Arreglos de NumPy

Los arreglos de NumPy son creados llamando al método array() de la librería de NumPy.

```
import numpy as np
sample_list = [1, 2, 3]
np.array(sample_list)
```

La última línea de ese bloque de código dará como resultado una salida que se ve así.

```
array([1,2,3])
```

Hay dos tipos diferentes de arreglos de NumPy: vectores y matrices.

Los vectores son arreglos de NumPy uni-dimensionales y se ve así:

```
my_vector = np.array(['este', 'es', 'un', 'vector'])
```

Las matrices son arreglo bi-dimensionales y son creadas pasando una lista de lista dentro del método np.array(). Un ejemplo es el siguiente.

```
my_matrix = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]

np.array(my_matrix)
```

Obtener un rango de números en Python utilizando Numpy

NumPy tiene un método útil llamado arange que toma dos números y devuelve un arreglo de números enteros que son mayores o iguales a (>=) el primer número y menores que (<) el segundo número.

Un ejemplo del método arange es el siguiente.

```
np.arange(0,5)
#Devuevle array([0, 1, 2, 3, 4])
```

Un ejemplo de uso de la tercera variable en el método arange:

```
np.arange(1,11,2)
#Returns array([1, 3, 5, 7, 9])
```

Cómo generar Unos y Ceros en Python usando NumPy

```
np.zeros(4)
#Devuelve array([0, 0, 0, 0])
```

También puedes hacer algo similar utilizando matrices tridimensionales. Por ejemplo, np.zeros(5, 5) crea un arreglo de 5x5 que contiene todos ceros.

Podemos crear arreglos de unos usando un método similar llamado ones. Un ejemplo es el que sigue.

```
np.ones(5)
#Returns array([1, 1, 1, 1, 1])
```

Cómo dividir uniformemente un rango de números en Python usando NumPy

Hay muchas situaciones en las que tienes un rango de números y te gustaría dividir por igual ese rango de números en intervalos. El método linspace de NumPy está diseñado para resolver este problema. linspace tiene tres argumentos:

- El inicio del intervalo
- El fin del intervalo
- El número de subintervalos en los que deseas que se divida el intervalo

```
np.linspace(0, 1, 10)

#Devuelve array([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
```

Cómo crear un arreglo Identidad en Python usando NumPy

```
np.eye(1)
#Devuelve un arreglo identidad de 1x1

np.eye(2)
#Devuelve un arreglo identidad de 2x2

np.eye(50)
#Devuelve un arreglo identidad de 50x50
```

Cómo crear números aleatorios en Python usando NumPy

NumPy tiene varios métodos integrados que te permiten crear matrices de números aleatorios.

```
np.random.rand(sample_size)
#Devuelve una muestra de números aleatorios entre 0 y 1.

#El tamaño de la muestra puede ser un número entero (para un arreglo unidimensional) o dos enteros separados por comas (para un arreglo bidimensional).

np.random.randn(sample_size)

#Devuelve una muestra de números aleatorios entre 0 y 1, siguiendo la distribución normal
```

#El tamaño de la muestra puede ser un número entero (para un arreglo unidimensional) o dos enteros separados por comas (para un arreglo bidimensional).

np.random.randint(low, high, sample_size)

#Devuelve una muestra de números enteros que son mayores o iguales que 'low' y menores que 'high'

Cómo remodelar arreglos de NumPy

```
arr = np.array([0,1,2,3,4,5])
arr.reshape(2,3)
```

La salida de esta operación es:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

Si tienes curiosidad sobre la forma actual de un arreglo NumPy, puede determinar su forma utilizando el atributo shape de NumPy. Usando nuestra estructura de la variable arr anterior, a continuación se muestra un ejemplo de cómo llamar al atributo shape:

```
arr = np.array([0,1,2,3,4,5])
arr.shape
#Devuelve (6,)- ten en cuenta que no hay un segundo elemento ya que es un arreglo unidimensional

arr = arr.reshape(2,3)
arr.shape
#Devuelve (2,3)
```

También puede combinar el método reshape con el atributo shape en una línea como esta:

```
arr.reshape(2,3).shape
#Devuelve (2,3)
```

Cómo encontrar el valor máximo y mínimo de un arreglo NumPy

```
simple_array = [1, 2, 3, 4]
```

Podemos usar el método max para encontrar el máximo valor de un arreglo de NumPy. A continuación se muestra un ejemplo.

```
simple_array.max()
#Devuelve 4
```

Podemos usar también el método argmax para encontrar el índice del máximo valor dentro de un arreglo.

```
simple_array.argmax()
#Devuelve 3
```

En forma similar, podemos usar los métodos min y argmin.

```
simple_array.min()
#Devuelve 1

simple_array.argmin()
#Devuelve 0
```

Métodos y Operaciones de NumPy

Cómo realizar operaciones aritméticas en Python usando NumPy

NumPy facilita realizar operaciones aritméticas con arreglos.

Al sumar un sólo número a un arreglo de NumPy, ese número se suma a cada elemento en el arreglo.

```
2 + arr
#Devuelve array([2, 3, 4, 5])
```

Puedes sumar dos arreglos NumPy usando el operador +.
A continuación se ve un ejemplo.

```
arr + arr

#Devuelve array([0, 2, 4, 6])
```

Resta

Como la suma, la resta se realiza elemento por elemento para arreglos de NumPy.

```
arr - 10
#Devuelve array([-10,  -9,  -8,  -7])

arr - arr
#Devuelve array([0, 0, 0, 0])
```

Multiplicación

```
6 * arr
#Devuelve array([ 0,  6, 12, 18])

arr * arr
```

#Devuelve array([0, 1, 4, 9])

División

arr / 2
#Devuelve array([0. , 0.5, 1. , 1.5])

Al dividir por cero es con un arreglo NumPy que se muestra a continuación.

arr / arr
#Devuelve array([nan, 1., 1., 1.])

Cómo calcular raíz cuadrada usando NumPy

Puedes calcular la raíz cuadrada de cada elemento en un arreglo usando el método np.sqrt:

np.sqrt(arr)

#Devuelve array([0., 1., 1.41421356, 1.73205081])

np.sin(arr)

#Calcula el seno trigonométrico de cada valor en el arreglo

np.cos(arr)
Indexación en arreglos NumPy.

arr = np.random.rand(5)
array([0.69292946, 0.9365295 , 0.65682359, 0.72770856, 0.83268616])
arr = np.round(arr, 2)

Nuevo arreglo:

array([0.69, 0.94, 0.66, 0.73, 0.83])

Cómo retornar un elemento específico de un arreglo de NumPy

arr[0]
#Devuelve 0.69

Referenciación de arreglos en NumPy

array_to_copy = np.array([1, 2, 3])
copied_array = array_to_copy.copy()
array_to_copy
#Returns array([1, 2, 3])
copied_array
#Returns array([1, 2, 3])

Arreglos NumPy de dos dimensiones

```
mat = np.array([[5, 10, 15],[20, 25, 30],[35, 40, 45]])
mat
```

Devuelve:

```
array([[ 5, 10, 15],

       [20, 25, 30],

       [35, 40, 45]])
```

Hay dos formas de indexar un arreglo NumPy de dos dimensiones:

```
mat[fila, columna]
mat[fila][columna]
```

```
mat[0]
#Luego, obtengamos el último elemento de la primera fila:
mat[0][-1]
```

También puede generar submatrices a partir de un arreglo NumPy bidimensional utilizando esta notación:

```
mat[1:][:2]

"""
Devuelve:
array([[20, 25, 30],
       [35, 40, 45]])"""
```

Los arreglos NumPy admiten una característica llamada selección condiciona, que le permite generar un nuevo arreglo de valores booleanos que indican si cada elemento dentro del arreglo satisface una condición particular.

Un ejemplo de esto está abajo (también recreé nuestra variable arr original ya que ha pasado un tiempo desde que la vimos):

```
arr = np.array([0.69, 0.94, 0.66, 0.73, 0.83])
arr > 0.7
#Devuelve array([False,  True, False,  True,  True])
```

También puedes generar un nuevo arreglo de valores que satisfagan esta condición.

```
arr[arr > 0.7]
#Devuelve array([0.94, 0.73, 0.83])
```