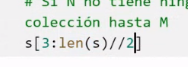
-Los saltos de línea se hacen con “\n”, siempre dentro de string. O sep=”/n” al final del print (este nos separa los elementos).

-Para buscar en la mitad de una colección, utilizamos la división absoluta (x // 2).

-Cogeremos posiciones consecutivas así.

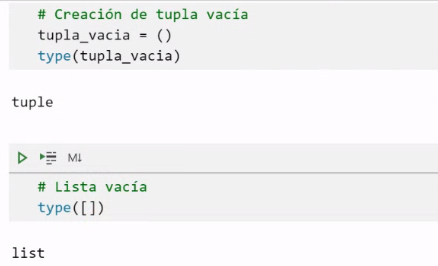


-Con la función len() y la división absoluta, podemos coger todo el tamaño de la lista y buscar el valor intermedio (x//2). Fíjate, aquí abajo, que estoy cojiendo una serie de números de la lista (en jupyter q viene con print predefinido) y hago división absoluta. Aquí no me cojerá el intermedio porque empiezo con el 3, pero se entiende).

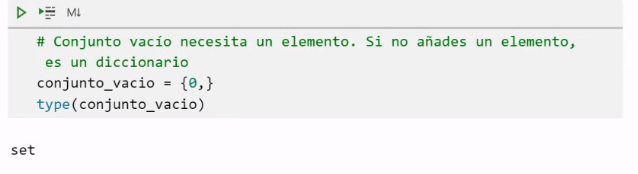


-Conjunto: no tienen índice, por lo que no se puede acceder a ellos por posición. Siempre que generamos un conjunto, se desordenará alfanuméricamente. Te ofrece quitar las repeticiones de elementos.

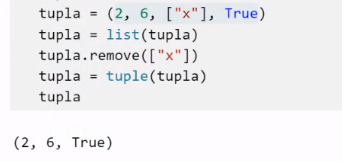
-Crear tuplas y listas.



-Crear conjunto y diccionario.



-Eliminar valores de una lista. Con .remove() quitamos VALORES con .pop() quitamos POSICIONES.

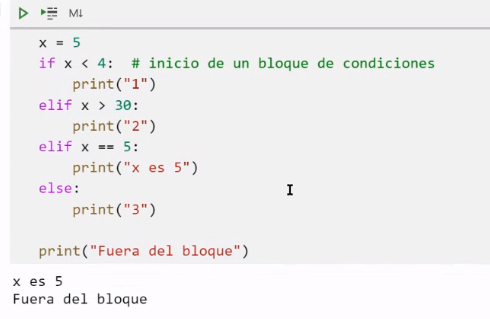


**Condiciones**

Los 2 puntitos en las condiciones son MUY IMPORTANTES!

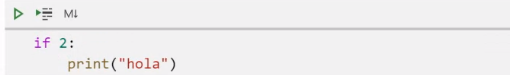


If, elif, else: Fijemonos en la imagen, que tiene == y se cumple, es indicación comparativa de igualdad. Con un solo =, no funciona. Elif puedo poner tantos como quiera para buscar distintas solicitudes de iniciación. El else funciona restando todo lo que le queda al if, por lo que solo podemos poner un else. Ni elif ni else son necesarios para que un comando funcione, if si lo es. Siempre, un sistema condicional, terminará cuando una condición sea cierta; si luego encontramos más que puedan ser ciertas, no las leerá.

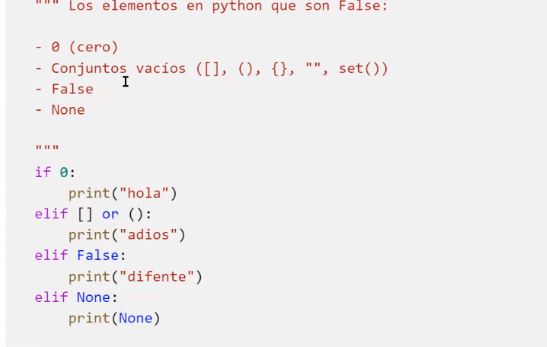


-Si después de un “if”, encontramos “and”, significa que tiene que cumplir 2 condiciones. Con el “or”, añadimos una condición, pero hará que se active la función si cualquiera de las 2 es válida, a diferencia del “and”, que requiere de las 2.

-Cuando nos encontramos un elemento solo en un condicional (sin comparativas o indicaciones de paso), es porque estamos buscando que exista en pyhton como TRUE.



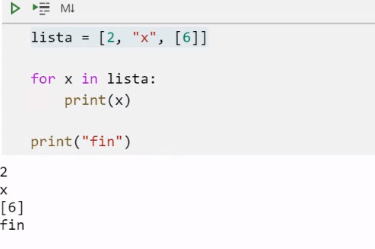
-En caso contrario, para estos elementos, es False y no saldrá nada al ejecutar, porque son todos detectados como False, por defecto.



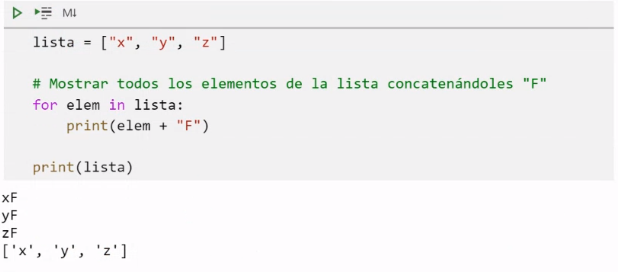
-El “not”, en un condicional, hace que un indicador FALSE pase a TRUE y viceversa. Por lo tanto, si encontramos algo como if not 0, será True directamente.

**BUCLES**

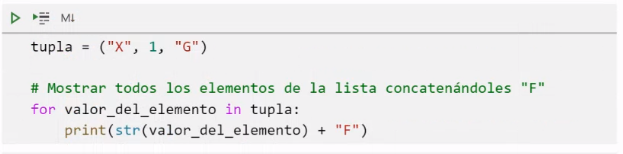
-Los bucles se escriben como aparecen en la imagen, hacen referencia a listas. La variable después de for coje todas las variables de la lista 1 por 1 y se transforma en cada 1 de ellas para luego repetir una acción con ellas 1 por 1.



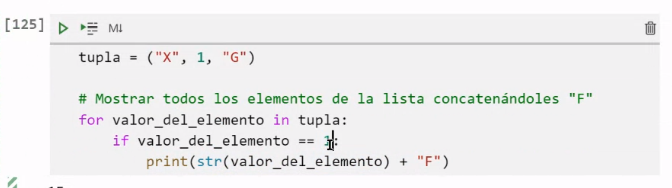
-En esta segunda imagen, vemos como la variable elem creada, actúa como sustituta de cada 1 de las variables de la lista para las repetidas acciones que toma después (sumar el string “F”).



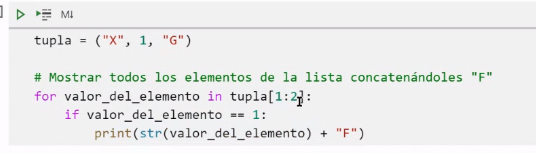
-En el caso de que tengamos problema con alguna variable, podemos aplicar funciones para modificarlas. Por ejemplo, como hacemos para modificar este int a str para poder sumar.



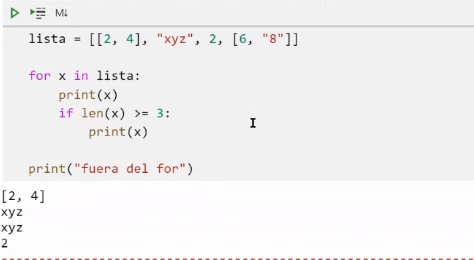
-Así hacemos que un bucle actue sobre los valores que cumplan una condición.



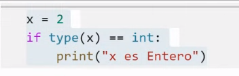
-Recordamos, que solo se puede utilizar un bucle para una colección por lo que lo que ahí abajo, funciona porque cogemos los elementos de una lista. Si en vez de [1:2], pongo [1], estoy cogiendo un valor, no una coleccion, por lo que no se ejecutaría el comando ya que no estoy actuando sobre una colección, sino sobre un valor.



-En el segundo 2 se efectua un error, porque estamos analizando a X como una colección con la función len(x). Al utilizar esa función, el objetivo X tiene que ser una colección (ya sea string, lista etc…). Entonces, el comando se ejecuta mostrando el print de 2,4 (con len 2), el print de xyz (con len 3) y se satisface además la condición de >=3 por lo que xyz vuelve a imprimirse en el segundo print. Sin embargo, al coger la tercera variable (2), el programa la imprime, pero la función len no logra completarla porque es un int, es decir ERROR.

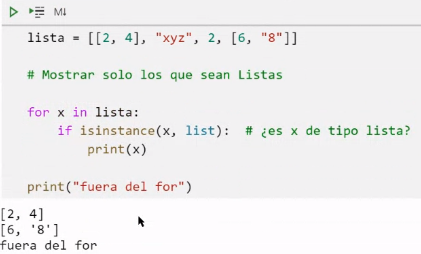


-Tenemos 2 maneras de establecer un condicional de type sobre una variable . Y podemos poner or, not etc… En el caso de isinstance, podemos establecer el or de esta manera (x ,(int, float).

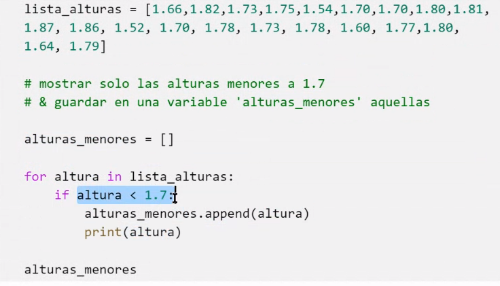




-Y así hacemos la comprobación de tipo para todos los elementos de una lista.



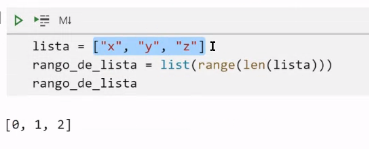
-Para coger los elementos de una lista con condicional y crear otra lista a partir de ello, es importante que creemos primero la variable como una lista vacía (paso que yo me olvidé). Ahora sabiendo que es una lista vacía, el programa, tras el for y el if, y utilizando un .append para ir añadiendo, cogerá los valores 1 por 1 y los irá añadiendo a la nueva lista cada vez que se ejecute ese for. Si no hacemos la lista, y aplicamos la variable como variable, el programa solo ejecutará su última rotación, y no todas las que se compilan para la lista.



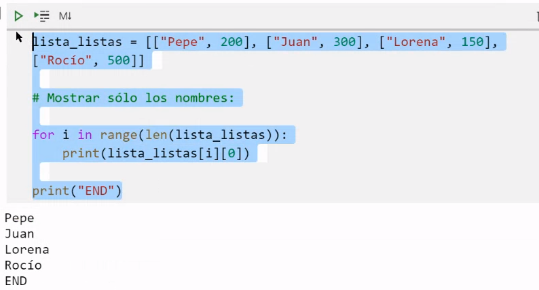
-OJO CUIDAO, en este caso, buscamos coger los str de la lista, pero tenemos que observar que esa lista, se compone de más listas, por lo que, tendremos que saber coger el elemento de cada lista que hay en esa lista matriz. Por lo que, hay que ser espavilaos, en este caso, cogemos el primer subelemento de cada elemento de la lista.



-Con el rango, sacamos, con la cantidad de variables que tiene una lista, posiciones.



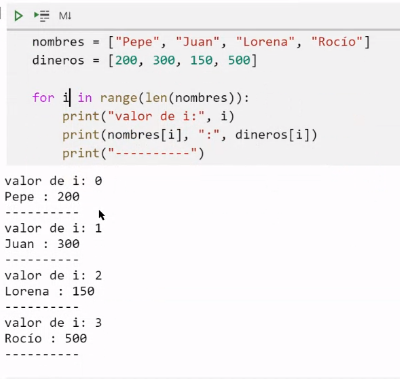
-Esto es muy clave, porque nos ayuda a recorrer las posiciones en trato con colecciones, y no los valores! MUY CLAVE. Fijate como en la imagen a continuación le damos a i el valor de cada POSICIÓN (si trabajamos con rangos, siempre cojemos posiciones) en cada vuelta (por cada posición). Luego en cada vuelta, se printea la posición i de la lista\_listas (es su formato original), en particular la primera posición de la posición i.



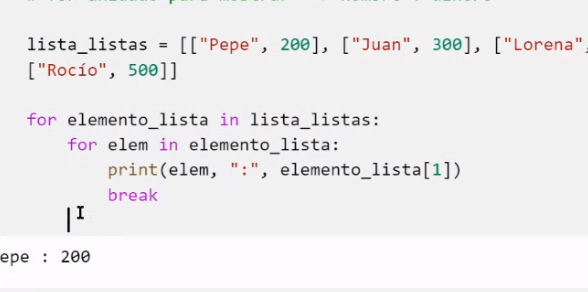
-Así, por ejemplo, podemos mostrar los nombres con sus respectivas cantidades de dinero si seguimos un orden líneal e igual de ambas listas.



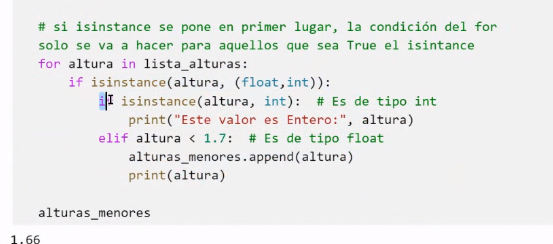
-Una bonita manera de representarlo también.



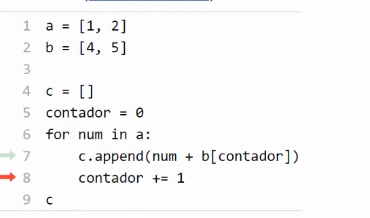
-El break nos vale para forzar un stop sobre el for que se ejecuta una y otra vez, es decir, solo lo ejecutará 1 vez, leído ese break.



EJERCICIO REPASO MUY BUENO: Con este ejercicio toco los types de una lista



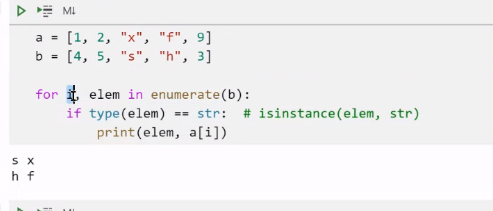
-Este ejercicio está muy bien, te muestra como sumar las posiciones de 2 listas de mismo rango. Para ello, como buscamos una lista nueva, para poder rellenarla c= [], y establecemos un contador para poder coger POSICIONES DIFERENTES en cada loop. Añadiendo ese contador (empieza en 0 por posición 0), se irá aumentando en cada loop, por 1, dando una posición a la derecha en cada loop.



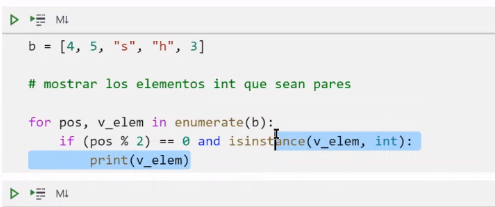
-La opción enumerate, te deja poner la posición y el valor de un elemento en un mismo for, para trabajar con ambos. “i” será la posición, en primer loop, la primera posición, y elem su valor.



-Ejercicio de posiciones y valores con enumerate: en este ejercicio, abro un loop de tabla b, le doy la condición de que el valor del elemento en posición i (nºloop) es tipo string, me imprima el string de a y el string de b. Es decir, sacar los strings de ambas tablas.



-Ejercicio de posiciones y valores con enumerate: aquí queremos coger los valores de posición par e int; como queremos tratar 2 condiciones, según su posición y valor, tratamos un enumerate!



-Y así podemos hacerlo con range, donde utilizamos posiciones de una lista que creamos

(ej no copiado)

. for i in range (5)

If type bla bla bla…

-En general, nos encontramos que tenemos / for / for + contador (que representa posición de elemento) / enumerate / range. Con todas estas, podemos solventar problemas de loop de posiciones y valores.