In [1]:
```
#COMPUTACIÓN BLANDA | ING DE SISTEMAS Y COMPUTACIÓN UTP CUBA
#JULIAN GIRALDO CARDONA 1004752912
#MACHINE LEARNING 01
```

In [2]:
```
#Se importa libreria numpy
import numpy as np
#Se crea un array con 8 elementos
a=np.arange(8)
#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]

Tipo de a = int32

Dimensión de a = 1

Número de elementos de a = (8,)
```

In [3]:
```
#Creando un arreglo multidimensional
#La matríz 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([[0,1,2,3,4], [5,6,7,8,9]])
print('a =\n', a, '\n')
# Elementos individuales
print('a[0,0] =', a[0,0], '\n')
print('a[0,2] =', a[0,2], '\n')
print('a[1,3] =', a[1,3], '\n')
print('a[0,1] =', a[0,1], '\n')
print('a[1,1] =', a[1,1])
```

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

a[0,0] = 0

a[0,2] = 2

a[1,3] = 8

a[0,1] = 1

a[1,1] = 6
```

In [5]:
```
#Crea un array con 10 elementos, desde 0 hasta 9

a = np.arange(10)
print('a =', a, '\n')
#Muestra los elementos desde 0 hasta 9. Imprime desde 0 hasta 10
print('a[0:10] = ', a[0:10], '\n')
# Muestra desde 2 hasta 7. Imprime desde 2 hasta 6
print('a[2,8] =', a[2:8])
```

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

In [6]:
```
#Mostrando todos los elementos, desde el 0 hasta el 9, de uno en uno

print('a[0:10:1] =', a[0:10:1], '\n')
#El mismo ejemplo, pero omitiendo el número 0 al principio, el cual no es nece
sario aquí
print('a[:10:1] =', a[:10:1], '\n')
#Mostrando los números, de dos en dos
print('a[0:10:2] =', a[0:9:2], '\n')
#Mostrando los números, de tres en tres
print('a[0:10:3] =', a[0:9:3])
```

```
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]
```

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[10:0:-1] =', a[10:0:-1], '\n')
# 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 [8]:
```
#Utilización de arreglos multidimensionales

b = np.arange(27).reshape(3,3,3)
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]
  [24 25 26]]]
```

In [9]:
```
#Acceso individual a los elementos del array
#Elemento en el bloque 1, fila 2, columna 2
print('b[1,2,3] =', b[1,2,2], '\n')
#Elemento en el bloque 0, fila 2, columna 2
print('b[0,2,2] =', b[0,2,2], '\n')
#Elemento en el bloque 2, fila 1, columna 1
print('b[0,1,1] =', b[2,1,1])
```

```
b[1,2,3] = 17

b[0,2,2] = 8

b[0,1,1] = 22
```

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 bloque
1
print('b[1,0,0] =', b[1,0,0], '\n')
#Luego, elegimos el componente en la fila 0, columna, pero del bloque 2
print('b[2,0,0] =', b[2,0,0], '\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] = 9

b[2,0,0] = 18

b[:,0,0] = [ 0  9 18]
```

In [11]:
```
#Si escribimos: b[1]

#Habremos elegido el segundo bloque, pero habríamos omitido las filas y las co
lumnas
#En tal caso, numpy toma todas las filas y columnas del bloque 1
print('b[1] =\n', b[1])
```

```
b[1] =
 [[ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

In [12]:
```
#Otra forma de representar b[1] es: b[1, :, :]

#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[1,:,:] =\n', b[1,:,:])
```

```
b[1,:,:] =
 [[ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

In [13]:
```
# Cuando se utiliza la notación de : a derecha o a izquierda, se puede reemplazar por ...
# El ejemplo anterior se puede escribir así:
print('b[1, ...] =\n', b[1, ...])
```

```
b[1, ...] =
 [[ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

In [14]:
```
#Si queremos la fila 1 en el bloque 2 (sin que importen las columnas), se tiene:
print(b, '\n')
print('b[2,0] =', b[2,0])
```

```
[[[ 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]]]

b[2,0] = [18 19 20]
```

In [15]:
```
#El resultado de una selección puede utilizar luego para un cálculo posterior

#Se obtiene la fila 2 del bloque 1 (como en ejemplo anterior)
#y se asigna dicha respuesta a la variable z
z = b[1,2]
print('z =', z, '\n')
#En este caso, la variable z toma el valor: [15 16 17]
#Si ahora queremos tomar de dicha respuesta los valores de 3 en 3, se tiene:
print('z[::3] =', z[::3])
```

```
z = [15 16 17]

z[::3] = [15]
```

In [16]:
```
#El ejercicio anterior se puede combinar en una expresión única, así:

print('b[1,2,::3] =', b[1,2,::3])
#Esta es una solución más compacta
```

```
b[1,2,::3] = [15]
```

In [17]:
```
#Imprime todas las columnas, independientemente de los bloques y filas

print(b, '\n')
print('b[:,:,0] =\n', b[:,:,0], '\n')
#Variante de notación (simplificada)
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]]] 

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

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

In [18]:
```
# Si queremos seleccionar todas las filas 3, independientemente

# de los bloques y columnas, se tiene:
print(b, '\n')
print('b[:,2] =', b[:,2])
# 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]]]

b[:,2] = [[ 6  7  8]
 [15 16 17]
 [24 25 26]]
```

In [19]:
```
# En el siguiente ejemplo seleccionmos la columna 2 del bloque 2

print(b, '\n')
print('b[2,:,2] =', b[2,:,2])
```

```
[[[ 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]]]

b[2,:,2] = [20 23 26]
```

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] = [2 5 8]
b[0, ::-1, -1] = [8 5 2]
b[0, ::2, -1] = [2 8]
```

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]
  [24 25 26]]]
-----------------------

[[[18 19 20]
  [21 22 23]
  [24 25 26]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

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

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]
  [24 25 26]]]
-------------------------

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]
```

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
 24 25 26]
```

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 = (9,3)
print('b(9x3) =\n', b)
```

```
b(9x3) =
 [[ 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]]
```

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]
 [24 25 26]]
------------------------

Transpuesta de b =
 [[ 0  3  6  9 12 15 18 21 24]
 [ 1  4  7 10 13 16 19 22 25]
 [ 2  5  8 11 14 17 20 23 26]]
------------------------
```

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([3,9])
# 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]]
```