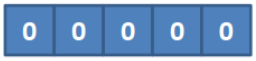
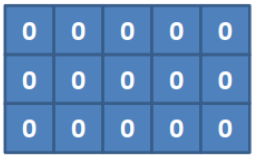
😎Indexación y Slicing en arrays de Numpy

rango / rank

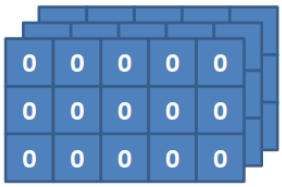
El rango de una matriz es simplemente el número de ejes (o dimensiones) que tiene. Una array simple tiene rango 1:



Una matriz bidimensional (también llamada matriz) tiene rango 2:



Una matriz tridimensional tiene rango 3. Aquí se muestra como una pila de matrices:



Numpy permite tener tantas dimensiones como desee

Indexando una matriz

La **indexación** se utiliza para obtener elementos individuales de una matriz, pero también se puede utilizar para obtener filas, columnas o planos completos de matrices multidimensionales.

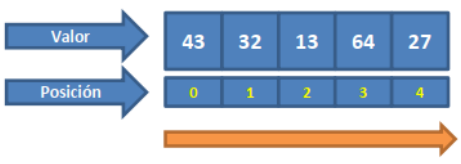
In [1]:
```
1 import numpy as np
```

Indexación 1D

Podemos crear una matriz numérica de 1 dimensión a partir de una lista como esta:

In [2]:
```
1 m1 = np.array([43, 32, 13, 64, 27])
2 m1
```

Out[2]: array([43, 32, 13, 64, 27])



Podemos indexar en esta matriz para obtener un elemento individual, exactamente igual que una lista o tupla normal:

In [3]:
```
1 m1[0], m1[2], m1[4]
```

Out[3]: (43, 13, 27)

In [4]:
```
1 m1[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 m1[5]

IndexError: index 5 is out of bounds for axis 0 with size 5
```

Emitir con un índice fuera de rango devuelve el mensaje de error: "el índice 5 está fuera de límites para el axis 0 con tamaño 5"

In [5]:
```
1 print(f"Un subset del array:\n{m1[-3:]}\nOtro subset del array:\n {m1[2:5]}")
```

```
Un subset del array:
[13 64 27]
Otro subset del array:
 [13 64 27]
```

In [6]:
```
1 m1[1:4], m1[1:], m1[:4], m1[:]
```

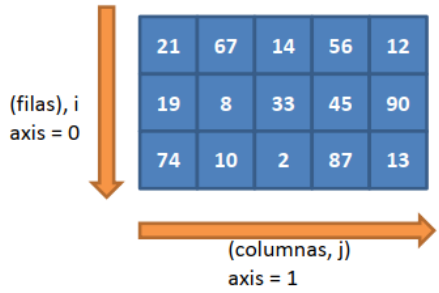
Out[6]:
```
(array([32, 13, 64]),
 array([32, 13, 64, 27]),
 array([43, 32, 13, 64]),
 array([43, 32, 13, 64, 27]))
```

Indexación 2D

Podemos indexar un elemento de la matriz usando dos índices: i selecciona la fila y j selecciona la columna.

In [7]:
```
1 m2 = np.array([[21, 67, 14,56, 12], [19,8,33,45,90], [74, 10, 2, 87,13]])
2 m2
```

Out[7]:
```
array([[21, 67, 14, 56, 12],
       [19,  8, 33, 45, 90],
       [74, 10,  2, 87, 13]])
```



In [8]:
```
1 m2[2, 1], m2[0, 3]
```

Out[8]:
```
(10, 56)
```

In [9]:
```
1 print(f"Elemento en los índices[1][1]: {m2[1][1]}")
2 print(f"Elementos individuales: {m2[0,1]}")
3 print(f"Elementos individuales: {m2[2,1]}")
```

```
Elemento en los índices[1][1]: 8
Elementos individuales: 67
Elementos individuales: 10
```

In [10]:
```
1 print(f"Array de elementos de la fila 1: {m2[1,:]}")
2 print(f"Vector de elementos de la columna 0: {m2[:,0]}")
3 print(f"Submatriz de 2x2 con las primeras dos filas:\n {m2[0:2,:]}")
4 print(f"Submatriz de 2x2 con las ultimas dos filas:\n {m2[1:3,:]}")
```

```
Array de elementos de la fila 1: [19  8 33 45 90]
Vector de elementos de la columna 0: [21 19 74]
Submatriz de 2x2 con las primeras dos filas:
 [[21 67 14 56 12]
 [19  8 33 45 90]]
Submatriz de 2x2 con las ultimas dos filas:
 [[19  8 33 45 90]
 [74 10  2 87 13]]
```

Observar los errores en qué axis se dan:

In [11]:
```
1 print(m2[1, 5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [11], in <cell line: 1>()
----> 1 print(m2[1, 5])

IndexError: index 5 is out of bounds for axis 1 with size 5
```

In [12]:
```
1 print(m2[3, 1])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [12], in <cell line: 1>()
----> 1 print(m2[3, 1])

IndexError: index 3 is out of bounds for axis 0 with size 3
```

In [13]:
```
1 print(m2[3, 6])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [13], in <cell line: 1>()
----> 1 print(m2[3, 6])

IndexError: index 3 is out of bounds for axis 0 with size 3
```

Observemos la sintaxis:

Los valores *i* y *j* están dentro de los corchetes, separados por una coma (el índice es en realidad una tupla (2, 1) , pero se usa el **empaquetado de tuplas**. El ejemplo selecciona la fila 2, columna 1, que tiene el valor 10. Esto se compara con la sintaxis que podría usar con una lista 2D (es decir, una lista de listas):

Elegir una fila o columna:

Si podemos proporcionar un solo índice, elegirá una fila (valor *i*) y la devolverá como una matriz de rango 1:

In [14]:
```
1 m2, m2[2]
```

Out[14]:
```
(array([[21, 67, 14, 56, 12],
        [19,  8, 33, 45, 90],
        [74, 10,  2, 87, 13]]),
 array([74, 10,  2, 87, 13]))
```

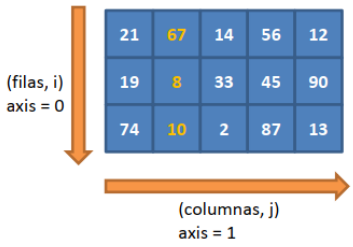
Eso es bastante similar a lo que sucedería con una lista 2D. Sin embargo, numpy también permite seleccionar una sola columna, lo que significa esta sintaxis es:

- para el valor *i* , tome todos los valores (**:** es un segmento completo, de principio a fin)
- para el valor *j* , tome 1

Dando esta matriz [67,8,10]:

In [15]:
```
1 m2[:, 1]
```

Out[15]:
```
array([67,  8, 10])
```



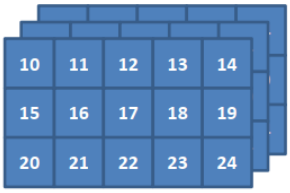
La matriz que obtiene cuando indexa o corta una matriz numpy es una **vista** de la matriz original. Son los mismos datos, solo que se accede en un orden diferente. Si cambia la vista, cambiará los elementos correspondientes en la matriz original.

Indexación 3D

Podemos crear una matriz numérica tridimensional a partir de una lista de Python de listas de listas, como esta:

In [16]:
```
m3 = np.array([[[10,11,12,13,14], [15,16,17,18,19], [20,21,22,23,24]],
               [[25, 26, 27,28,29], [30,31,32,33,34],
               [35,36,37,38,39]], [[40,41,42,43,44],
                   [45,46,47,48,49], [50,51,52,53,54]]])
m3
```

Out[16]:
```
array([[[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24]],

       [[25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39]],

       [[40, 41, 42, 43, 44],
        [45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54]]])
```



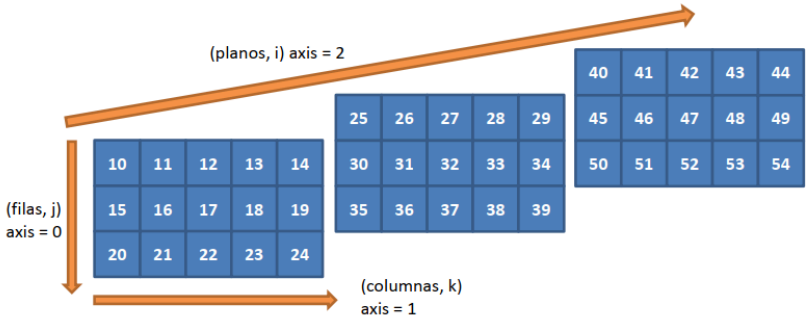
Una matriz 3D es como una pila de matrices:

•El primer índice, i , selecciona los planos de la matriz

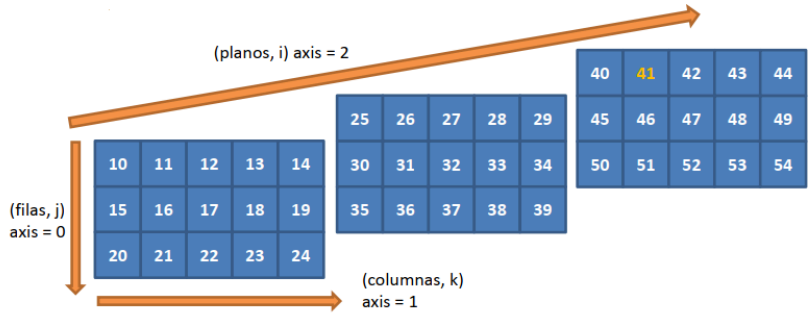
•El segundo índice, j , selecciona la fila

•El tercer índice, k , selecciona la columna

Aquí está el mismo diagrama, extendido para que podamos ver los valores:



A continuación, se explica cómo indexar un valor particular en una matriz 3D:



Esto selecciona el índice del plano de la matriz 2, fila 0, columna 1, dando el valor 41.

Elegir una fila o columna en una matriz 3D:

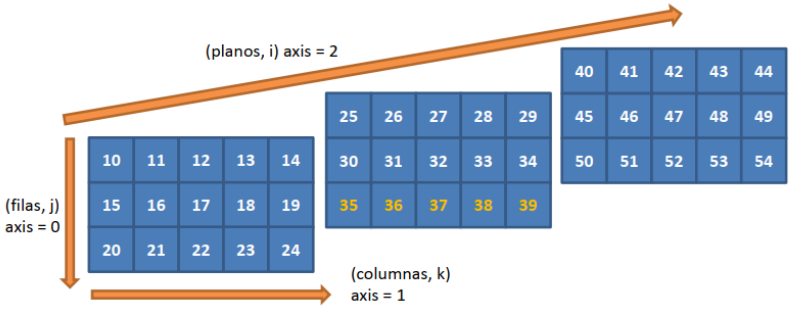
Puede acceder a cualquier fila o columna de una matriz 3D. Hay 3 casos.

Caso 1 : especificación de los dos primeros índices.

In [17]:
```
m3[1, 2]
```

Out[17]:
```
array([35, 36, 37, 38, 39])
```

En este caso, está eligiendo el valor i, plano de la matriz, y el valor j (la fila). Esto seleccionará una fila específica. En este ejemplo, estamos seleccionando la fila 2 de la matriz 1:



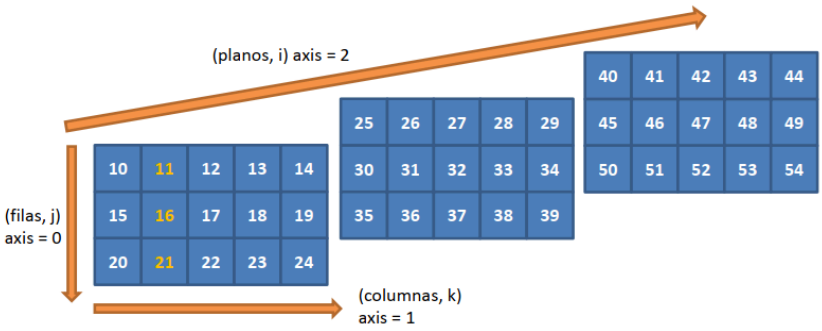
Esto selecciona el índice del plano de matriz 1, fila 2, dando los valores 35,36,37,38,39

Caso 2 - especificando el i valor, planos de la matriz, y la k valor de la columna, utilizando un subconjunto completo (:) para el j valor fila.

```
In [18]:  1 m3[0, :, 1]
```

```
Out[18]: array([11, 16, 21])
```

Esto seleccionará una columna específica. En este ejemplo, estamos seleccionando la columna 1 del plano de la matriz 0:



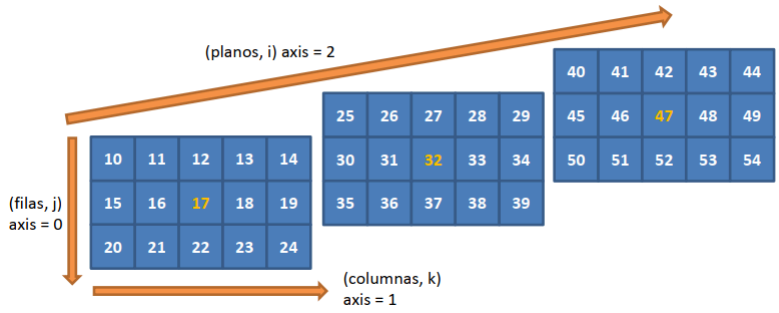
Esto selecciona el índice del plano de matriz 0, todas las filas de la columna 1, dando los valores 11, 16, 21

Caso 3 - especificando el j valor de la fila, y la k valor de la columna, utilizando una rebanada completa (:) para el i valor del plano de la matriz.

```
In [19]:  1 m3[:, 1, 2]
```

```
Out[19]: array([17, 32, 47])
```

Esto creará una fila tomando el mismo elemento de cada matriz. En este caso, estamos tomando la fila 1, la columna 2 de la matriz:



Esto selecciona todos los planos de la matriz, filas 1, columnas 2, dando los valores 17,32,47

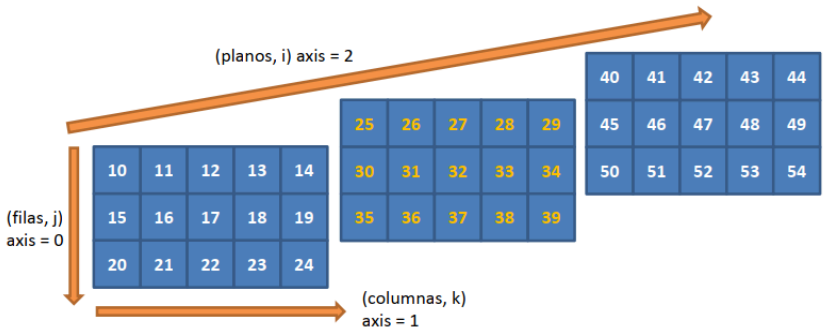
Elegir una matriz en una matriz 3D

Caso 1:

```
In [20]:  1 m3[1]
```

```
Out[20]: array([[25, 26, 27, 28, 29],
                [30, 31, 32, 33, 34],
                [35, 36, 37, 38, 39]])
```

Si solo especificamos el índice i , numpy devolverá la matriz correspondiente. En este ejemplo solicitaremos la matriz 1:

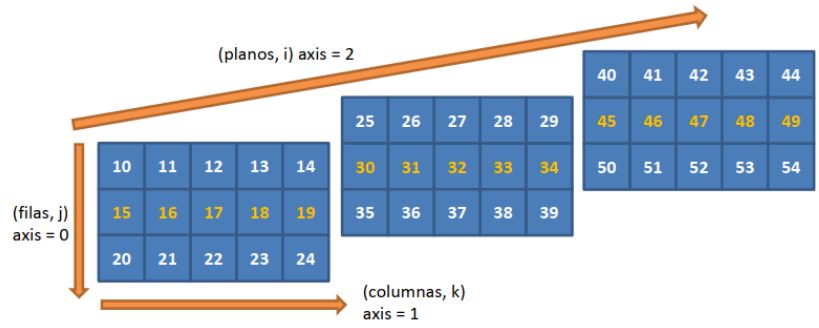


Caso 2:

```
In [21]:  1 m3[:, 1]
```

```
Out[21]: array([[15, 16, 17, 18, 19],
                [30, 31, 32, 33, 34],
                [45, 46, 47, 48, 49]])
```

si especificamos solo el valor j (usando un corte completo para los valores i), obtendremos una matriz hecha de la fila seleccionada tomada de cada plano. En este ejemplo tomaremos la fila 1:

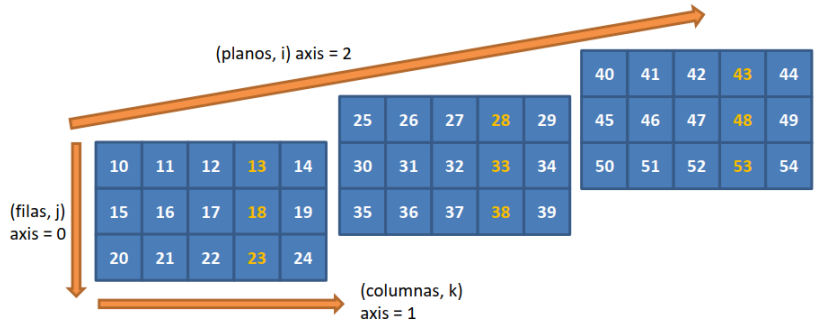


Caso 3:

```
In [22]:  1 m3[:, :, 3]
```

```
Out[22]: array([[13, 18, 23],
                [28, 33, 38],
                [43, 48, 53]])
```

si especificamos solo el valor k (usando cortes completos para los valores i y j), obtendremos una matriz hecha de la columna seleccionada tomada de cada plano. En este ejemplo tomaremos la columna 3:



Cortar una matriz

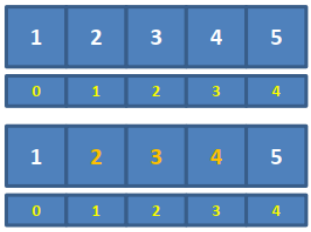
Puede dividirse una matriz numérica de una manera similar a dividir una lista, excepto que puede hacerse en más de una dimensión.
Al igual que con la indexación, la matriz que se obtiene cuando se indexa o corta una matriz numpy es una vista de la matriz original. Son los mismos datos, solo que se accede en un orden diferente. Esto es diferente a las listas, donde un segmento devuelve una lista completamente nueva.

Slicing 1D en arrays de numpy

Cortar una matriz numpy 1D es casi exactamente lo mismo que cortar una lista:

In [23]:
```
m1 = np.array([1, 2, 3, 4, 5])
m1[1:4]
```

Out[23]:
```
array([2, 3, 4])
```



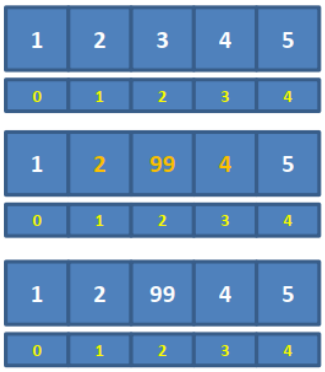
In [24]:
```
corte_m1 = m1[1:4]
corte_m1
```

Out[24]:
```
array([2, 3, 4])
```

A diferencia de una lista, ambos, dim1 y corte_dim1, miran los mismos datos subyacentes (corte_dim1 es una vista de los datos). Entonces, si cambia un elemento en corte_dim1, afectará a dim1 (y viceversa):

In [25]:
```
corte_m1[1] = 99
corte_m1, m1
```

Out[25]:
```
(array([ 2, 99,  4]), array([ 1,  2, 99,  4,  5]))
```

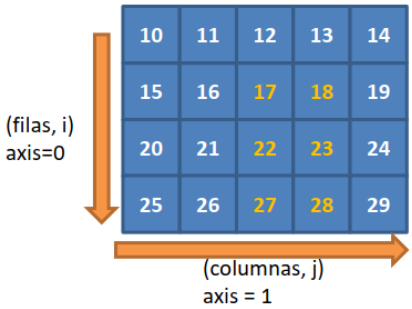


Slicing 2D en arrays de numpy

Se puede cortar una matriz 2D en ambos ejes para obtener un subconjunto rectangular de la matriz original. Por ejemplo:

In [26]:
```
m2 = np.array([[10, 11, 12, 13, 14], [15, 16, 17, 18, 19],
                   [20, 21, 22, 23, 24], [25, 26, 27, 28, 29]])
m2[1:,2:4]
```

Out[26]:
```
array([[17, 18],
       [22, 23],
       [27, 28]])
```



Esto selecciona a partir de la fila 1: (1 hasta el final de la matriz) y las columnas 2: 4 (columnas 2 y 3), como se muestra en la figura.

Se puede utilizar cortes completos : (dos puntos) para seleccionar todos los planos, columnas o filas. Sin embargo, para los índices finales, simplemente hay que omitir el índice y cuenta como un segmento completo. Entonces, para matrices 2D:

In [27]:
```
1 m2[1:4,:]
```

Out[27]:
```
array([[15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

Ambas expresiones devuelven lo mismo:

In [28]:
```
1 m2[1:3], m2[1:3,:]
```

Out[28]:
```
(array([[15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24]]),
 array([[15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24]]))
```

También puede usarse un segmento de longitud 1 para hacer algo similar (segmento 1: 2 en lugar del índice 1):

In [29]:
```
1 m2[1,2:4], m2[1:2,2:4]
```

Out[29]:
```
(array([17, 18]), array([[17, 18]]))
```

Note la sutil diferencia. El primero crea una matriz 1D, el segundo crea una matriz 2D con una sola fila.

Slicing 3D en arrays de numpy

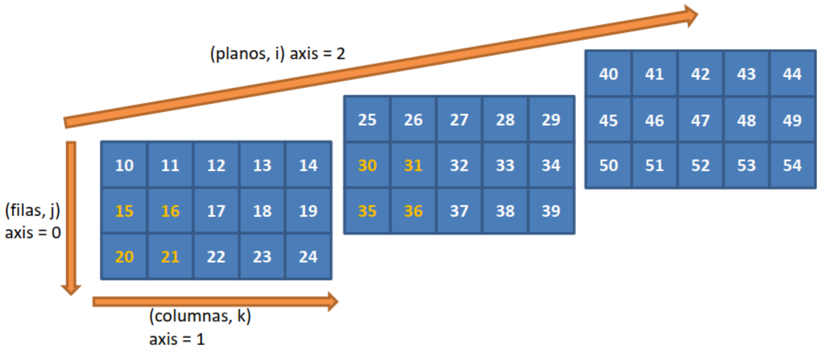
Puede cortar una matriz 3D en los 3 ejes para obtener un subconjunto cuboide de la matriz original:

In [30]:
```
1 m3 = np.array([[[10,11,12,13,14], [15,16,17,18,19], [20,21,22,23,24]],
2                [[25, 26, 27,28,29], [30,31,32,33,34],
3                   [35,36,37,38,39]], [[40,41,42,43,44],
4                     [45,46,47,48,49], [50,51,52,53,54]]])
5 m3
```

Out[30]:
```
array([[[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24]],

       [[25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39]],

       [[40, 41, 42, 43, 44],
        [45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54]]])
```

In [31]:
```
1 m3[:2,1:,:2]
```

Out[31]:
```
array([[[15, 16],
        [20, 21]],

       [[30, 31],
        [35, 36]]])
```



Esto selecciona:

- los 2 primeros planos
- a partir de la fila 1, el resto
- las 2 primeras columnas

In [32]:
```
print(f"{m3[1:,:2,:]}\n\n{m3[1:,:2]}\nAmbas expresiones devuelven el mismo resultado")
```

```
[[[25 26 27 28 29]
  [30 31 32 33 34]]

 [[40 41 42 43 44]
  [45 46 47 48 49]]]

[[[25 26 27 28 29]
  [30 31 32 33 34]]

 [[40 41 42 43 44]
  [45 46 47 48 49]]]
Ambas expresiones devuelven el mismo resultado
```

También:

In [33]:
```
m3[1:,:,:], m3[1:,:], m3[1:], m3[1:3]
```

Out[33]:
```
(array([[[25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34],
         [35, 36, 37, 38, 39]],

        [[40, 41, 42, 43, 44],
         [45, 46, 47, 48, 49],
         [50, 51, 52, 53, 54]]]),
 array([[[25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34],
         [35, 36, 37, 38, 39]],

        [[40, 41, 42, 43, 44],
         [45, 46, 47, 48, 49],
         [50, 51, 52, 53, 54]]]),
 array([[[25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34],
         [35, 36, 37, 38, 39]],

        [[40, 41, 42, 43, 44],
         [45, 46, 47, 48, 49],
         [50, 51, 52, 53, 54]]]),
 array([[[25, 26, 27, 28, 29],
         [30, 31, 32, 33, 34],
         [35, 36, 37, 38, 39]],

        [[40, 41, 42, 43, 44],
         [45, 46, 47, 48, 49],
         [50, 51, 52, 53, 54]]]))
```