In [ ]:

*# COMPUTACIÓN BLANDA - Sistemas y Computación*  
*# -------------------------------------------*  
*# Introducción a numpy*  
*# -------------------------------------------*  
*# Lección 01*  
*#*  
*# \*\* Creación de arrays*  
*# \*\* Acceso a los arrays*  
*# \*\* Manejo de rangos*  
*# \*\* Modificación de arrays*  
*# Juan Sebastian Palacio Villada*  
*# Código: 1088351596*  
*# -------------------------------------------*

In [35]:

*# Se importa la librería numpy*  
  
**import** **numpy** **as** **np**  
*# Se crea una array con 10 elementos*  
a = np.arange(10)  
*# Se imprime en pantalla el contenido del array a*  
print('Arreglo a =', a, '**\n**')  
*# Se muestra el tipo de los elementos del array*  
print('Tipo de a =', a.dtype, '**\n**')  
*# Se calcula la dimensión del array a, en este caso dimensión = 1 (vector)*  
print('Dimensión de a =', a.ndim, '**\n**')  
*# Se calcula el número de elementos del array a*  
*# No olvidar que existe un elemento con índice 0*  
print('Número de elementos de a =', a.shape)

Arreglo a = [0 1 2 3 4 5 6 7 8 9]   
  
Tipo de a = int32   
  
Dimensión de a = 1   
  
Número de elementos de a = (10,)

In [40]:

*# Creando un arreglo multidimensional*  
*# La matriz se crea con la función: array de 10 X 5*  
m = np.array([np.arange(10), np.arange(10), np.arange(10), np.arange(10), np.arange(10)])  
print(m)

[[0 1 2 3 4 5 6 7 8 9]  
 [0 1 2 3 4 5 6 7 8 9]  
 [0 1 2 3 4 5 6 7 8 9]  
 [0 1 2 3 4 5 6 7 8 9]  
 [0 1 2 3 4 5 6 7 8 9]]

In [4]:

a = np.array([[1,2], [3,4]])  
print('a =**\n**', a, '**\n**')  
*# Elementos individuales*  
print('a[0,0] =', a[0,0], '**\n**')  
print('a[0,1] =', a[0,1], '**\n**')  
print('a[1,0] =', a[1,0], '**\n**')  
print('a[1,1] =', a[1,1])

a =  
 [[1 2]  
 [3 4]]   
  
a[0,0] = 1   
  
a[0,1] = 2   
  
a[1,0] = 3   
  
a[1,1] = 4

In [42]:

a = np.arange(10)  
print('a =', a, '**\n**')  
*# Muestra los elementos desde 0 hasta 10. Imprime desde 0 hasta 9*  
print('a[0:10] = ', a[0:10], '**\n**')  
*# Muestra desde 3 hasta 7. Imprime desde 3 hasta 7*  
print('a[3,8] =', a[3:8])

a = [0 1 2 3 4 5 6 7 8 9]   
  
a[0:10] = [0 1 2 3 4 5 6 7 8 9]   
  
a[3,7] = [3 4 5 6]

In [7]:

*# Mostrando todos los elementos, desde el 0 hasta el 8, de uno en uno*  
  
print('a[0:9:1] =', a[0:9:1], '**\n**')  
*# El mismo ejemplo, pero omitiendo el número 0 al principio, el cual no es necesario aquí*  
print('a[:9:1] =', a[:9:1], '**\n**')  
*# Mostrando los números, de dos en dos*  
print('a[0:9:2] =', a[0:9:2], '**\n**')  
*# Mostrando los números, de tres en tres*  
print('a[0:9:3] =', a[0:9:3])

a[0:9:1] = [0 1 2 3 4 5 6 7 8]   
  
a[:9:1] = [0 1 2 3 4 5 6 7 8]   
  
a[0:9:2] = [0 2 4 6 8]   
  
a[0:9:3] = [0 3 6]

In [8]:

print('a[9:0:-1] =', a[9:0:-1], '**\n**')  
*# Si se omiten los valores de índice, el resultado es preciso*  
print('a[::-1] =', a[::-1])

a[9:0:-1] = [8 7 6 5 4 3 2 1]   
  
a[::-1] = [8 7 6 5 4 3 2 1 0]

In [43]:

*# Utilización de arreglos multidimensionales*  
  
b = np.arange(40).reshape(2,4,5)  
print('b =**\n**', b)  
*# La instrucción reshape genera una matriz con 2 bloques, 4 filas y 5 columnas*  
*# El número total de elementos es de 40 (generados por arange)*

b =  
 [[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]

In [44]:

*# Acceso individual a los elementos del array*  
*# Elemento en el bloque 1, fila 2, columna 3*  
print('b[1,2,3] =', b[1,2,3], '**\n**')  
*# Elemento en el bloque 0, fila 2, columna 2*  
print('b[0,2,2] =', b[0,2,2], '**\n**')  
*# Elemento en el bloque 0, fila 1, columna 1*  
print('b[0,1,1] =', b[0,1,1])

b[1,2,3] = 33   
  
b[0,2,2] = 12   
  
b[0,1,1] = 6

In [46]:

*# Mostraremos cómo generalizar una selección*  
  
*# Primero elegimos el componente en la fila 0, columna 0, del bloque 0*  
print('b[0,0,0] =', b[0,0,0], '**\n**')  
*# A continuación, elegimos el componente en la fila 0, columna, pero del bloque 1*  
print('b[1,0,0] =', b[1,0,0], '**\n**')  
*# Para elegir SIMULTÁNEAMENTE ambos elementos, lo hacemos utilizando dos puntos*  
print('b[:,0,0] =', b[:,0,0])

b[0,0,0] = 0   
  
b[1,0,0] = 20   
  
b[:,0,0] = [ 0 20]

In [47]:

print('b[0] =**\n**', b[0])

b[0] =  
 [[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]

In [48]:

*# Otra forma de representar b[0] es: b[0, :, :]*  
  
*# Los dos puntos sin ningún valor, indican que se utilizarán todos los términos disponibles*  
*# En este caso, todas las filas y todas las columnas*  
print('b[0,:,:] =**\n**', b[0,:,:])

b[0,:,:] =  
 [[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]

In [49]:

*# Cuando se utiliza la notación de : a derecha o a izquierda, se puede reemplazar por ...*  
*# El ejemplo anterior se puede escribir así:*  
print('b[0, ...] =**\n**', b[0, ...])

b[0, ...] =  
 [[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]

In [50]:

*# Si queremos la fila 1 en el bloque 0 (sin que importen las columnas), se tiene:*  
print('b[0,1] =', b[0,1])

b[0,1] = [5 6 7 8 9]

In [51]:

*# El resultado de una selección puede utilizar luego para un cálculo posterior*  
  
*# Se obtiene la fila 1 del bloque 0 (como en ejemplo anterior)*  
*# y se asigna dicha respuesta a la variable z*  
z = b[0,1]  
print('z =', z, '**\n**')  
*# En este caso, la variable z toma el valor: [4 5 6 7]*  
*# Si ahora queremos tomar de dicha respuesta los valores de 2 en 2, se tiene:*  
print('z[::2] =', z[::2])

z = [5 6 7 8 9]   
  
z[::2] = [5 7 9]

In [52]:

*# El ejercicio anterior se puede combinar en una expresión única, así:*  
  
print('b[0,1,::2] =', b[0,1,::2])  
*# Esta es una solución más compacta*

b[0,1,::2] = [5 7 9]

In [53]:

*# Imprime todas las columnas, independientemente de los bloques y filas*  
  
print(b, '**\n**')  
print('b[:,:,1] =**\n**', b[:,:,1], '**\n**')  
*# Variante de notación (simplificada)*  
print('b[...,1] =**\n**', b[...,1])

[[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]   
  
b[:,:,1] =  
 [[ 1 6 11 16]  
 [21 26 31 36]]   
  
b[...,1] =  
 [[ 1 6 11 16]  
 [21 26 31 36]]

In [54]:

*# Si queremos seleccionar todas las filas 2, independientemente*  
  
*# de los bloques y columnas, se tiene:*  
print(b, '**\n**')  
print('b[:,1] =', b[:,1])  
*# Puesto que no se menciona en la notación las columnas, se toman todos*  
*# los valores según corresponda*

[[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]   
  
b[:,1] = [[ 5 6 7 8 9]  
 [25 26 27 28 29]]

In [55]:

*# En el siguiente ejemplo seleccionamos la columna 1 del bloque 0*  
  
print(b, '**\n**')  
print('b[0,:,1] =', b[0,:,1])

[[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]   
  
b[0,:,1] = [ 1 6 11 16]

In [56]:

*# Si queremos seleccionar la última columna del primer bloque, tenemos:*  
  
print('b[0,:,-1] =', b[0,:,-1])  
*# Podemos observar lo siguiente: entre corchetes encontramos tres valores*  
*# El primero, el cero, selecciona el primer bloque*  
*# El tercero, -1, se encarga de seleccionar la última columna*  
*# Los dos puntos, en la segunda posición, SELECCIONAN todos los*  
*# componentes de las FILAS, que FORMARÁN PARTE de dicha COLUMNA*  
*# Dado que los dos puntos definen todos los valores de las FILAS en*  
*# una columna específica, si quisiéramos que DICHOS VALORES estuvieran*  
*# en orden inverso, ejecutaremos la instrucción*  
print('b[0, ::-1, -1] =', b[0, ::-1, -1])  
*# La expresión ::-1 invierte todos los valores que se hubieran seleccionado*  
*# Si en lugar de invertir la columna, quisiéramos imprimir sus*  
*# valores de 2 en 2, tendríamos:*  
print('b[0, ::2, -1] =', b[0, ::2, -1])

b[0,:,-1] = [ 4 9 14 19]  
b[0, ::-1, -1] = [19 14 9 4]  
b[0, ::2, -1] = [ 4 14]

In [57]:

*# El array original*  
  
print(b, '**\n**-----------------------**\n**')  
*# Esta instrucción invierte los bloques*  
print(b[::-1])

[[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]   
-----------------------  
  
[[[20 21 22 23 24]  
 [25 26 27 28 29]  
 [30 31 32 33 34]  
 [35 36 37 38 39]]  
  
 [[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]]

In [58]:

*# La instrucción: ravel(), de-construye el efecto de la instrucción: reshape*  
  
*# Este es el array b en su estado matricial*  
print('Matriz b =**\n**', b, '**\n**--------------------------**\n**')  
*# Con ravel() se genera un vector a partir de la matriz*  
print('Vector b = **\n**', b.ravel())

Matriz b =  
 [[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]   
--------------------------  
  
Vector b =   
 [ 0 1 2 3 4 5 6 7 8 9 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]

In [59]:

*# La instrucción: flatten() es similar a ravel()*  
  
*# La diferencia es que flatten genera un nuevo espacio de memoria*  
print('Vector b con flatten =**\n**', b.flatten())

Vector b con flatten =  
 [ 0 1 2 3 4 5 6 7 8 9 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]

In [68]:

*# Se puede cambiar la estructura de una matriz con la instrucción: shape*  
  
*# Transformamos la matriz en 6 filas x 4 columnas*  
b.shape = (8,5)  
print('b(8x5) =**\n**', b)

b(8x5) =  
 [[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]

In [61]:

*# A partir de la matriz que acaba de ser generada, vamos a mostrar*  
  
*# cómo se construye la transpuesta de la matriz*  
*# Matriz original*  
print('b =**\n**', b, '**\n**------------------------**\n**')  
*# Matriz transpuesta*  
print('Transpuesta de b =**\n**', b.transpose(), '**\n**------------------------**\n**')

b =  
 [[[ 0 1 2 3 4]  
 [ 5 6 7 8 9]  
 [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]]]   
------------------------  
  
Transpuesta de b =  
 [[[ 0 20]  
 [ 5 25]  
 [10 30]  
 [15 35]]  
  
 [[ 1 21]  
 [ 6 26]  
 [11 31]  
 [16 36]]  
  
 [[ 2 22]  
 [ 7 27]  
 [12 32]  
 [17 37]]  
  
 [[ 3 23]  
 [ 8 28]  
 [13 33]  
 [18 38]]  
  
 [[ 4 24]  
 [ 9 29]  
 [14 34]  
 [19 39]]]   
------------------------

In [67]:

*# Para concluir este primer módulo de numpy, mostraremos que la instrucción*  
  
*# resize, ejecuta una labor similar a reshape*  
*# La diferencia está en que resize altera la estructura del array*  
*# En cambio reshape crea una copia del original, razón por la cual en*  
*# reshape se debe asignar el resultado a una nueva variable*  
*# Se cambia la estructura del array b*  
b.resize([4,10])  
*# Al imprimir el array b, se observa que su estructura ha cambiado*  
print('b =**\n**', b)

b =  
 [[ 0 1 2 3 4 5 6 7 8 9]  
 [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]]

In [ ]: