

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 lista simple tiene rango 1:

0	0	0	0	0
---	---	---	---	---

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

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

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

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Numpy permite tener tantas dimensiones como desee.

Forma /Shape:

Especifica la longitud de la matriz en cada dimensión. Suele representarse como una tupla. Para una matriz dada, el número de elementos en la tupla de forma, será igual al rango de la matriz.

Nuestra matriz de rango 1 anterior tiene 5 elementos, por lo que su forma es la tupla **(5,)** .

0	0	0	0	0
---	---	---	---	---

La matriz de rango 2 tiene 3 filas y 5 columnas, por lo que sus formas son (3, 5) . Por convención, tomamos la forma de la matriz como filas primero, luego columnas.

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

En la matriz de rango 3 sus formas son (3, 3, 5) . Por convención, tomamos la forma de la matriz como planos primero, luego filas y luego columnas.

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

La forma de una matriz se puede encontrar utilizando el atributo **ndim**

Talla / Size

Es el número total de elementos. Se encuentra multiplicando todos los elementos de la forma.

Nuestra matriz de rango 1 anterior tiene 5 elementos, por lo que su tamaño es 5.

0	0	0	0	0
---	---	---	---	---

La matriz de rango 2 tiene forma de 3 por 5, por lo que su tamaño es 15 (hay 15 elementos en total).

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

La matriz de rango 3 tiene forma de 3 por 5 por 3, por lo que su tamaño es 45 (hay 45 elementos en total).

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

El tamaño de una matriz se puede encontrar mediante el atributo de **size**

Tipo de datos /Data type

A diferencia de una lista de Python, las matrices numpy se componen de *tipos de datos primitivos*, típicamente ints o floats, que se almacenan directamente en la memoria sin ninguna información adicional.

Por ejemplo, el tipo de datos **numpy.int32** es un entero de 32 bits que ocupa exactamente 4 bytes (32 bits) de memoria.

Numpy ofrece diferentes tamaños de entradas que puede elegir para que coincidan con los datos con los que está tratando. Por ejemplo, si está procesando datos de imagen, los valores rojo, verde y azul de cada píxel generalmente se representarán como un valor entero de 8 bits.

Podría almacenar la imagen en una matriz numerosa de valores int32, pero eso usaría 4 veces más memoria de la necesaria, por lo que sería mejor usar el tipo de datos int8.

El tipo de datos de una matriz se puede encontrar mediante el atributo **dtype** .

Otros parámetros de matriz

Puede encontrar el tamaño de un elemento en una matriz utilizando el atributo **itemsize**.

El tamaño del elemento está determinado por el tipo de datos, por lo que, por ejemplo, una matriz `int32` tendrá un tamaño de elemento si es 4 (32 bits dividido 8 = 4 bytes). Una matriz `int8` con un tamaño de elemento de 1, etc.

También puede obtener la dirección de memoria de los datos de la matriz real, utilizando el atributo de **data**. No es frecuente si se está escribiendo un programa en Python puro, porque Python no puede acceder directamente a las ubicaciones de la memoria, pero puede ser útil si se está llamando a otras bibliotecas de bajo nivel que tienen enlaces de Python y pueden aceptar punteros de memoria como parámetros.

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 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. Esto es diferente a las listas, donde un segmento devuelve una lista completamente nueva.

#Ejemplo

```
import numpy as np
```

1D

```
dim1 = np.zeros((5) )
```

```
print(dim1)
```

```
'''
```

```
[0. 0. 0. 0. 0.]
```

```
'''
```

```
print(dim1.ndim)    # 1
```

```
print(dim1.shape)   # (5,)
```

```
print(dim1.size)    # 5
```

```
print(dim1.dtype)   # float64
```

```
print(dim1.itemsize) # 4
```

```
print(dim1.data)    # <memory at 0x000001F795CE8700
```

```
>
```

2D

```
dim2 = np.zeros((3, 5))
```

```
print(dim2)
```

```
'''
```

```
[[0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]]
```

```
'''
```

```
print(dim2.ndim)    # 2
```

```
print(dim2.shape)   # (3, 5)
```

```
print(dim2.size)    # 15
```

```
print(dim2.dtype)   # float64
```

```
print(dim2.itemsize) # 8 (float64 is an 8 byte quantity)
```

```
print(dim2.data)    # <memory at 0x000001F795C74AD0>
```

3D

```
dim3 = np.zeros((2,3,5))
```

```
print(dim3)
```

```
'''
```

2 planos de 3 filas por 5 columnas

```
[[[0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]]
```

```
[[[0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]]]
```

```
'''
```

```
print(dim3.ndim)    # 3
```

```
print(dim3.shape)   # (2, 3, 5)
```

```
print(dim3.size)    # 30
```

```
print(dim3.dtype)   # float64
```

```
print(dim3.itemsize) # 8 (float64 is an 8 byte quantity)
```

```
print(dim3.data)    # <memory at 0x000001F78D179400>
```

Analicemos algunos atributos útiles de la matriz. Comenzaremos definiendo tres matrices aleatorias, una matriz unidimensional, bidimensional y tridimensional. Usaremos el generador de números aleatorios de NumPy, que cargaremos con un valor establecido para garantizar que se generen las mismas matrices aleatorias cada vez que se ejecute este código:

```
import numpy as np
```

```
np.random.seed(0) # semilla para la reproducibilidad
```

```
dim1 = np.random.randint(10, size=6) # 1-dimensional array
```

```
print(dim1)
```

```
dim2 = np.random.randint(10, size=(3, 4)) # 2-dimensional array
```

```
print(dim2)
```

```
dim3 = np.random.randint(10, size=(3, 4, 5)) # 3-dimensional array
```

```
print(dim3)
```

```
>>> import numpy as np
>>> np.random.seed(0) # semilla para la reproducibilidad
>>> dim1 = np.random.randint(10, size=6) # 1-dimensional array
>>> print(dim1)
[5 0 3 3 7 9]
>>> dim2 = np.random.randint(10, size=(3, 4)) # 2-dimensional array
>>> print(dim2)
[[3 5 2 4]
 [7 6 8 8]
 [1 6 7 7]]
>>> dim3 = np.random.randint(10, size=(3, 4, 5)) # 3-dimensional array
>>> print(dim3)
[[[8 1 5 9 8]
  [9 4 3 0 3]
  [5 0 2 3 8]
  [1 3 3 3 7]]
 [[0 1 9 9 0]
  [4 7 3 2 7]
  [2 0 0 4 5]
  [5 6 8 4 1]]
 [[4 9 8 1 1]
  [7 9 9 3 6]
  [7 2 0 3 5]
  [9 4 4 6 4]]]
```

Otros atributos de la matriz incluyen `itemsize`, que enumeran el tamaño (en bytes) de cada elemento de la matriz y `nbytes`, que enumera el tamaño total (en bytes) de la matriz:

```
print("itemsize:", dim3.itemsize, "bytes")
```

```
print("nbytes:", dim3.nbytes, "bytes")
```

```
>>> print("itemsize:", dim3.itemsize, "bytes")
itemsize: 4 bytes
>>> print("nbytes:", dim3.nbytes, "bytes")
nbytes: 240 bytes
```

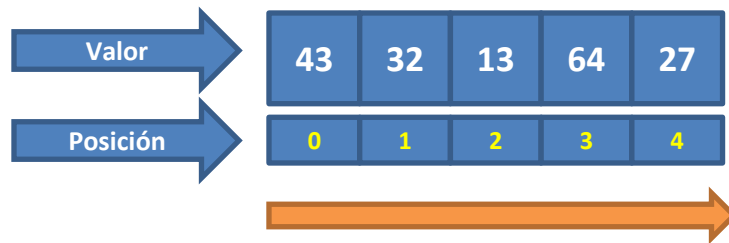
En general, esperamos que `nbytes` sea igual a `itemsize` veces `size`.

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 .

Indexación 1D

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



```
import numpy as np
dim1 = np.array([43, 32, 13, 64, 27])
print(dim1)
```

```
>>> import numpy as np
>>> dim1 = np.array([43, 32, 13, 64, 27])
>>> print(dim1) # [43 32 13 64 27]
[43 32 13 64 27]
```

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

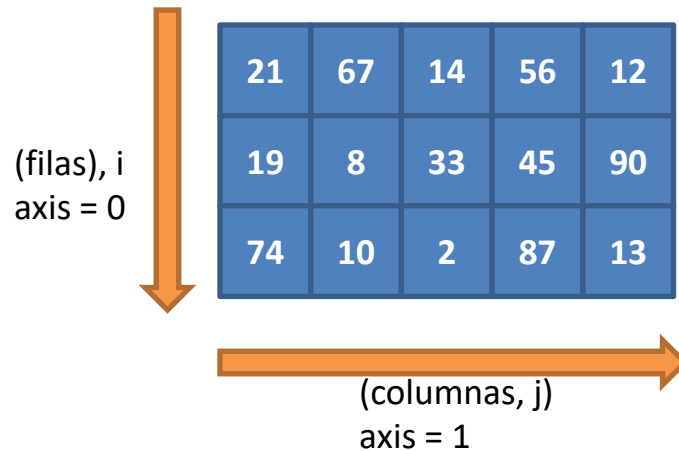
```
print(dim1[0])
print(dim1[2])
print(dim1[4])
print(dim1[5])
```

```
>>> print(dim1[0])
43
>>> print(dim1[2])
13
>>> print(dim1[4])
27
>>> print(dim1[5])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 5 is out of bounds for axis 0 with size 5
```

Cuando queremos emitir con un índice fuera de rango el mensaje de error es: *el índice 5 está fuera de límites para el **axis 0** con tamaño 5*

Indexación 2 D

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



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

```
import numpy as np
```

```
dim2 = np.array([[21, 67, 14, 56, 12],  
                [19, 8, 33, 45, 90],  
                [74, 10, 2, 87, 13]])
```

```
print(dim2)  
print(dim2[2, 1])  
print(dim2[0, 3])
```

```
print(dim2[1, 5])
```

```
print(dim2[3, 1])
```

```
print(dim2[3, 6])
```

Observar los
errores en
qué axis se
dan.

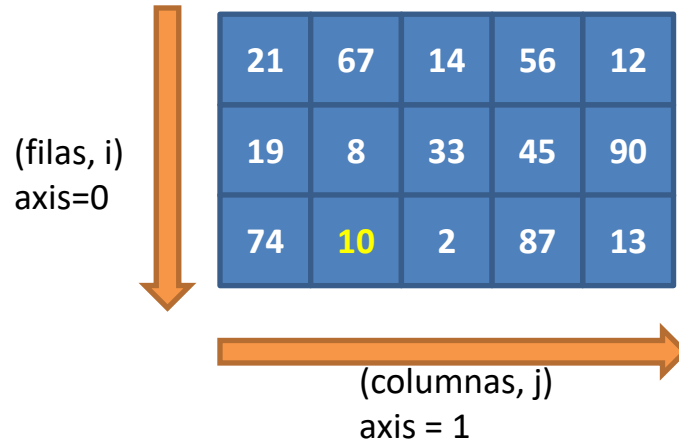
```
>>> print(dim2)  
[[21 67 14 56 12]  
 [19  8 33 45 90]  
 [74 10  2 87 13]]  
>>> print(dim2[2, 1])  
10  
>>> print(dim2[0, 3])  
56
```

```
>>> print(dim2[1, 5])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IndexError: index 5 is out of bounds for axis 1 with size 5
```

```
>>> print(dim2[3, 1])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IndexError: index 3 is out of bounds for axis 0 with size 3
```

```
>>> print(dim2[3, 6])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
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:

```
print(dim2[2])
```

```
>>> print(dim2[2])  
[74 10  2 87 13]  
>>>
```

Eso es bastante similar a lo que sucedería con una lista 2D.

Sin embargo, numpy también nos 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]:

21	67	14	56	12
19	8	33	45	90
74	10	2	87	13

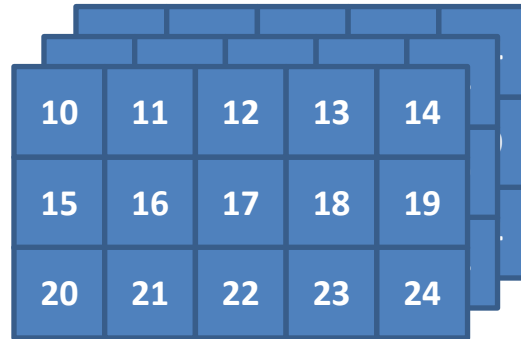
`print(dim2[:, 1])`

```
>>> print(dim2[:, 1])  
[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:



10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

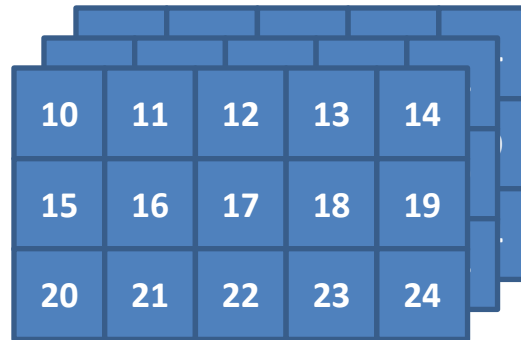
#3D

```
import numpy as np
```

```
dim3 = 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]]])  
print(dim3)
```

```
>>> import numpy as np  
>>>  
>>> dim3 = 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]]])  
>>>  
>>> print(dim3)  
[[[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]]]
```

Indexación 3D

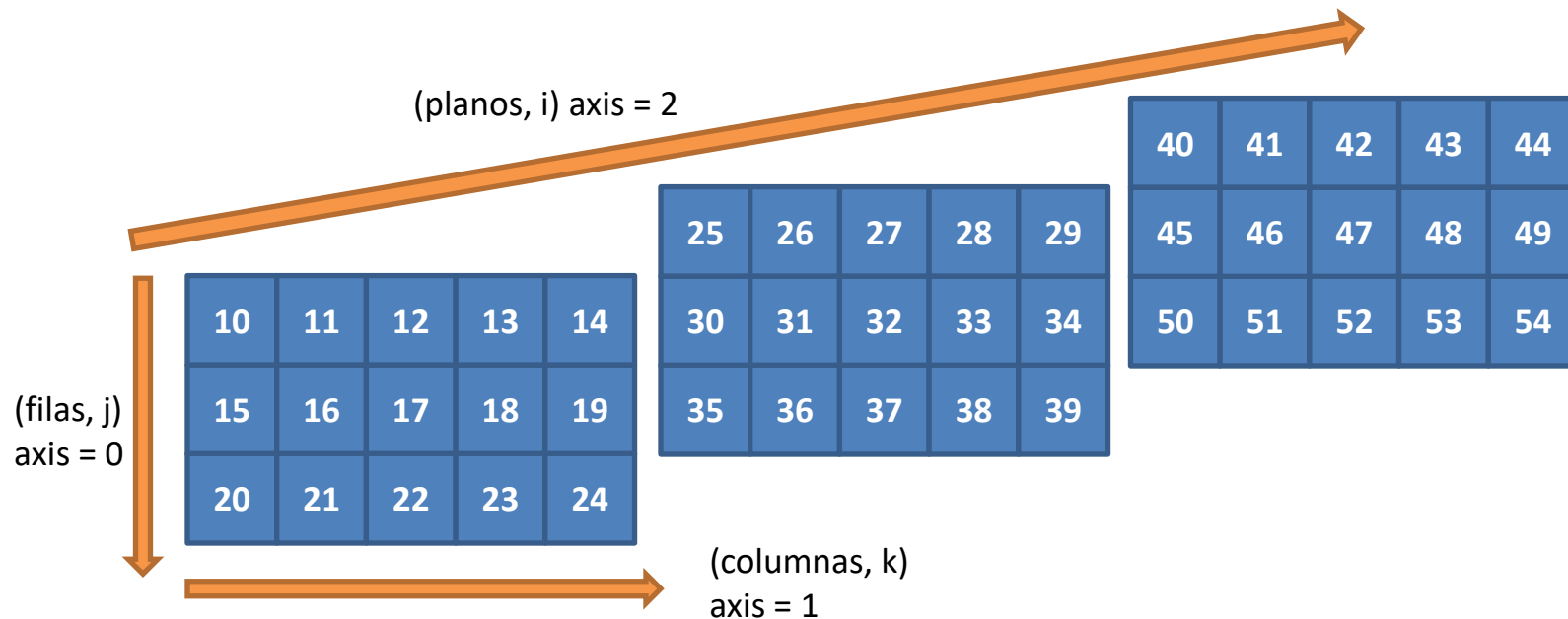


10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

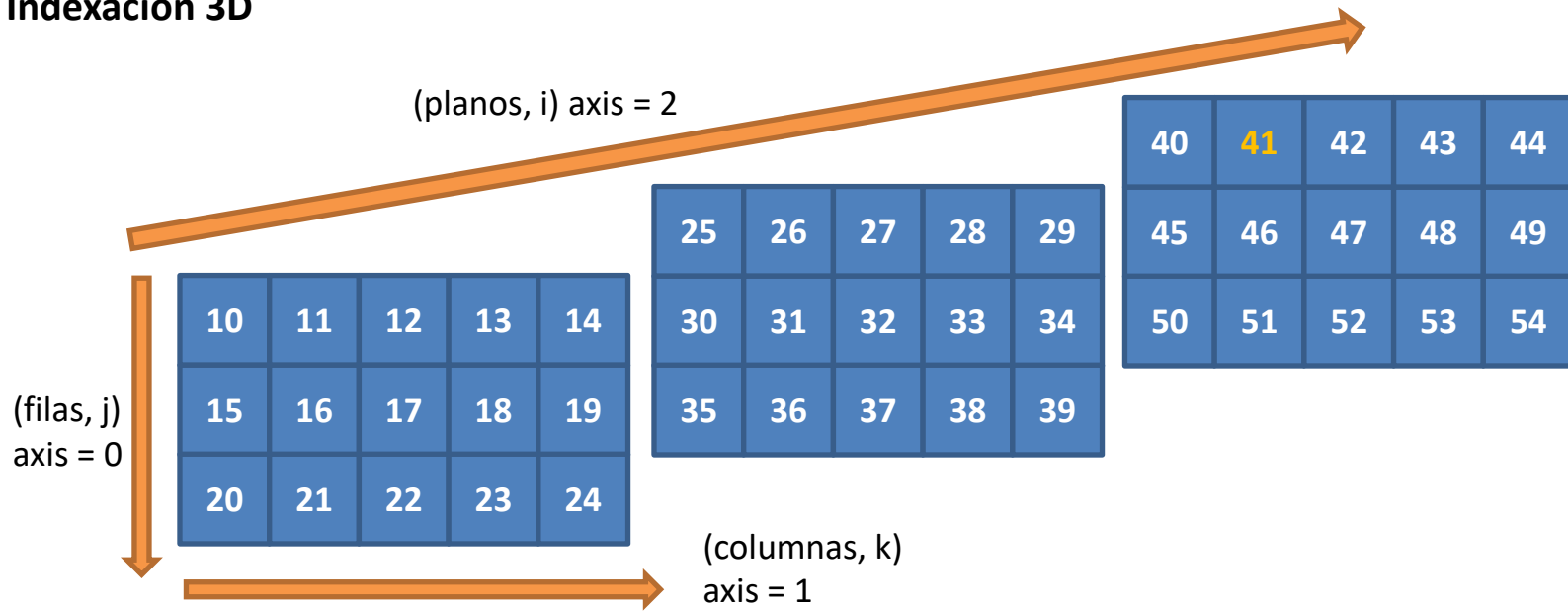
Una matriz 3D es como una pila de matrices:

- 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 un poco para que podamos ver los valores:



Indexación 3D



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

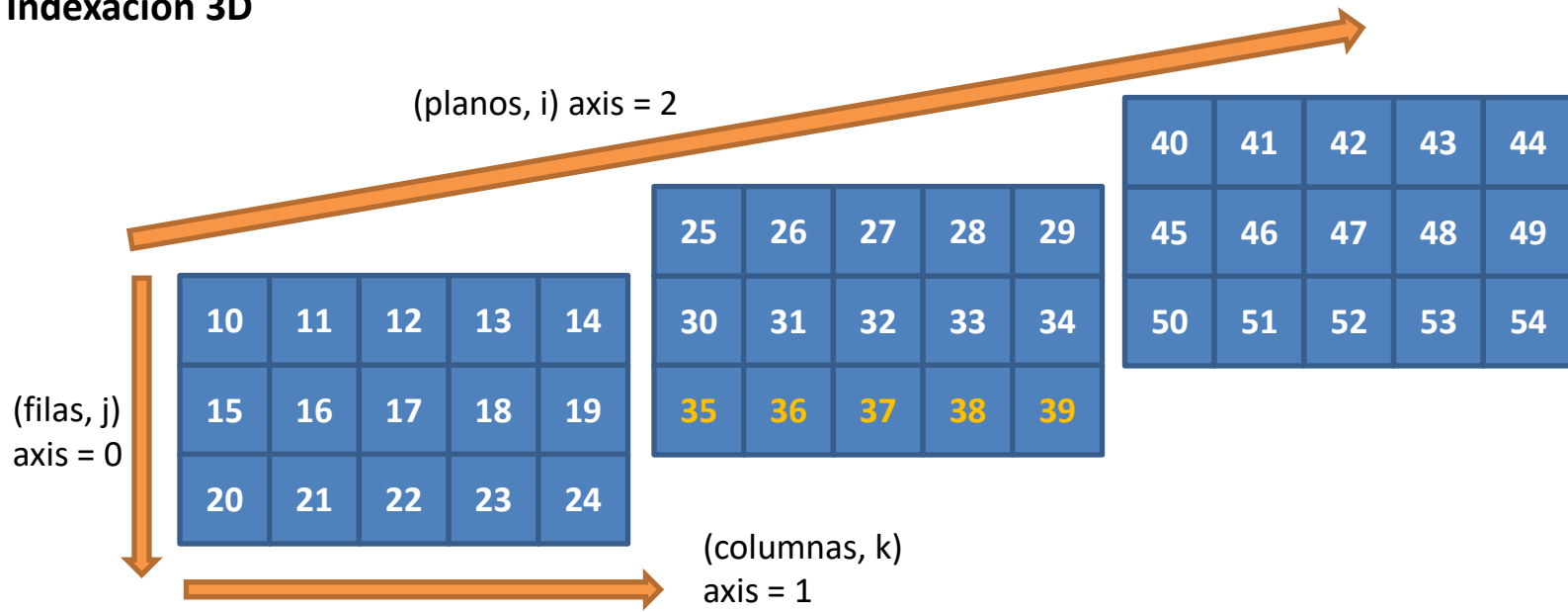
```
import numpy as np
```

```
dim3 = 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]]])  
print(dim3[2,0,1])
```

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

```
>>> import numpy as np  
>>>  
>>> dim3 = 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]]])  
>>>  
>>> print(dim3[2,0,1])  
41
```

Indexación 3D



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. 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:

```
import numpy as np
```

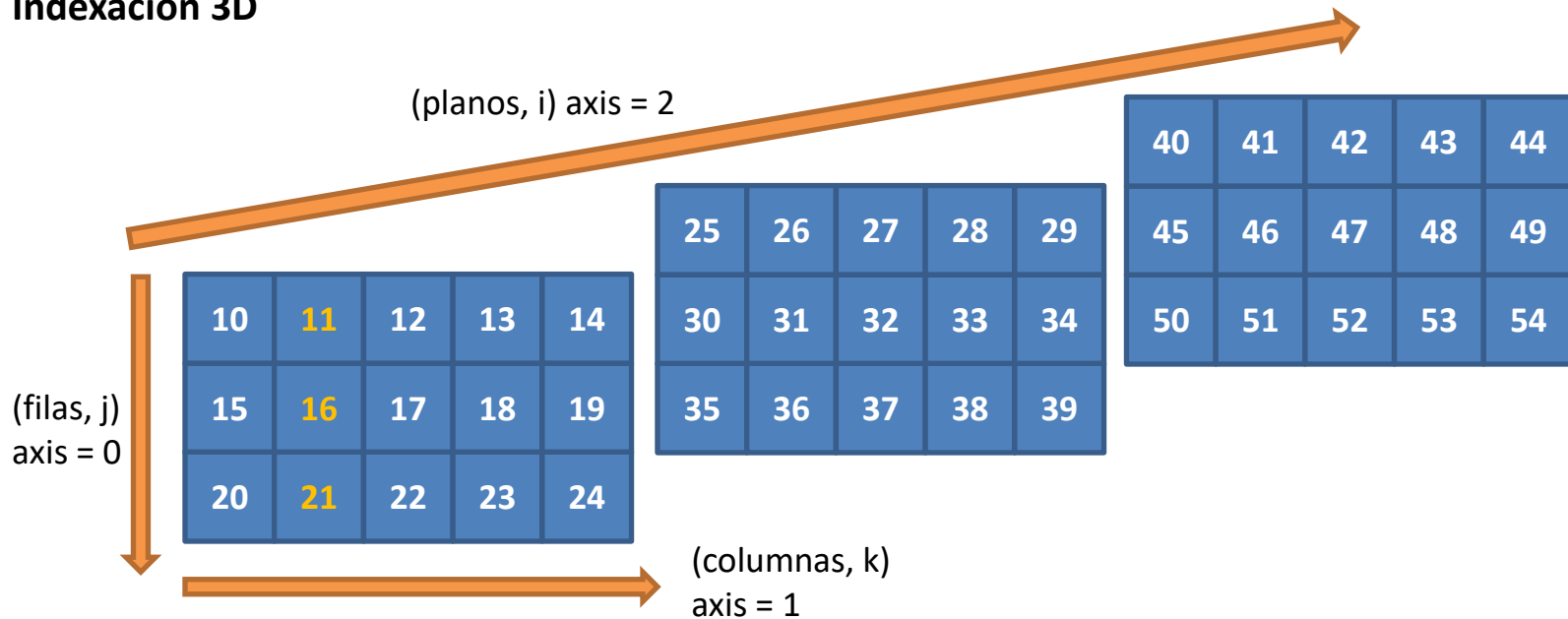
```
dim3 = np.array([[[10,11,12,13,14], [15,16,17,18,19], [20,21,22,23,24]],  
                [[25, 26, 27,28,29], [30,31,32,33,34], [35,36,37,38,39]],  
                [[40,41,42,43,44],[45,46,47,48,49], [50,51,52,53,54]]])  
print(dim3)
```

```
print(dim3[1, 2])
```

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

```
>>> import numpy as np  
>>>  
>>> dim3 = 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]]])  
>>>  
>>> print(dim3)  
[[[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]]]  
>>> print(dim3[1, 2])  
[35 36 37 38 39]
```


Indexación 3D



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

```
import numpy as np
dim3 = 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]]])
print(dim3)

print(dim3[0, :, 1])
```

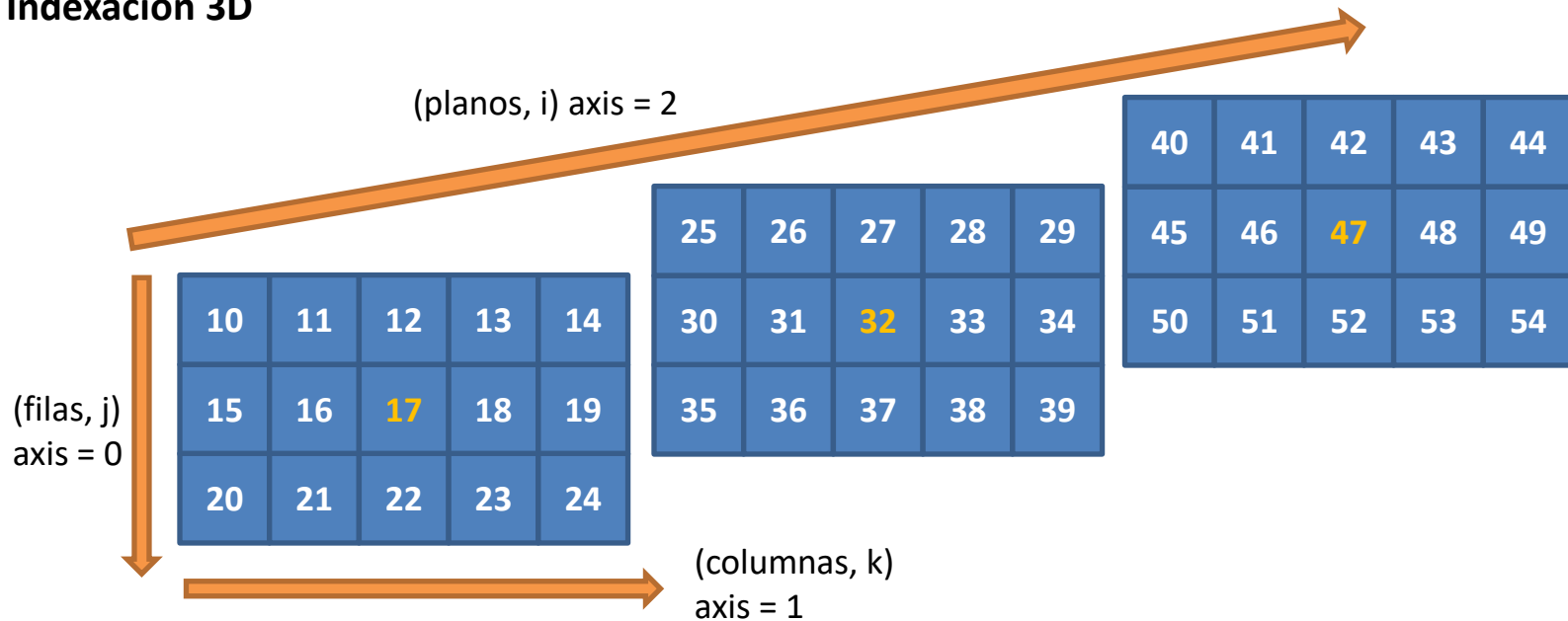
Esto selecciona el índice del plano de matriz 0, todas las filas de la columna 1, dando los valores 11, 16, 21

```
>>> import numpy as np
>>>
>>> dim3 = 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]]])
>>>
>>> print(dim3)
[[[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]]]
>>> print(dim3[0, :, 1])
[11 16 21]
```

Indexación 3D



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. 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:

```
import numpy as np
```

```
dim3 = 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]]])
```

```
print(dim3)
```

```
print(dim3[:, 1, 2])
```

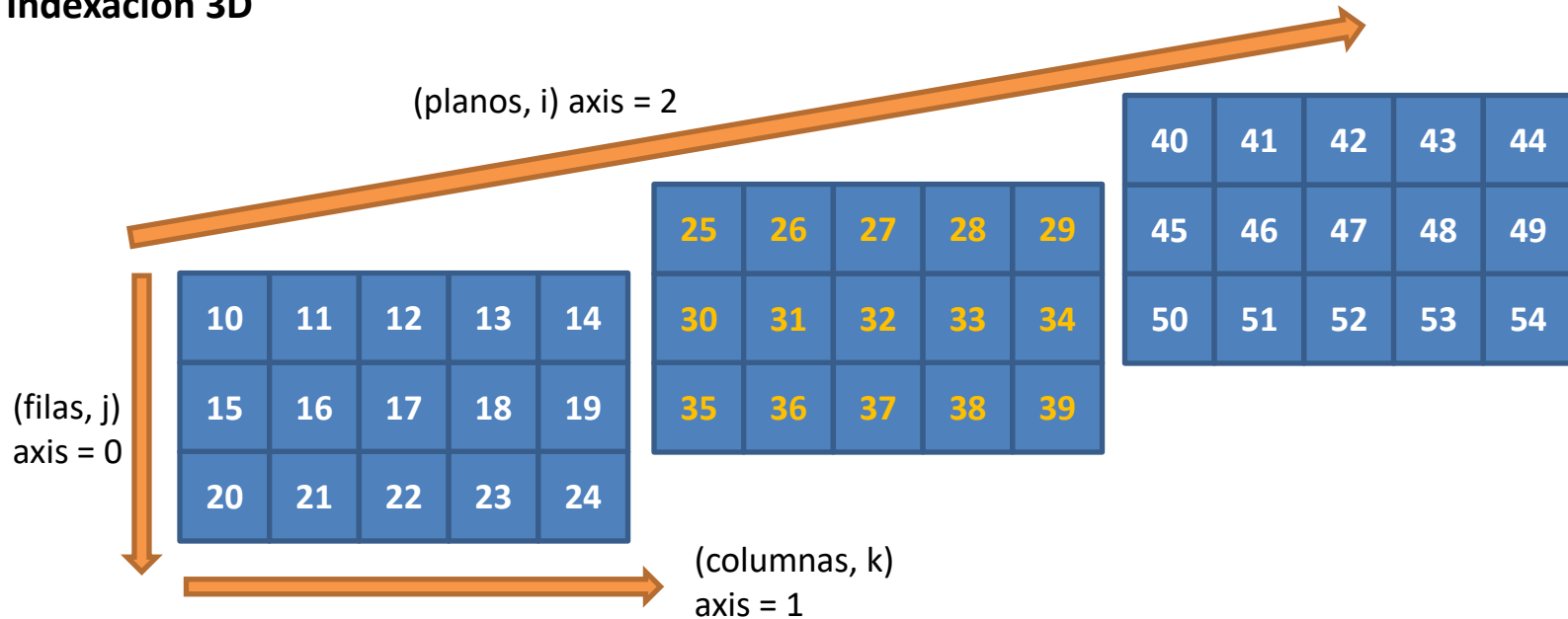
Esto selecciona todos los planos de la matriz, filas 1, columnas 2, dando los valores 17,32,47

```
>>> import numpy as np
>>>
>>> dim3 = 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]]])
>>> print(dim3)
[[[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]]]
>>> print(dim3[:, 1, 2])
[17 32 47]
```

Indexación 3D



Elegir una matriz en una matriz 3D

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

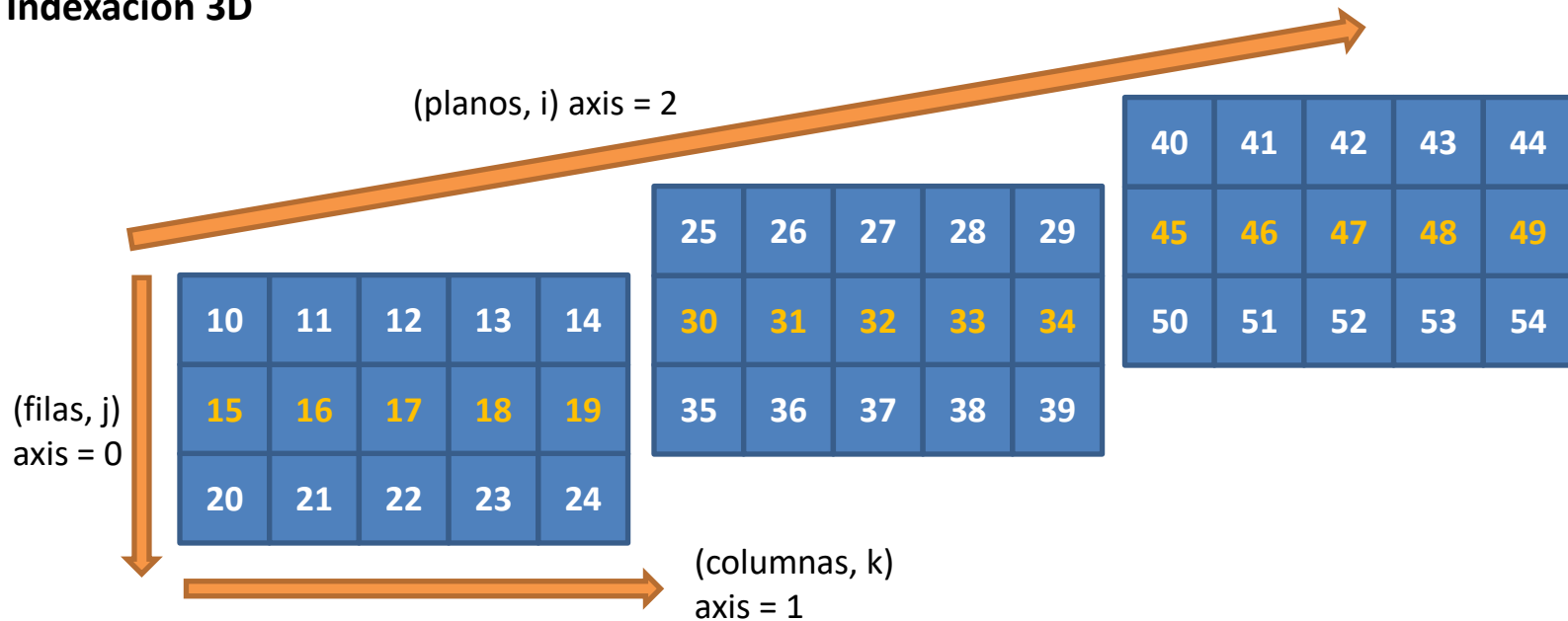
```
import numpy as np
dim3 = np.array([[[10,11,12,13,14], [15,16,17,18,19], [20,21,22,23,24]],
                 [[25, 26, 27,28,29], [30,31,32,33,34], [35,36,37,38,39]],
                 [[40,41,42,43,44],[45,46,47,48,49], [50,51,52,53,54]]])
print(dim3)
print(dim3[1])
```

```
>>> import numpy as np
>>>
>>> dim3 = 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]]])
>>>
>>> print(dim3)
[[[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]]]
>>> print(dim3[1])
[[25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]]
```

Indexación 3D



Elegir una matriz en una matriz 3D

Caso 2: 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:

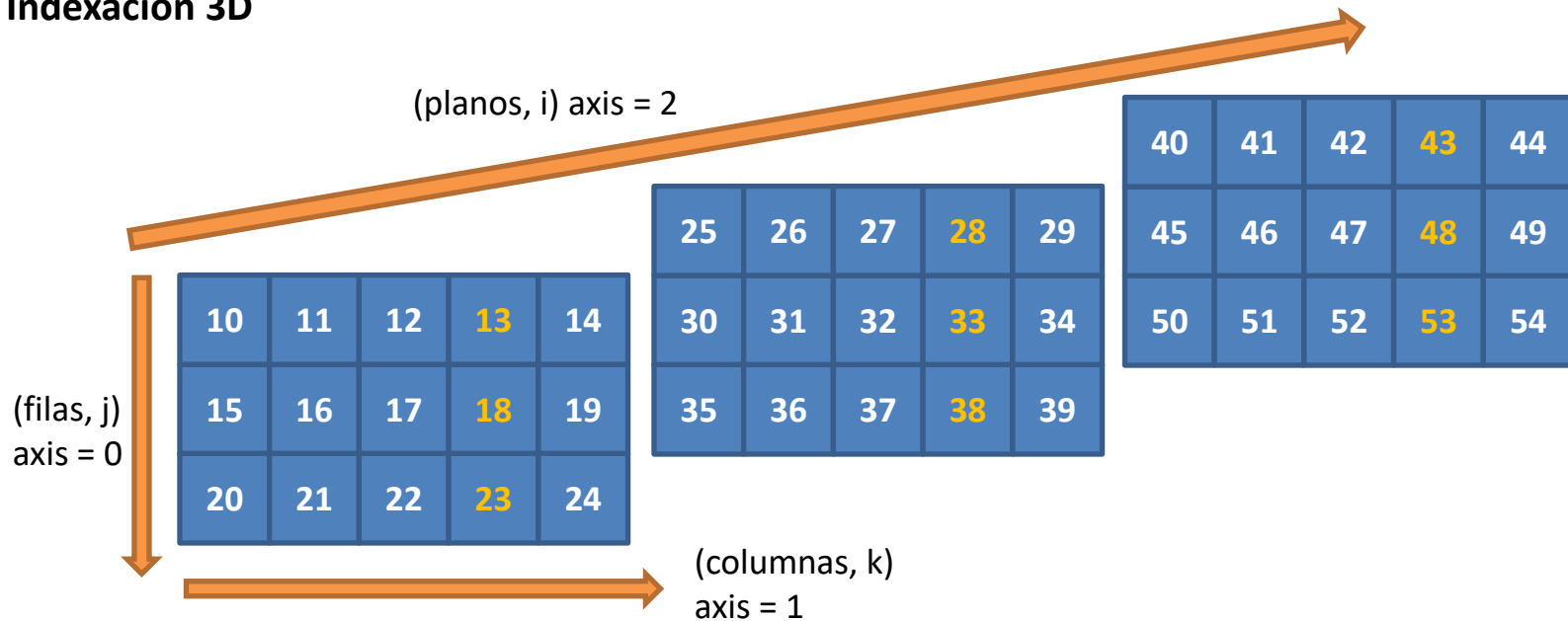
```
import numpy as np
dim3 = np.array([[[10,11,12,13,14], [15,16,17,18,19], [20,21,22,23,24]],
                [[25, 26, 27,28,29], [30,31,32,33,34], [35,36,37,38,39]],
                [[40,41,42,43,44],[45,46,47,48,49], [50,51,52,53,54]]])
print(dim3)
print(dim3[:, 1])
```

```
>>> import numpy as np
>>>
>>> dim3 = 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]]])
>>>
>>> print(dim3)
[[[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]]]
>>> print(dim3[:, 1])
[[15 16 17 18 19]
 [30 31 32 33 34]
 [45 46 47 48 49]]
```

Indexación 3D



Elegir una matriz en una matriz 3D

Caso 3: 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:

```
import numpy as np
dim3 = 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]]])
print(dim3)
print(dim3[:, :, 3])
```

```
>>> import numpy as np
>>>
>>> dim3 = 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]]])
>>>
>>> print(dim3)
[[[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]]]
>>> print(dim3[:, :, 3])
[[13 18 23]
 [28 33 38]
 [43 48 53]]
```

Cortes de listas. Resumen:

Cómo funciona la división con listas normales de Python? Supongamos que tenemos una lista:

```
a = [ 10 , 11 , 12 , 13 , 14 ]
```

Podemos usar la división para tomar una sublista, como esta:

```
b = a [ 1 : 4 ] # [11, 12, 13]
```

La notación de sector especifica un valor inicial y final **[inicio: final]** y copia la lista desde el principio hasta el final, pero sin incluirlo.

Podemos omitir el inicio, en cuyo caso el segmento comienza al principio de la lista.

Podemos omitir el final, por lo que el segmento continúa hasta el final de la lista.

Si omitimos ambos, el segmento creado es una copia de la lista completa:

```
c = a[:3] # [10, 11, 12]
```

```
d = a[2:] # [12, 13, 14]
```

```
e = a[:] # [10, 11, 12, 13, 14]
```

Hay que tener en cuenta la diferencia entre un índice y un segmento de longitud 1:

```
f = a[2]    # 12
```

```
g = a[2:3] # [12]
```

El índice devuelve un elemento de la matriz, el segmento devuelve una lista de un elemento.

Slicing 1D en arrays de numpy

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

```
import numpy as np
dim1 = np.array([1, 2, 3, 4, 5])
corte_dim1 = dim1[1:4]
print(corte_dim1)
```

1	2	3	4	5
0	1	2	3	4
1	2	3	4	5
0	1	2	3	4

```
>>> import numpy as np
>>> dim1 = np.array([1, 2, 3, 4, 5])
>>> corte_dim1 = dim1[1:4]
>>> print(corte_dim1) # [2, 3, 4]
[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):

```
import numpy as np

dim1 = np.array([1, 2, 3, 4, 5])
corte_dim1 = dim1[1:4]
print(corte_dim1)
```

```
corte_dim1[1] = 99
print(corte_dim1)
print(dim1)
```

1	2	3	4	5
0	1	2	3	4
1	2	99	4	5
0	1	2	3	4
1	2	99	4	5
0	1	2	3	4

```
>>> import numpy as np
>>> dim1 = np.array([1, 2, 3, 4, 5])
>>> corte_dim1 = dim1[1:4]
>>> print(corte_dim1)
[2 3 4]
>>> corte_dim1[1] = 99
>>> print(corte_dim1)
[ 2 99  4]
>>> print(dim1)
[ 1  2 99  4  5]
```

Slicing 2D en arrays de numpy

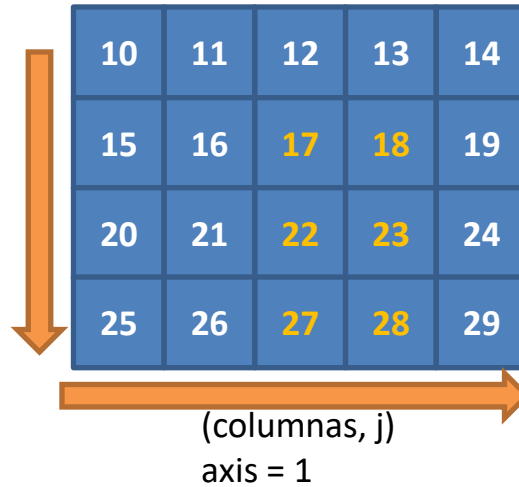
Se puede cortar una matriz 2D en ambos ejes para obtener un subconjunto rectangular de la matriz original. Por ejemplo:

```
import numpy as np
```

```
dim2 = np.array([[10, 11, 12, 13, 14],  
                [15, 16, 17, 18, 19],  
                [20, 21, 22, 23, 24],  
                [25, 26, 27, 28, 29]])
```

(filas, i)
axis=0

```
print(dim2[1:,2:4])
```



```
>>> import numpy as np  
>>>  
>>> dim2 = np.array([[10, 11, 12, 13, 14],  
...                 [15, 16, 17, 18, 19],  
...                 [20, 21, 22, 23, 24],  
...                 [25, 26, 27, 28, 29]])  
>>>  
>>> print(dim2[1:,2:4])  
[[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 : para seleccionar todos los planos, columnas o filas. Sin embargo, para los índices finales, simplemente omitir el índice y cuenta como un segmento completo. Entonces, para matrices 2D:

```
dim2[1:3,:]  
Es lo mismo que:  
dim2[1:3]
```

```
>>> print(dim2[1:3,:]) # es lo mismo que:  
[[15 16 17 18 19]  
 [20 21 22 23 24]]  
>>> print(dim2[1:3])  
[[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):

```
print(dim2[1,2:4])  
print(dim2[1:2,2:4])
```

```
>>> print(dim2[1,2:4])  
[17 18]  
>>> print(dim2[1:2,2:4])  
[[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:

```
import numpy as np
```

```
dim3 = 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]]])
```

```
print(dim3)  
print(dim3[:2,1,:2])
```

Esto selecciona:

planos **2**: los 2 primeros planos

filas **1**: a partir de la fila 1, el resto

columnas **2**: las 2 primeras columnas

```
>>> import numpy as np  
>>>  
>>> dim3 = 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]]])  
>>>  
>>> print(dim3)  
[[[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]]]  
>>> print(dim3[:2,1,:2])  
[[[15 16]  
  [20 21]]  
  
 [[30 31]  
  [35 36]]]
```

(planos, i) axis = 2

(filas, j)
axis = 0

(columnas, k)
axis = 1

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

```
print(dim3[1,:2,:]) # es lo mismo que:  
print(dim3[1,:2])  
# y  
print(dim3[1,:,:,]) # es lo mismo que:  
print(dim3[1,:,:]) # y también es lo mismo que:  
print(dim3[1,:])
```

Arrays de numpy y gráficos

Se pueden construir gráficos a partir de arrays de numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
#primer conjunto de datos
```

```
x1 = np.array([5,7,9])
```

```
y1 = np.array([8,10,7])
```

```
#segundo conjunto de datos
```

```
x2 = np.array([6,8,10])
```

```
y2 = np.array([15,7,9])
```

```
#barras para el primer conjunto
```

```
plt.bar(x1,y1, color='r',align='center')
```

```
#barras para el segundo conjunto
```

```
plt.bar(x2,y2, color='g',align='center')
```

```
#titulo
```

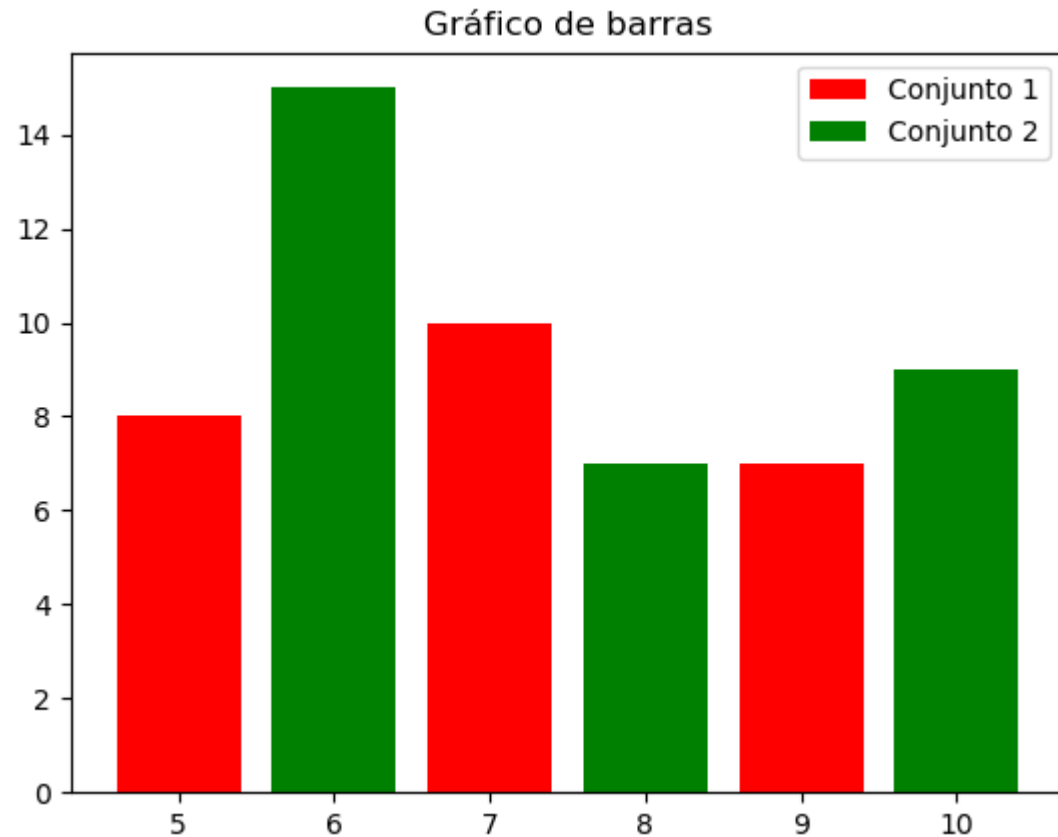
```
plt.title('Gráfico de barras')
```

```
#leyenda
```

```
plt.legend(['Conjunto 1', 'Conjunto 2'])
```

```
# emitir gráfica
```

```
plt.show()
```



Arrays de numpy y gráficos

```
import numpy as np
import matplotlib.pyplot as plt
```

```
#creamos la ventana
fig=plt.figure('Calificaciones')
#agregamos dos gráficas
grupo1=fig.add_subplot(211)
grupo2=fig.add_subplot(212)

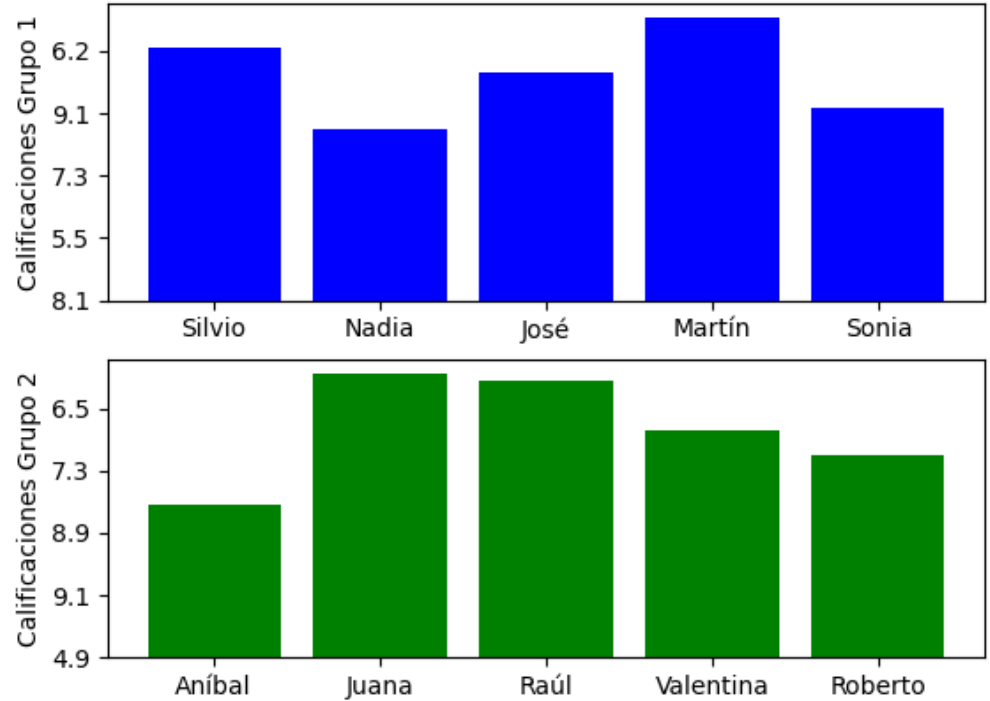
#obtenemos datos para el primer conjunto
alu_1 = ['Silvio', 'Nadia', 'José', 'Martín', 'Sonia']
calif_1 = np.array([8.1, 5.5, 7.3, 9.1, 6.2])

#obtenemos datos para el segundo conjunto
alu_2 = ['Aníbal', 'Juana', 'Raúl', 'Valentina', 'Roberto']
calif_2 = np.array([4.9, 9.1, 8.9, 7.3, 6.5])

#barras para el primer conjunto
grupo1.bar(alu_1, calif_1, color='b', align='center')
grupo1.set_xticks(alu_1)
grupo1.set_xticklabels(alu_1)
grupo1.set_yticklabels(calif_1)
grupo1.set_ylabel('Calificaciones Grupo 1')
# si queremos las etiquetas debajo debemos usar el método xlabel

#barras para el segundo conjunto
grupo2.bar(alu_2, calif_2, color='g', align='center')
grupo2.set_xticks(alu_2)
grupo2.set_xticklabels(alu_2)
grupo2.set_yticklabels(calif_2)
grupo2.set_ylabel('Calificaciones Grupo 2')

plt.show()
```



Arrays de numpy y gráficos 3D

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
#creamos la ventana
fig=plt.figure()
```

```
#agregamos plano 3D
plano=fig.add_subplot(111, projection='3d')
# plt.show()
#obtenemos datos para x, y, z
x=np.array([[1,2,3,4,5,6,7,8,9,10]])
y=np.array([[3,5,6,3,7,9,8,5,5,7]])
z=np.array([[2,7,3,1,4,6,4,5,6,7]])
```

```
#agregamos los arrays al plano
plano.plot_wireframe(x,y,z)
```

```
plano.set_xlabel('x-axis')
plano.set_ylabel('y-axis')
plano.set_zlabel('z-axis')
```

```
plt.title('Gráfica 3D')
plt.show()
```

