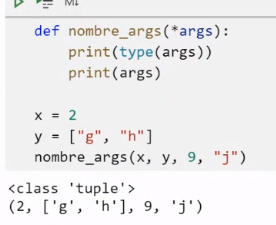
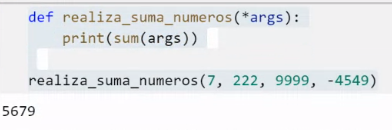
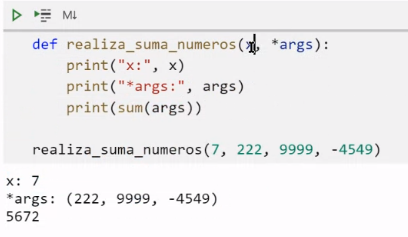
**\*args**

Con \*args en parámetro de la definición de una función, indicamos al usuario que no le especificamos un número de parámetros, puede poner los que quiera, y los usará todos.

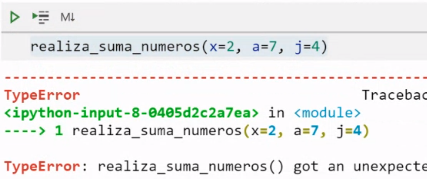




También puedo trabajar con \*args, y algun argumento especificado. El argumento especificado será siempre obligatorio, y el \*args, será opcional, y cogerá los valores que resten. Es decir, es una tupla de valores que no están asociados a ningún argumento obligatorio en la función.



No podemos asignar nombres de variable a \*args.

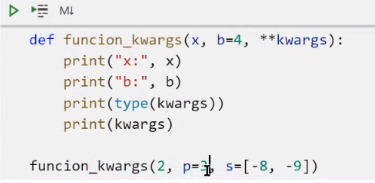


**\*\*Kwargs**

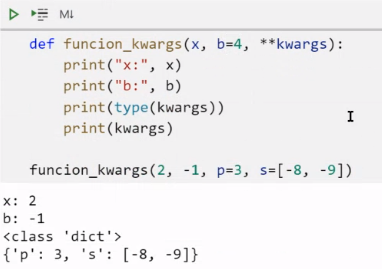
\*\*Kwargs es un argumento que representa un diccionario. Siempre tengo que asignarle variables, que luego me los ajusta como keys.



Y funciona más o menos como \*args, siempre se le asignan las variables que restan en los argumentos. En el siguiente caso, mete en diccionario p y s.

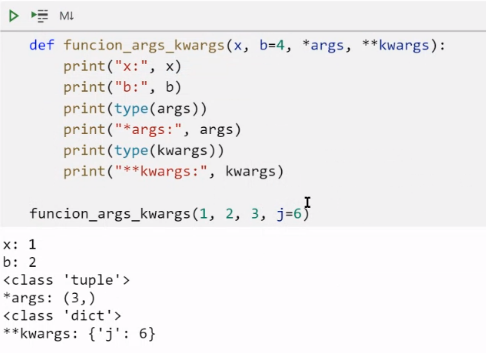


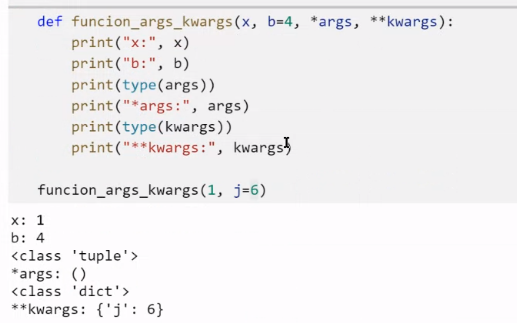
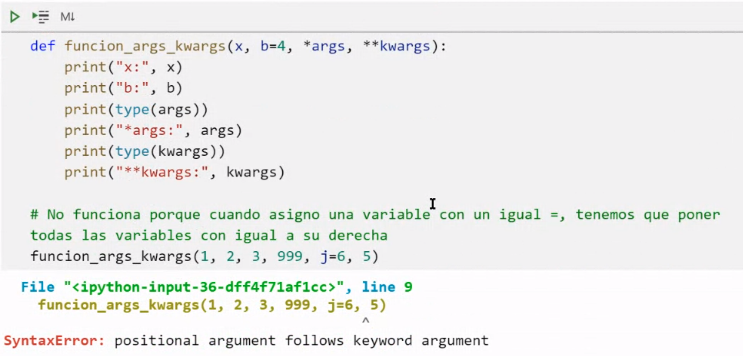
Mucho cuidao con como se cogen los argumentos! En este caso, b coge el -1.

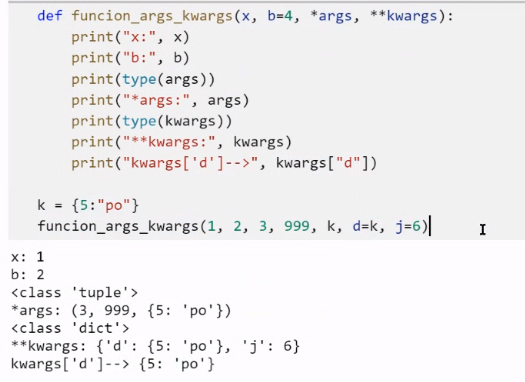




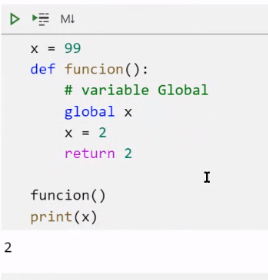
Los argumentos que especificamos, tienen prioridad ante args y kwargs. En \*args, y en \*\*kwargs solo entran los valores que van después de los parámetros especificados. Y siempre se ponen donde quepan. Si están asignados a una variable entra en \*\*kwargs, sino en \*args.



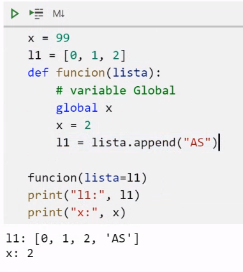
  


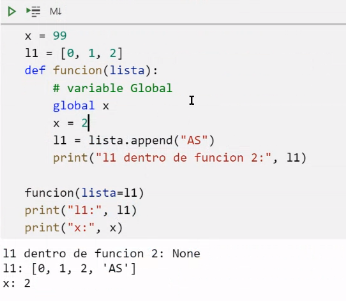


Dentro de una función puedo trabajar con una variable global con global. Si no hiciésemos es global, el print daría 99 porque, recordemos que, si trabajamos con variables creadas dentro de una función, siempre se quedan dentro de la función y no salen. Con global, salen, se hacen globales.



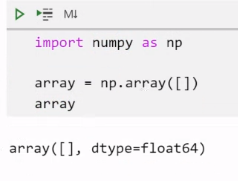
Caso muy raro: ¿Por que l1 no es None, si a l1 le damos el valor de una función con return None? 🡪 Porque al ser l1 una variable global, no podemos modificarla dentro de una función! La función .append, tiene definido dentro que modifica variables globales, pero nosotros, manualmente, no podemos cambiar una variable global dentro de una función, ya que ese cambio solo se dará dentro de la función. Pongo un segundo ejemplo que nos explica que dentro de la función es none, y dentro es l1 con el append. Ya que el append se lee.



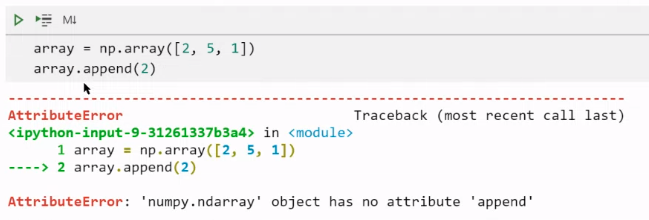


Librería Numpy

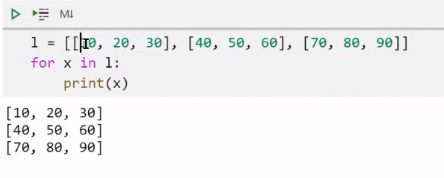
Vamos a utilizar una función de Numpy. Transformamos una lista a array.



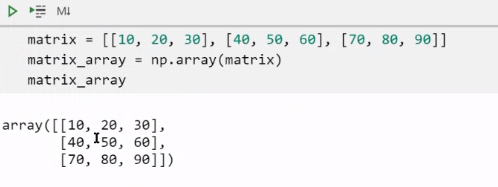
Con array no podemos utilizar funciones que vienen para listas.

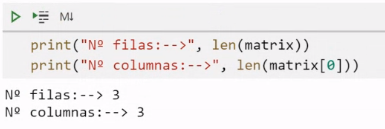


Mala praxis : crear una matriz con listas.

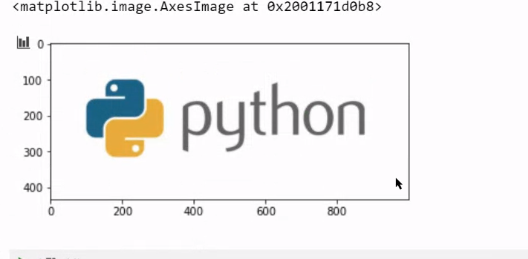


Buena praxis: utilizar numpy para trabajar con conjuntos de conjuntos, matrices… (Numpy te gestiona las nomenclaturas matemáticas).



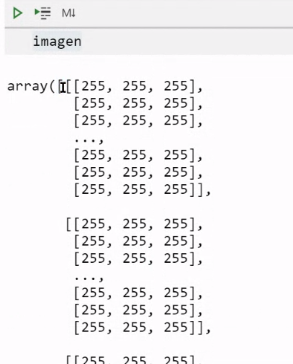


**Imágenes**

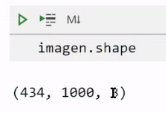


Los colores están desde el 0 al 255. El 0 representa el negro, el blanco representa el blanco.

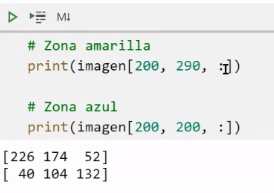
Cuando imprimimos la imagen, nos salen solo 255 porque nos imprime el principio del array y el final, que son todo blancos (255).



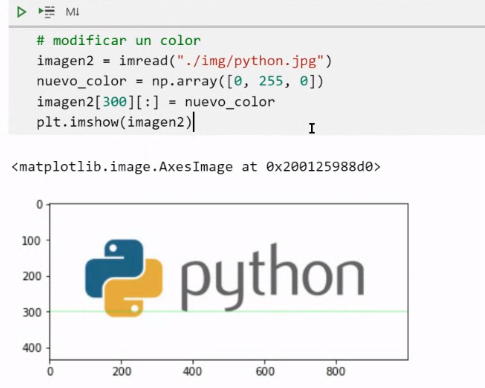
Una imagen viene representada por : (Filas, Columnas, Profundidad (color))



Así le decimos al programa que nos coja dimensiones dentro de un array y nos muestre sus valores. Es importante tener en cuenta, que aquí trabajamos con rgb (red, blue, green) y la fusión de estos 3 nos da un color. Y tabajamos con dimensiones (fila, columna, profundidad).



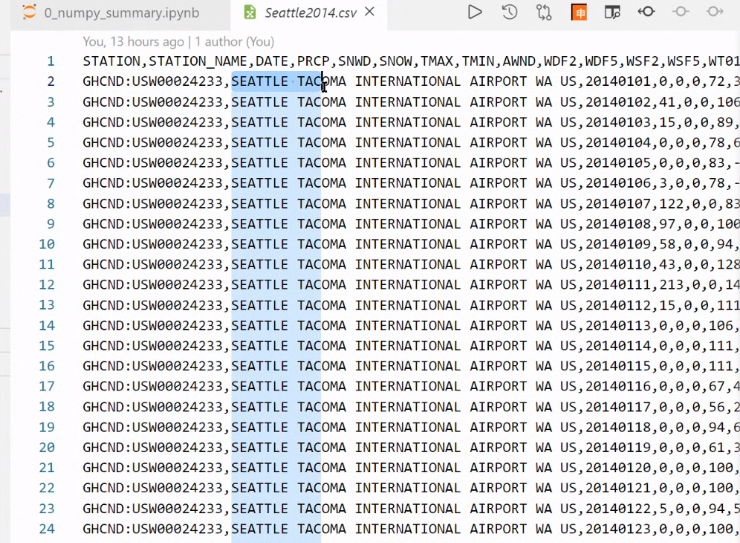
Así podemos modificar un array:



**Pandas**

Pandas sirven para indexar de una forma legible y ordenada

Pandas trae los datos y trabaja con filas y columnas. ComaSeparateValues .csv

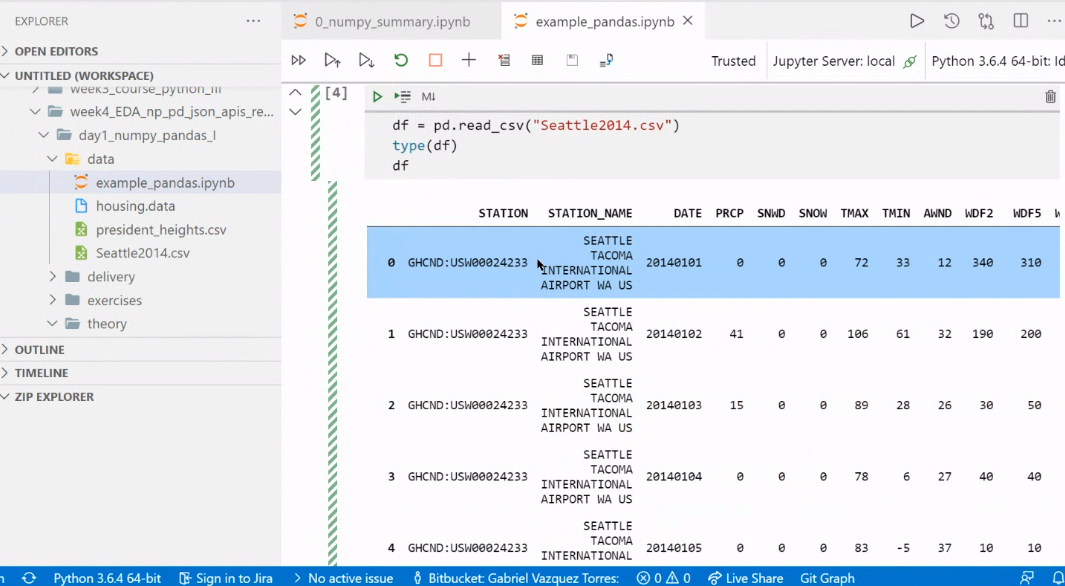


**Método**

-Creo un archivo ipynb.

-Importar pandas (en este caso lo importamos como pd.

-Utilizar la función read\_csv(“archivo”)



Cada columna es una serie. Y luego todo queda indexado por filas.

Podemos pasar esto a array con .values

