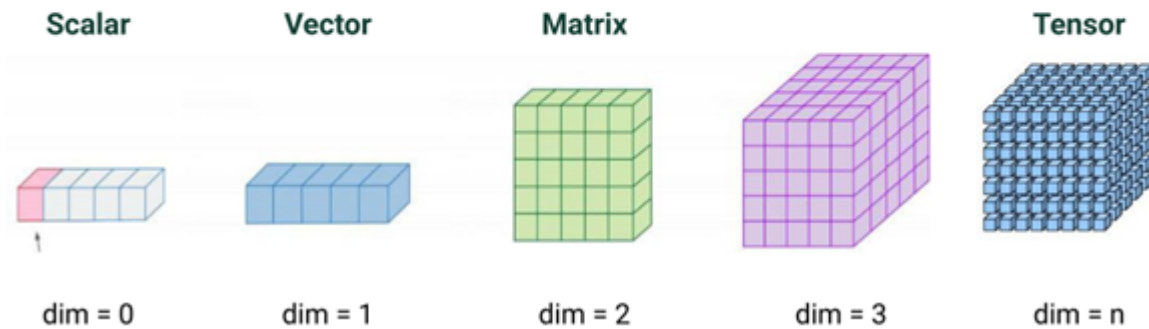


```
In [ ]: #Importando Numpy
import numpy as np
```

Dimensiones de los datos



Las dimensiones en Numpy se ven así.

- Cuando solo tengo un escalar es una dimensión 0
- Cuando tengo una lista es una dimensión 1
- Cuando tengo una matriz es una dimensión 2
- Cuando tengo un tensor es a partir de una dimensión 3

Como se puede observar este es un ejemplo en 2D o matriz

Matrix - 2D

(Ejemplos, características)

	A	B	C	D	E	F
1	Order ID	Product	Category	Amount	Date	Country
2	1	Carrots	Vegetables	\$4,270	1/6/2012	United States
3	2	Broccoli	Vegetables	\$8,239	1/7/2012	United Kingdom
4	3	Banana	Fruit	\$617	1/8/2012	United States
5	4	Banana	Fruit	\$8,384	1/10/2012	Canada
6	5	Beans	Vegetables	\$2,626	1/10/2012	Germany
7	6	Orange	Fruit	\$3,610	1/11/2012	United States
8	7	Broccoli	Vegetables	\$9,062	1/11/2012	Australia
9	8	Banana	Fruit	\$6,906	1/16/2012	New Zealand
10	9	Apple	Fruit	\$2,417	1/16/2012	France
11	10	Apple	Fruit	\$7,431	1/16/2012	Canada
12	11	Banana	Fruit	\$8,250	1/16/2012	Germany
13	12	Broccoli	Vegetables	\$7,012	1/18/2012	United States
14	13	Carrots	Vegetables	\$1,903	1/20/2012	Germany

Aquí las dimensiones están dadas por las filas y columnas

Este es un ejemplo de una aplicación en 3D

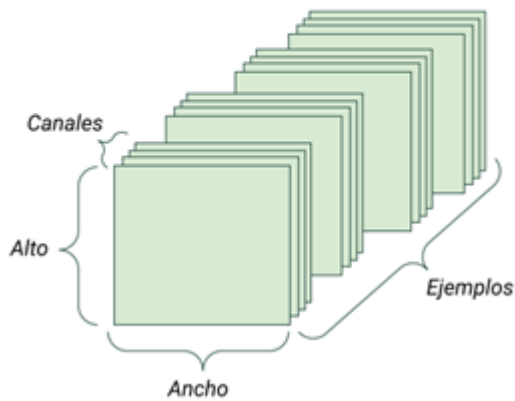
Tensor - 3D



Finalmente tenemos el 4D que podría ser imagen

Tensor - 4D - Images

(Ejemplos, Ancho, Alto, Canales de color)



```
In [ ]: #Dimensiones en Numpy
escalar = np.array(42)
print(f'Escalar = {escalar}\t\tNúmero de dimensiones = {escalar.ndim}')

vector = np.array([1, 2, 3, 4, 5])
print(f'Vector = {vector}\tNúmero de dimensiones = {vector.ndim}')

matriz = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(f'Matriz = \n{matriz}\t\tNúmero de dimensiones = {matriz.ndim}')

tensor = np.array([[ [1,2,2],[3,4,3],[5,6,8] ],[[5,6,2],[6,7,8],[8,9,10]]])
print(f'Tensor = \n{tensor}\t\tNúmero de dimensiones = {tensor.ndim}')
```

```
Escalar = 42                Número de dimensiones = 0
Vector = [1 2 3 4 5]       Número de dimensiones = 1
Matriz =
[[1 2 3]
 [4 5 6]
 [7 8 9]]                  Número de dimensiones = 2
Tensor =
[[[ 1  2  2]
  [ 3  4  3]
  [ 5  6  8]]
 [[ 5  6  2]
  [ 6  7  8]
  [ 8  9 10]]]             Número de dimensiones = 3
```

¿Pero que pasa si yo quiero agregar o eliminar dimensiones?

```
In [ ]: #Definiendo un vector con un número de dimensiones
vector = np.array([1,2,3],ndmin=5)
print(f'Vector = {vector}\nDimensiones = {vector.ndim}')
```

```
Vector = [[[[[1 2 3]]]]]
Dimensiones = 5
```

```
In [ ]: #Expandiendo una dimension en sus ejes
#axis = 0 es filas Axis = 1 son columnas
vector2 = np.expand_dims(np.array([1,2,3]),axis=0)
print(f'np.array([1,2,3]) = {np.array([1,2,3])}\nDimensiones = {np.array([1,2,3]).ndim}\n')
print(f'np.expand_dims(np.array([1,2,3]),axis=0) = {vector2}\nDimensiones = {vector2.ndim}')
```

```
np.array([1,2,3]) = [1 2 3]
Dimensiones = 1
```

```
np.expand_dims(np.array([1,2,3]),axis=0) = [[1 2 3]]
Dimensiones = 2
```

Esto se puede hacer tanto en el axis 0, como en el axis 1

```
In [ ]: #Eliminando dimensiones
vector = np.array([1,2,3],ndmin=5)
print(f'Vector = {vector}\nDimensiones = {vector.ndim}\n')

vector3 = np.squeeze(vector)
print('Vamos a eliminar dimensiones al vector y guardarlo en vector 3')
print(f'Vector 3 = {vector3}\nDimensiones = {vector3.ndim}\n')
```

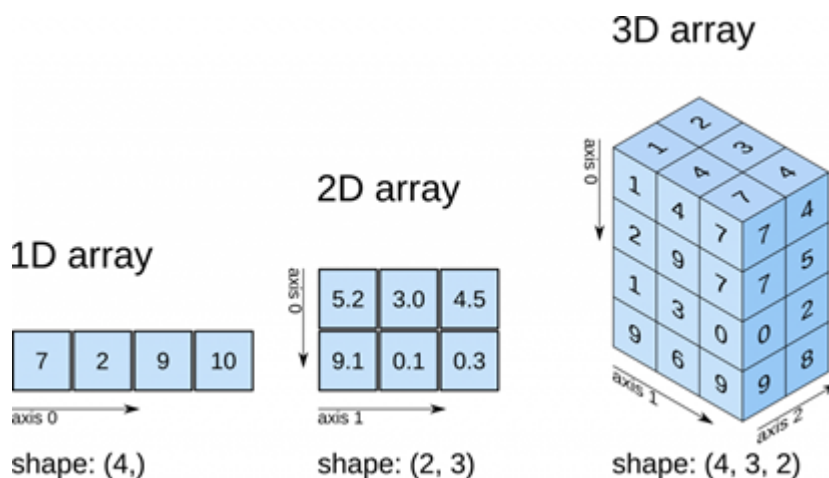
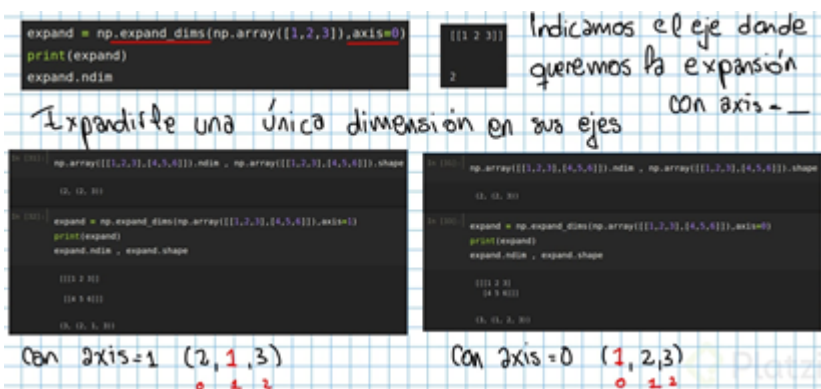
```
Vector = [[[[[1 2 3]]]]]
Dimensiones = 5
```

Vamos a eliminar dimensiones al vector y guardarlo en vector 3

```
Vector 3 = [1 2 3]
Dimensiones = 1
```

El método 'squeeze()' hace que el vector se comprima hasta el número de dimensiones que no se están usando

Recordatorio:



```
In [ ]: #Declaremos el ejemplo anterior para ver que hace la función shape
#Aquí se puede ver que este array tiene 4 elementos
```

```

array1 = np.array([7,2,9,10])
print(f'Array 1 = {array1}\t\tShape = {array1.shape}\n')
#Por ello shape arrojará 4 en la primera dimension

array2 = np.array([ [5.2,3.0,4.5],[9.1,0.1,0.3] ])
print(f'Array 2 = \n{array2}\t\t\tShape = {array2.shape}\n')

array3 = np.array([ [ [1,2],[4,3],[7,4] ], [ [2,5],[9,10],[7,5] ], [ [1,8],[3,7],[0,2] ], [ [9,3
print(f'Array 3 = \n{array3}\t\t\tShape = {array3.shape}\n')

```

Array 1 = [7 2 9 10] Shape = (4,)

Array 2 =
[[5.2 3. 4.5]
[9.1 0.1 0.3]] Shape = (2, 3)

Array 3 =
[[[1 2]
[4 3]
[7 4]]

[[2 5]
[9 10]
[7 5]]

[[1 8]
[3 7]
[0 2]]

[[9 3]
[6 5]
[9 8]]] Shape = (4, 3, 2)

Reto:

Define un tensor con almenos 5 dimensiones e intenta sumar una dimensión en cualquiera de sus ejes o intenta eliminar las dimensiones que no están importando en el objeto que definiste

In []: *#Voy a tomar como referencia la variable array 1; realizar el expand & squeeze*

```

#Proceso de expand a array1
print(f'Array 1 = {array1}\t\tShape = {array1.shape}\n')
print('Expandiendo en dim 1 (filas) => axis = 0')
array1_2 = np.expand_dims(array1,axis=0)
print(f'Array 1v2 = {array1_2}\t\tShape = {array1_2.shape}\n')

#Aplicando squeeze Array 1v2
print('Aplicando squeeze a array 1v2')
array1_2 = np.squeeze(array1_2)
print(f'Array 1v3 = {array1_2}\t\tShape = {array1_2.shape}\n')

#Proceso de expand a array1
print('Expandiendo en dim 2 (columnas) => axis = 1')
array1_2 = np.expand_dims(array1,axis=1)
print(f'Array 1v2 = \n{array1_2}\t\tShape = {array1_2.shape}\n')

#Aplicando squeeze a Array 1v2
print('Aplicando squeeze a array 1v2')

```

```
array1_2 = np.squeeze(array1_2)
print(f'Array 1v3 = {array1_2}\t\tShape = {array1_2.shape}\n')
```

Array 1 = [7 2 9 10] Shape = (4,)

Expandiendo en dim 1 (filas) => axis = 0

Array 1v2 = [[7 2 9 10]] Shape = (1, 4)

Aplicando squeeze a array 1v2

Array 1v3 = [7 2 9 10] Shape = (4,)

Expandiendo en dim 2 (columnas) => axis = 1

Array 1v2 =

```
[[ 7]
 [ 2]
 [ 9]
 [10]]                      Shape = (4, 1)
```

Aplicando squeeze a array 1v2

Array 1v3 = [7 2 9 10] Shape = (4,)

In []: *#Voy a tomar como referencia la variable array 2; realizar expand & squeeze*

#Imprimiendo el valor de array 2

```
print(f'Array 1 = \n{array2}\t\tShape = {array2.shape}\n')
```

#Proceso de expand a array2

```
print('Expandiendo en dim 1 (filas) => axis = 0')
```

```
array2_2 = np.expand_dims(array2,axis=0)
```

```
print(f'Array 1v2 = \n{array2_2}\tShape = {array2_2.shape}\n')
```

#Aplicando squeeze Array 2v2

```
print('Aplicando squeeze a array 2v2')
```

```
array2_2 = np.squeeze(array2_2)
```

```
print(f'Array 1v3 = \n{array2_2}\t\tShape = {array2_2.shape}\n')
```

#Proceso de expand a array2

```
print('Expandiendo en dim 2 (columnas) => axis = 1')
```

```
array2_2 = np.expand_dims(array2,axis=1)
```

```
print(f'Array 1v2 = \n{array2_2}\tShape = {array2_2.shape}\n')
```

#Aplicando squeeze Array 2v2

```
print('Aplicando squeeze a array 2v2')
```

```
array2_2 = np.squeeze(array2_2)
```

```
print(f'Array 1v3 = \n{array2_2}\t\tShape = {array2_2.shape}\n')
```

```

Array 1 =
[[5.2 3.  4.5]
 [9.1 0.1 0.3]]          Shape = (2, 3)

Expandiendo en dim 1 (filas) => axis = 0
Array 1v2 =
[[[5.2 3.  4.5]
  [9.1 0.1 0.3]]]        Shape = (1, 2, 3)

```

```

Aplicando squeeze a array 2v2
Array 1v3 =
[[5.2 3.  4.5]
 [9.1 0.1 0.3]]          Shape = (2, 3)

Expandiendo en dim 1 (columnas) => axis = 1
Array 1v2 =
[[[5.2 3.  4.5]]
 [[9.1 0.1 0.3]]]        Shape = (2, 1, 3)

```

```

Aplicando squeeze a array 2v2
Array 1v3 =
[[5.2 3.  4.5]
 [9.1 0.1 0.3]]          Shape = (2, 3)

```

In []: *#Quiero verificar que no se puede expandir en una dimensión que no existe, en este caso en dim 3,*

```

#Proceso de expand a array2
print('Expandiendo en dim 3 => axis = 2')
array2_2 = np.expand_dims(array2,axis=2)
print(f'Array 1v2 = \n{array2_2}\tShape = {array2_2.shape}\n')

```

```

Expandiendo en dim 3 => axis = 2
Array 1v2 =
[[[5.2]
  [3. ]
  [4.5]]

 [[9.1]
  [0.1]
  [0.3]]]      Shape = (2, 3, 1)

```

In []: *#Si se pudo pero ahora comprobemos que no podemos brinca de dim 3 a dim 5; axis = 4*

```

#Proceso de expand a array2
print('Expandiendo en dim 5 => axis = 4')
array2_2 = np.expand_dims(array2,axis=4)
print(f'Array 1v2 = \n{array2_2}\tShape = {array2_2.shape}\n')

```

```

Expandiendo en dim 5 => axis = 4

```

AxisError

Traceback (most recent call last)

Cell In[48], line 4

```
1 #Si se pudo pero ahora comprobemos que no podemos brinca de dim 3 a dim 5; axis = 4
2 #Proceso de expand a array2
3 print('Expandiendo en dim 5 => axis = 4')
----> 4 array2_2 = np.expand_dims(array2,axis=4)
5 print(f'Array 1v2 = \n{array2_2}\tShape = {array2_2.shape}\n')
```

File ~/miniforge3/lib/python3.10/site-packages/numpy/lib/shape_base.py:597, in `expand_dims(a, axis)`

```
594     axis = (axis,)
596 out_ndim = len(axis) + a.ndim
--> 597 axis = normalize_axis_tuple(axis, out_ndim)
599 shape_it = iter(a.shape)
600 shape = [1 if ax in axis else next(shape_it) for ax in range(out_ndim)]
```

File ~/miniforge3/lib/python3.10/site-packages/numpy/core/numeric.py:1380, in `normalize_axis_tuple(axis, ndim, argname, allow_duplicate)`

```
1378     pass
1379 # Going via an iterator directly is slower than via list comprehension.
-> 1380 axis = tuple([normalize_axis_index(ax, ndim, argname) for ax in axis])
1381 if not allow_duplicate and len(set(axis)) != len(axis):
1382     if argname:
```

File ~/miniforge3/lib/python3.10/site-packages/numpy/core/numeric.py:1380, in `<listcomp>(.0)`

```
1378     pass
1379 # Going via an iterator directly is slower than via list comprehension.
-> 1380 axis = tuple([normalize_axis_index(ax, ndim, argname) for ax in axis])
1381 if not allow_duplicate and len(set(axis)) != len(axis):
1382     if argname:
```

AxisError: axis 4 is out of bounds for array of dimension 3

Como podemos comprobar nos manda el siguiente mensaje:

AxisError: axis 4 is out of bounds for array of dimension 3

Esto quiere decir que la dimensión a expandir está fuera del alcance

```
In [ ]: #Ahora vamos a usar la variable array 3 para crear otra variable con una dimensión minima de 5
#Recordando a la variable array 3
print(f'Array 3 = \n{array3}\t\t\tShape = {array3.shape}')

#Creamos una nueva variable con una dimensión minima de 5
array4 = np.array(array3,ndmin=5)
print(f'Array 4 = \n{array4}\t\t\tShape = {array4.shape}')
```


Array 3 =

```
[[[ 1  2]
   [ 4  3]
   [ 7  4]]
```

```
[[ 2  5]
 [ 9 10]
 [ 7  5]]
```

```
[[ 1  8]
 [ 3  7]
 [ 0  2]]
```

```
[[ 9  3]
 [ 6  5]
 [ 9  8]]]
```

Shape = (4, 3, 2)

Array 4 =

```
[[[[[ 1  2]
     [ 4  3]
     [ 7  4]]
```

```
[[ 2  5]
 [ 9 10]
 [ 7  5]]
```

```
[[ 1  8]
 [ 3  7]
 [ 0  2]]
```

```
[[ 9  3]
 [ 6  5]
 [ 9  8]]]]]
```

Shape = (1, 1, 4, 3, 2)