TrabajoMCL01

August 27, 2020

[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,)
```

[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]]
```

[4]:
```
# Seleccionando elementos de un array
a = np.array([[4,2], [10,4]])
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 =
 [[ 4  2]
 [10  4]]

a[0,0] = 4

a[0,1] = 2

a[1,0] = 10

a[1,1] = 4
```

[5]:
```
# Crea un array con 9 elementos, desde 0 hasta 8
a = np.arange(20)
print('a =', a, '\n')
# Imprime desde 0 hasta 19
print('a[0:20] = ', a[0:20], '\n')
#  Imprime desde 10 hasta 16
print('a[10,17] =', a[10:17])
# Mostrando todos los elementos de uno en uno
print('a[0:20:1] =', a[0:20:1], '\n')
print('a[:20:1] =', a[:20:1], '\n')
# Mostrando los números, de dos en dos
print('a[0:20:2] =', a[0:20:2], '\n')
# Mostrando los números, de tres en tres
print('a[0:20:3] =', a[0:20:3])
# Si utilizamos un incremento negativo, el array se muestra en orden inverso
# El problema es que no muestra el valor 0
print('a[20:0:-1] =', a[20:0:-1], '\n')
# Si se omiten los valores de índice, el resultado es preciso
print('a[::-1] =', a[::-1])
```

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

a[0:20] =  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

a[10,17] = [10 11 12 13 14 15 16]
a[0:20:1] = [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

a[:20:1] = [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

a[0:20:2] = [ 0  2  4  6  8 10 12 14 16 18]

a[0:20:3] = [ 0  3  6  9 12 15 18]
a[20:0:-1] = [19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]

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

[9]:
```
# Utilización de arreglos multidimensionales
b = np.arange(12).reshape(3,2,2)
print('b =\n', b)
# La instrucción reshape genera una matriz con 3 bloques, 2 filas y 2 columnas
# El número total de elementos es de 12 (generados por arange)
# Acceso individual a los elementos del array
# Elemento en el bloque 1, fila 1, columna 1
print('b[1,2,3] =', b[1,1,1], '\n')
# Elemento en el bloque 0, fila 1, columna 1
print('b[0,2,2] =', b[0,1,1], '\n')
# Elemento en el bloque 0, fila 0, columna 0
print('b[0,1,1] =', b[0,0,0])
# Para elegir SIMULTANEAMENTE ambos elementos, lo hacemos utilizando dos puntos
print('b[:,0,0] =', b[:,0,0])
#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])
# 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[1,:,:] =\n', b[1,:,:])
# 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, ...])
# Si queremos la fila 2 en el bloque 1 (sin que importen las columnas), se␣
 ↪tiene:
print('b[1,1] =', b[1,1])
# 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')
# Si ahora queremos tomar de dicha respuesta los valores de 2 en 2, se tiene:
print('z[::2] =', z[::2])
# 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
# Imprime todas las columnas, independientemente de los bloques y filas
print('b[:,:,1] =\n', b[:,:,1], '\n')
# Variante de notación (simplificada)
print('b[...,1] =\n', b[...,1])
# Si queremos seleccionar todas las filas 2, independientemente
# de los bloques y columnas, se tiene:
```

```
print('b[:,1] =', b[:,1])
# Puesto que no se menciona en la notación las columnas, se toman todos
# los valores según corresponda
# En el siguiente ejemplo seleccionmos la columna 0 del bloque 0
print('b[0,:,1] =', b[0,:,0])
# 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])
# El array original
print(b, '\n-----------------------\n')
# Esta instrucción invierte los bloques
print(b[::-1])
# 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())
# Se puede cambiar la estructura de una matriz con la instrucción: shape
# Transformamos la matriz en 6 filas x 4 columnas
b.shape = (3,4)
print('b(3x4) =\n', b)
# 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]]]
b[1,2,3] = 7

b[0,2,2] = 3

b[0,1,1] = 0
b[:,0,0] = [0 4 8]
b[0] =
 [[0 1]
 [2 3]]
b[1,:,:] =
 [[4 5]
 [6 7]]
b[1, …] =
 [[4 5]
 [6 7]]
b[1,1] = [6 7]
z = [2 3]

z[::2] = [2]
b[0,1,::2] = [2]
b[:,:,1] =
 [[ 1  3]
 [ 5  7]
 [ 9 11]]

b[…,1] =
 [[ 1  3]
 [ 5  7]
 [ 9 11]]
b[:,1] = [[ 2  3]
 [ 6  7]
 [10 11]]
b[0,:,1] = [0 2]
b[0,:,-1] = [1 3]
b[0, ::-1, -1] = [3 1]
b[0, ::2, -1] = [1]
[[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]]]
------------------------
```

```
[[[ 8  9]
  [10 11]]

 [[ 4  5]
  [ 6  7]]

 [[ 0  1]
  [ 2  3]]]
Matriz b =
 [[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]]]
--------------------------

Vector b =
 [ 0  1  2  3  4  5  6  7  8  9 10 11]
b(3x4) =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
b =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
------------------------

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

[ ]: