In [230]:
```
#PRESENTADO POR:angie paola villada ortiz
#codigo:1089721336



#COMPUTACIÓN BLANDA - Sistemas y Computación
# ----------------------------------------
# Introducción a numpy
# ----------------------------------------
# Lección 01
#
# ** Creación de arrays
# ** Acceso a los arrays
# ** Manejo de rangos
# ** Modificación de arrays
#
# ----------------------------------------




# Se importa la librería numpy
import numpy as np
# Se crea una array con 6 elementos
c = np.arange(9)
# Se imprime en pantalla el contenido del array a
print('Arreglo c =', c, '\n')
# Se muestra el tipo de los elementos del array
print('Tipo de c =', c.dtype, '\n')
# Se calcula la dimensión del array a, en este caso dimensión = 1 (vector)
print('Dimensión de c =', c.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 c =', c.shape)
```

```
Arreglo c = [0 1 2 3 4 5 6 7 8]

Tipo de c = int32

Dimensión de c = 1

Número de elementos de c = (9,)
```

In [231]:
```
# Creando un arreglo multidimensional
# La matriz se crea con la función: array
m = np.array([np.arange(3), np.arange(3)])
print(m)
```

```
[[0 1 2]
 [0 1 2]]
```

In [232]:
```
# Seleccionando elementos de un array
a = np.array([[1,2], [4,8]])
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]
 [4 8]]

a[0,0] = 1

a[0,1] = 2

a[1,0] = 4

a[1,1] = 8
```

In [233]:
```
# Crea un array con 9 elementos, desde 0 hasta 10
a = np.arange(10)
print('a =', a, )
# Muestra los elementos desde 0 hasta 9. Imprime desde 0 hasta 10
print('a[0:10] = ', a[0:10],)
# Muestra desde 1 hasta 10. Imprime desde 1 hasta 9
print('a[1,10] =', a[1:10])
```

```
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[1,10] = [1 2 3 4 5 6 7 8 9]
```

In [234]:
```
# Mostrando todos los elementos, desde el 0 hasta el 10, de uno en uno
print('a[0:10:1] =', a[0:10:1], )
# El mismo ejemplo, pero omitiendo el número 0 al principio, el cual no es nec
esario aquí
print('a[:10:1] =', a[:10:1], )
# Mostrando los números, de dos en dos
print('a[0:10:2] =', a[0:10:2], )
# Mostrando los números, de tres en tres
print('a[0:10:3] =', a[0:10:3])
```

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

In [235]:
```
# Si utilizamos un incremento negativo, el array se muestra en orden inverso
# El problema es que no muestra el valor 0
print('a[10:0:-1] =', a[10:0:-1], )
# Si se omiten los valores de índice, el resultado es preciso
print('a[::-1] =', a[::-1])
```

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

In [236]:
```
# Utilización de arreglos multidimensionales
b = np.arange(36).reshape(3,4,3)
print('b =\\n', b)
# La instrucción reshape genera una matriz con 3 bloques, 4 filas y 3 columnas
# El número total de elementos es de 36 (generados por arange)"
```

```
b =\n [[[ 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]]]
```

In [237]:
```
# Acceso individual a los elementos del array
# Elemento en el bloque 1, fila 2, columna 2
print('b[1,2,2] =', b[1,2,2], )
# Elemento en el bloque 0, fila 3, columna 1
print('b[0,3,1] =', b[0,3,1], )
# Elemento en el bloque 2, fila 1, columna 1
print('b[2,1,1] =', b[2,1,1])
```

```
b[1,2,2] = 20
b[0,3,1] = 10
b[2,1,1] = 28
```

In [238]:
```
# Mostraremos como 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], )
# A continuación, elegimos el componente en la fila 0, columna, pero del bloqu
e 1
print('b[1,0,0] =', b[1,0,0], )
# A continuación, elegimos el componente en la fila 0, columna, pero del bloqu
e 2
print('b[2,0,0] =', b[2,0,0], )

# Para elegir SIMULTANEAMENTE ambos elementos, lo hacemos utilizando dos punto
s
print('b[:,0,0] =', b[:,0,0])
```

```
b[0,0,0] = 0
b[1,0,0] = 12
b[2,0,0] = 24
b[:,0,0] = [ 0 12 24]
```

In [239]:
```
# Si escribimos: b[0]
# Habremos elegido el segundo bloque, pero habríamos omitido las filas y las c
olumnas
# En tal caso, numpy toma todas las filas y columnas del bloque 2
print('b[2] =\\n', b[2])
```

```
b[2] =\n [[24 25 26]
 [27 28 29]
 [30 31 32]
 [33 34 35]]
```

In [240]:
```
# Otra forma de representar b[2] es: b[2, :, :]
# Los dos puntos sin ningún valor, indican que se utilizarán todos los término
s disponibles
# En este caso, todas las filas y todas las columnas
print('b[2,:,:] =\\n', b[2,:,:])
```

```
b[2,:,:] =\n [[24 25 26]
 [27 28 29]
 [30 31 32]
 [33 34 35]]
```

In [241]:
```
# Cuando se utiliza la notación de : a derecha o a izquierda, se puede reempla
zar por ...
# El ejemplo anterior se puede escribir así:
print('b[2, ...] =\\n', b[2, ...])
```

```
b[2, ...] =\n [[24 25 26]
 [27 28 29]
 [30 31 32]
 [33 34 35]]
```

In [242]:
```
# Si queremos la fila 2 en el bloque 2 (sin que importen las columnas), se tie
ne:
print('b[2,2] =', b[2,2])
```

```
b[2,2] = [30 31 32]
```

In [243]:
```
# El resultado de una selección puede utilizar luego para un cálculo posterior
# Se obtiene la fila 2 del bloque 2 (como en ejemplo anterior)
# y se asigna dicha respuesta a la variable z
z = b[2,2]
print('z =', z, )
# En este caso, la variable z toma el valor: [30 31 32]
# Si ahora queremos tomar de dicha respuesta los valores de 2 en 2, se tiene:
print('z[::2] =', z[::2])
```

```
z = [30 31 32]
z[::2] = [30 32]
```

In [244]:
```
# El ejercicio anterior se puede combinar en una expresión única, así:
print('b[2,2,::2] =', b[2,2,::2])
# Esta es una solución más compacta
```

```
b[2,2,::2] = [30 32]
```

In [262]:
```
# Imprime todas las columnas, independientemente de los bloques y filas\n",
print(b, '\n')
print('b[:,:,0] =\n', b[:,:,0], '\n')
# Variante de notación (simplificada)\n",
print('b[...,0] =\n', b[...,0])
```

```
[[[ 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]]]

b[:,:,0] =
 [[ 0  3  6  9]
 [12 15 18 21]
 [24 27 30 33]]

b[...,0] =
 [[ 0  3  6  9]
 [12 15 18 21]
 [24 27 30 33]]
```

In [263]:
```
# Si queremos seleccionar todas las filas 3, independientemente
# de los bloques y columnas, se tiene:
print(b, '\n')
print('b[:,3] =', b[:,3])
# 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]]]

b[:,3] = [[ 9 10 11]
 [21 22 23]
 [33 34 35]]
```

In [264]:
```
 # En el siguiente ejemplo seleccionmos la columna 1 del bloque 1
print(b, '\n')
print('b[1,:,1] =', b[1,:,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]]]

b[1,:,1] = [13 16 19 22]
```

In [266]:
```
# Si queremos seleccionar la última columna del segundo bloque, tenemos:
print('b[1,:,-1] =', b[1,:,-1])
# Podemos observar lo siguiente: entre corchetes encontramos tres valores
# El primero, el cero, selecciona el segundo bloque
# El tercero, -1, se encarga de seleccionar la última columna
# Los dos puntos, en la segunda posición, SELECCIONAN todos los
# componentes de lad FILAS, que FORMARÁN PARTE de dicha COLUMNA
# Dado que los dos puntos definen todos los valores de las FILAS en
# una columna específica, si quisieramos que DICHOS VALORES estuvieran
# en orden inverso, ejecutaríamos la instrucción\n",
print('b[1, ::-1, -1] =', b[1, ::-1, -1])
# La expresión ::-1 invierte todos los valores que se hubieran seleccionado
# Si en lugar de invertir la columna, quisieramos imprimir sus
# valores de 2 en 2, tendríamos:
print('b[1, ::2, -1] =', b[1, ::2, -1])
```

```
b[1,:,-1] = [14 17 20 23]
b[1, ::-1, -1] = [23 20 17 14]
b[1, ::2, -1] = [14 20]
```

In [267]:
```
# El array original\n",
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]]]
----------------------

[[[24 25 26]
  [27 28 29]
  [30 31 32]
  [33 34 35]]

 [[12 13 14]
  [15 16 17]
  [18 19 20]
  [21 22 23]]

 [[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]]
```

In [268]:
```
# 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]]]
-------------------------

Vector b = \n [ 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]
```

In [269]:
```
# 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]
```

In [271]:
```
# Se puede cambiar la estructura de una matriz con la instrucción: shape
# Transformamos la matriz en 6 filas x 6 columnas
b.shape = (6,6)
print('b(6x6) =\n', b)
```

```
b(6x6) =
 [[ 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]]
```

In [277]:
```
# A partir de la matriz que acaba de ser generada, vamos a mostrar
# como se construye la transpuesta de la matriz
# Matriz original
print('b =\n', b, '\n------------------------\n')
# Matri 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]]
------------------------

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

In [278]:
```
# 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([6,6])
# 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]]
```

In [ ]: