In [ ]:

```
#ALEJANDRA LÓPEZ OCAMPO

#Introducción a numpy

# * Creación de arrays
# * Acceso a los arrays
# * Manejo de rangos
# * Modificación de arrays
```

In [1]:

```
# Se importa la librería numpy como np
import numpy as np

# Se crea un array con 6 elementos
a = np.arange(6)

# Se imprime en pantalla el contenido del array a
print('Arreglo a =', a, '\n')

#Tipo de elementos
print('Tipo de a =', a.dtype, '\n')

# Dimensión 1= (vector)
print('Dimensión de a =', a.ndim, '\n')

# Se calcula el número de elementos del array con index 0
print('Número de elementos de a =', a.shape)
```

```
Arreglo a = [0 1 2 3 4 5]

Tipo de a = int32

Dimensión de a = 1

Número de elementos de a = (6,)
```

In [2]:

```
#Para crear una matriz (Arreglo multidimendsional)
m = np.array([np.arange(2), np.arange(2)])
print(m)
```

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

In [3]:

```
#Creando otra matriz con valores preestablecidos
a = np.array([[1,2], [3,4]])
print('a =\n', a, '\n')

#Para imprimir todos los elementos individualmente
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 [9]:

```
# Crea un array con 16 elementos, desde 0 hasta 15
a = np.arange(16)
print('a =', a, '\n')
# Imprime en un rango de elementos
print('a[0:16] = ', a[0:16], '\n')
# Muestra desde 3 hasta 7. Imprime desde 3 hasta 6
print('a[3,7] =', a[3:7])
```

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

a[0:16] =  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]

a[3,7] = [3 4 5 6]
```

In [11]:

```
# Mostrando todos los elementos, desde el 0 hasta el 15, de uno en uno
print('a[0:16:1] =', a[0:16:1], '\n')
# El mismo ejemplo, pero omitiendo el número 0 al principio, el cual no es necesario aquí
print('a[:16:1] =', a[:16:1], '\n')
# Mostrando los números, de dos en dos
print('a[0:16:2] =', a[0:16:2], '\n')
# Mostrando los números, de tres en tres
print('a[0:16:3] =', a[0:16:3])
```

```
a[0:16:1] = [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]

a[:16:1] = [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]

a[0:16:2] = [ 0  2  4  6  8 10 12 14]

a[0:16:3] = [ 0  3  6  9 12 15]
```

In [12]:

```
# Si utilizamos un incremento negativo, el array se muestra en orden inverso

# El problema es que no muestra el valor 0
print('a[16:0:-1] =', a[16:0:-1], '\n')
# Si se omiten los valores de índice, el resultado es preciso
print('a[::-1] =', a[::-1])
```

```
a[16:0:-1] = [15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]

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

In [13]:

```
# Utilización de arreglos multidimensionales

b = np.arange(24).reshape(2,3,4)

#Reshape genera una matriz (2 bloques, 3 filas, 4 columnas)
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 [14]:

```
#Acceso [bloque, fila, columna]
# 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 [17]:

```
# 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 bloque 1
print('b[1,0,0] =', b[1,0,0], '\n')
# Para elegir SIMULTANEAMENTE ambos elementos, lo hacemos utilizando dos puntos
print('b[:,0,0] =', b[:,0,0])
```

```
b[0,0,0] = 0

b[1,0,0] = 12

b[:,0,0] = [ 0 12]
```

In [19]:

```
# Si escribimos: b[0]

# Habremos elegido el primer bloque, pero habríamos omitido las filas y las columnas
# 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 [21]:

```
# 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]]
```

In [23]:

```
# 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]]
```

In [24]:

```
# 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] = [4 5 6 7]
```

In [25]:

```
# 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 [26]:

```
# 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 [27]:

```
# 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 [28]:

```
# 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
```

```
# 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 [29]:

```
# 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 [30]:

```
# 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 [31]:

```
# 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 [32]:

```
# 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 [33]:

```
# 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 [34]:

```
# El array original

print(b, '\n-----------------------\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]]]
-----------------------

[[[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 [36]:

```
# 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 [ ]:

```

```