In [1]:
```
# COMPUTACIÓN BLANDA - Sistemas y Computación
# ------------------------------------------
# Introducción a numpy
# ------------------------------------------
# Lección 01
# Natalia Rios Agudelo - 1088350269
```

In [2]:
```
# 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 [3]:
```
# Creando un arreglo multidimensional
# La matriz se crea con la función: array
m = np.array([np.arange(4), np.arange(4)])
print(m)
```

```
[[0 1 2 3]
 [0 1 2 3]]
```

In [4]:
```
# Seleccionando elementos de un array
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 [5]:
```
# Crea un array con 9 elementos, desde 0 hasta 8
a = np.arange(10)
print('a =', a, '\n')
# Muestra los elementos desde 0 hasta 9. Imprime desde 0 hasta 8
print('a[0:10] = ', a[0:10], '\n')
# Muestra desde 3 hasta 7. Imprime desde 3 hasta 6
print('a[5,10] =', a[5:10])
```

```
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[5,10] = [5 6 7 8 9]
```

In [6]:
```
# Crea un array con 9 elementos, desde 0 hasta 8
n = np.arange(15)
print('n =', n, '\n')
# Mostrando todos los elementos, desde el 0 hasta el 8, de uno en uno
print('n[0:15:1] =', n[0:15:1], '\n')
# Mostrando los números, de tres en tres
print('n[0:15:2] =',n[0:15:3], '\n')
# Mostrando los números, de cuatro en cuatro
print('n[0:15:5] =', n[0:15:5])
```

```
n = [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

n[0:15:1] = [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

n[0:15:2] = [ 0  3  6  9 12]

n[0:15:5] = [ 0  5 10]
```

In [7]:
```
# Si utilizamos un incremento negativo, el array se muestra en orden inverso
# El problema es que no muestra el valor 0
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])

print('a[:-1] =', a[:-1])
```

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

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

In [8]:
```
# Utilización de arreglos multidimensionales
b = np.arange(24).reshape(2,3,4)
print('b =\n', b)
# La instrucción reshape genera una matriz con 2 bloques, 3 filas y 4 columnas
# El número total de elementos es de 24 (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]]]
```

In [9]:
```
# 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] = 23

b[0,2,2] = 10

b[0,1,1] = 5
```

In [10]:
```
# 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], '\n')
# A continuación, elegimos el componente en la fila 0, columna, pero del bloqu
e 1
print('b[1,0,0] =', b[1,0,0], '\n')
# 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[:,0,0] = [ 0 12]
```

In [11]:
```
# Si escribimos: b[0]
# Habremos elegido el primer bloque, pero habríamos omitido las filas y las co
lumnas
# En tal caso, numpy toma todas las filas y columnas del bloque 0
print('b[0] =\n', b[0])
```

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

In [12]:
```
# Otra forma de representar b[0] es: b[0, :, :]
# 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[0,:,:] =\n', b[0,:,:])
```

```
b[0,:,:] =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [13]:
```
# 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[0, ...] =\n', b[0, ...])
```

```
b[0, ...] =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [14]:
```
# Si queremos la fila 1 en el bloque 0 (sin que importen las columnas), se tie
ne:
print('b[0,1] =', b[0,1])
```

```
b[0,1] = [4 5 6 7]
```

In [15]:
```
# 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 = [4 5 6 7]

z[::2] = [4 6]
```

In [16]:
```
# 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] = [4 6]
```

In [17]:
```
# 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]]]

b[:,:,1] =
 [[ 1  5  9]
 [13 17 21]]

b[...,1] =
 [[ 1  5  9]
 [13 17 21]]
```

In [18]:
```
# 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]]]

b[:,1] = [[ 4  5  6  7]
 [16 17 18 19]]
```

In [19]:
```
# En el siguiente ejemplo seleccionmos 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]]]

b[0,:,1] = [1 5 9]
```

In [20]:
```
# 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 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
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, quisieramos imprimir sus
# valores de 2 en 2, tendríamos:
print('b[0, ::2, -1] =', b[0, ::2, -1])
```

```
b[0,:,-1] = [ 3  7 11]
b[0, ::-1, -1] = [11  7  3]
b[0, ::2, -1] = [ 3 11]
```

In [21]:
```
# 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]]]
----------------------

[[[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 [22]:
```
# 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]]]
-------------------------

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]
```

In [23]:
```
# 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]
```

In [24]:
```
# Se puede cambiar la estructura de una matriz con la instrucción: shape
# Transformamos la matriz en 6 filas x 4 columnas
b.shape = (6,4)
print('b(6x4) =\n', b)
```

```
b(6x4) =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

In [25]:
```
# 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]]
------------------------

Transpuesta de b =
 [[ 0  4  8 12 16 20]
 [ 1  5  9 13 17 21]
 [ 2  6 10 14 18 22]
 [ 3  7 11 15 19 23]]
------------------------
```

In [26]:
```
# 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([2,12])
# 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]]
```

In [ ]:
```

```