Teoría Básica Numpy

basic numpy theory

Autor: Sebastián Arroyave Ramírez

IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: s.arroyave@utp.edu.co

1. Se crea un array de 7 elementos, una línea mas abajo se imprime el tamaño que contiene el array, luego se imprime que tipo de elementos contiene este array, una línea debajo de esta se imprime como se calcula la dimensión del array y por ultimo se calcula el numero de elementos que contiene el array.

```
[2]: import numpy as np

a = np.arange(7)

print('Arreglo a =', a, '\n')

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

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

print('Número de elementos de a =', a.shape)
```

```
Arreglo a = [0 1 2 3 4 5 6]

Tipo de a = int64

Dimensión de a = 1

Número de elementos de a = (7,)
```

2. En estas líneas creamos lo que es un arreglo multidimensional y con esta función se crea la matriz: array

```
[4]: m = np.array([np.arange(4), np.arange(4)])
print(m)
```

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

3. Seleccionamos los elementos del array y luego imprimimos los elementos seleccionados, pero individualmente.

```
[6]: a = np.array([[2,3], [4,3]])
print('a =\n', a, '\n')

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 =
 [[2 3]
 [4 3]]

a[0,0] = 2

a[0,1] = 3

a[1,0] = 4

a[1,1] = 3
```

4. Creamos un nuevo array de 10 elementos, que va desde 0 a 9, luego imprime los elementos que van desde 0 a 10 y por último nos muestra los elementos del 0 a 9.

```
[6]: a = np.arange(10)

print('a =', a, '\n')

print('a[0:10] = ', a[0:10], '\n')

print('a[3,9] =', a[3:9])
```

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

5. En estas líneas lo que hacemos es mostrar los elementos del 0 al 9, luego los mostramos de 2 en 2 y luego tres en tres.

```
[8]: print('a[0:10:1] =', a[0:10:1], '\n')
     print('a[0:10:1] =', a[0:10:1], '\n')
     print('a[0:10:2] =', a[0:10:2], '\n')
     print('a[0:10:3] =', a[0:10:3])

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

a[0:10:1] = [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]
```

6. Si ponemos un -1 en la función lo que nos mostrara el array de forma inversa y si no se escriben los primeros números no pasara nada nos mostrara lo mismo.

```
[9]: print('a[10:0:-1] =', a[10:0:-1], '\n')
     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]
```

7. Lo que hacemos aquí es trabajar con los arreglos multidimensionales, haciendo un array con 2 bloques, 3 filas y 4 columnas, también 24 números arrojados por la función arange.

```
[15]: b = np.arange(24).reshape(2,3,4)
      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]]]
```

8. Aquí accesamos a los elementos individualmente y luego seleccionamos que bloque, fila y columna queremos mostrar.

```
[19]: print('b[0,2,2] =', b[0,2,2], '\n')
      print('b[0,1,1] =', b[0,1,1])

b[0,2,2] = 10

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

9. Aquí podemos mostrar como generalizamos la selección de los bloques, filas y columnas.

```
[20]: print('b[0,0,0] =', b[0,1,0], '\n')
      print('b[1,0,0] =', b[1,0,0], '\n')
      print('b[:,0,0] =', b[0:,0,1])

b[0,0,0] = 4

b[1,0,0] = 12

b[:,0,0] = [ 1 13]
```

10. Aquí cuando ponemos b[1], lo que haremos es que seleccionamos el segundo bloque pero numpy al ver que no pusimos ni filas o columnas, solo coger el segundo bloque.

```
[22]: print('b[1] =\n', b[1])

b[1] =
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

11. En este caso se utilizaran todas las filas y columnas.

```
[24]: print('b[1,:,:] =\n', b[1,:,:])

b[1,:,:] =
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

12. Aquí lo que hacemos es remplazar la notación : por los puntos ya que nos mostrara el mismo resultado.

```
[25]: print('b[1, ...] =\n', b[1, ...])
```

```
b[1, ...] =
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

13. En esta parte la fila 0 y el bloque 1 nos lo mostrar sin importar las columnas.

```
[27]: print('b[1,0] =', b[1,0])
```

```
b[1,0] = [12 13 14 15]
```

14. Aquí obtenemos la fila 0 del bloque 1 y lo vamos mostrando de 2 en 2.

```
[29]: z = b[1,0]

      print('z =', z, '\n')

      print('z[::2] =', z[::2])
```

```
z = [12 13 14 15]

z[::2] = [12 14]
```

15. En este paso, se muestra el mismo resultado del anterior paso, pero con otra notación.

```
[30]: print('b[1,0,::2] =', b[1,0,::2])
```

```
b[1,0,::2] = [12 14]
```

16. Aquí mostramos solo las columnas.

```
[31]: print(b, '\n')
      print('b[:,:,0] =\n', b[:,:,0], '\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]]]

b[:,:,0] =
 [[ 0  4  8]
 [12 16 20]]

b[...,0] =
 [[ 0  4  8]
 [12 16 20]]
```

17. Aquí se muestran solo las filas.

```
[33]: print(b, '\n')
      print('b[:,0] =', 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]]]

b[:,0] = [[ 0  1  2  3]
 [12 13 14 15]]
```

18. Aquí mostramos la columna 1 del bloque 0.

```
[34]: 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]
```

19. Aquí podemos mostrar la ultima columna del segundo bloque y luego lo podemos mostrar de dos en dos.

```
[34]: 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]
```

20. En este paso mostramos el array original y lo invertimos.

```
[38]: # El array original

      print(b, '\n-----------------------\n'
      # Esta instrucción invierte los bloque
      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]]]
```

21. Aquí la función ravel ( ) genera un vector por medio de la matriz.

```
[39]: print('Matriz b =\n', b, '\n-------------------------\n')

      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]
```

22. La función flotten lo que hace es generar un nuevo espacio en memoria.

```
[40]: 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]
```

23. Aquí podemos cambiarle el tamaño a la matriz.

```
[44]: 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]]
```

24. Aquí podemos mostrar la matriz traspuesta.

```
[44]: 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]]
```

25. la función resize lo que hace es cambiar la estructura de la matriz.

[46]:
```
b.resize([2,12])

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]]
```