Python para Análisis de Datos

Módulo 03

Numpy

Numpy

Numpy (de *Numerical Python*) es la principal librería de cálculo numérico de Python y el ecosistema de librerías científicas del lenguaje (como Pandas y Matplotlib) recurre a Numpy para realizar sus operaciones.

Se basa en una estructura de datos multidimensional llamada **array** y en una serie de funciones muy optimizadas para su manipulación, incluyendo operaciones matemáticas, lógicas, de selección y ordenamiento, operaciones estadísticas, álgebra lineal y más.

Array

El *array* (o *ndarray*) es la estructura que permite acelerar las operaciones matemáticas en Python, sin la necesidad de recurrir a bucles para operar con todos sus elementos, lo que se llama **vectorización**.

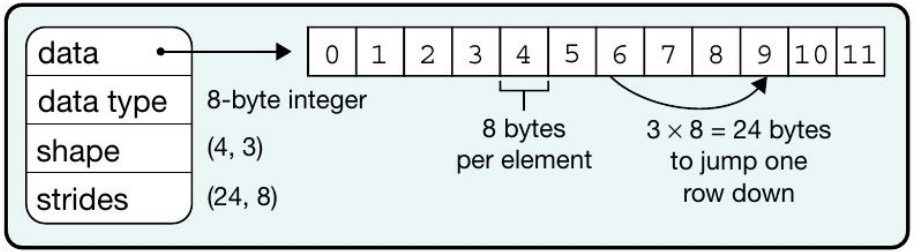
El array es una estructura de datos mucho más optimizada que la lista, pero a cambio presenta menos flexibilidad. El array sólo admite elementos de un único tipo de dato y no se puede cambiar la cantidad de elementos. Esto garantiza que siempre ocupe el mismo espacio y todos sus elementos se puedan representar en una sección continua de memoria.

Es una estructura multidimensional, por lo que permite representar vectores, matrices o arreglos de más dimensiones.

Atributos de un array

- **dtype**: el tipo de dato del array, en general numérico.
- **size**: cantidad de elementos.
- **nbytes**: total de bytes consumidos por los datos (bytes del tipo de dato por la cantidada de elementos).
- **ndim**: número de dimensiones.
- **shape**: cantidad de elementos en cada dimensión.
- **strides**: cantidad de bytes (o paso) para moverse una posición en cada dimensión.

$$x = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix}$$



Tipos de datos

Es uno de los atributos fundamentales de un array. El tipo de dato se define cuando se crea el array. Una vez creado, no es posible cambiar el tipo de dato del mismo, a lo sumo es posible crear otro array en base al original.

Tiene tipos de datos numéricos de distinta precisión; `datetime64` y `timedelta` para trabajar con fechas, horas y diferencias de tiempo. También tiene el tipo de dato `object` que crea referencias a cualquier tipo de objeto de Python (como strings). Los objetos no son almacenados en el array, sólo las referencias.

dtype	Variants
int	int8, int16, int32, int64
uint	uint8, uint16, uint32, uint64
bool	Bool
float	float16, float32, float64, float128
complex	complex64, complex128, complex256

Creación de arrays

Hay muchas formas de crear arrays. A través de la función **`np.array`** se pueden usar secuencias como listas (que pueden estar anidadas para representar arreglos multidimensionales). Podemos pasar el parámetro `dtype` para forzar a representar los datos en un tipo determinado. Si no se pasa, la función tratará de inferirlo.

```
1 arr1 = np.array([1,2,3,4,5])
2 arr2 = np.array([1,2,3,4,5], dtype=np.float64)
3 arr3 = np.array([1,2,3,4,5], dtype=np.int8)
4
5 print(arr1, arr1.dtype, arr1.nbytes)
6 print(arr2, arr2.dtype, arr2.nbytes)
7 print(arr3, arr3.dtype, arr3.nbytes)
```

```
[1 2 3 4 5] int64 40
[1. 2. 3. 4. 5.] float64 40
[1 2 3 4 5] int8 5
```

Notar que los flotantes se imprimen con un punto y que el array de enteros de 8 bits ocupa 8 veces menos espacio en memoria.

Podemos armar un array de dos dimensiones con listas anidadas, siempre y cuando todas tengan la misma cantidad de elementos.

```
1  arr = np.array([[1,2,3,4], [4,5,6,7], [8,9,0,1]])
2
3  print(arr)
4  print("ndim:", arr.ndim, "- shape:", arr.shape)
```

```
[[1 2 3 4]
 [4 5 6 7]
 [8 9 0 1]]
ndim: 2 - shape: (3, 4)
```

Además hay numerosas funciones para crear arrays:

- **`arange`** y **`linspace`** para rangos de números equiespaciados.
- **`zeros, ones`** y **`full`** para arrays de cualquier dimensión y forma.
- **`identity, eye`** y **`diagonal`** para arrays de 2 dimensiones.
- El módulo **`random`** permite crear arrays con distintas distribuciones de números al azar.
- Las funciones **`loadtxt`** y **`genfromtxt`** permiten leer archivos de texto tipo csv.

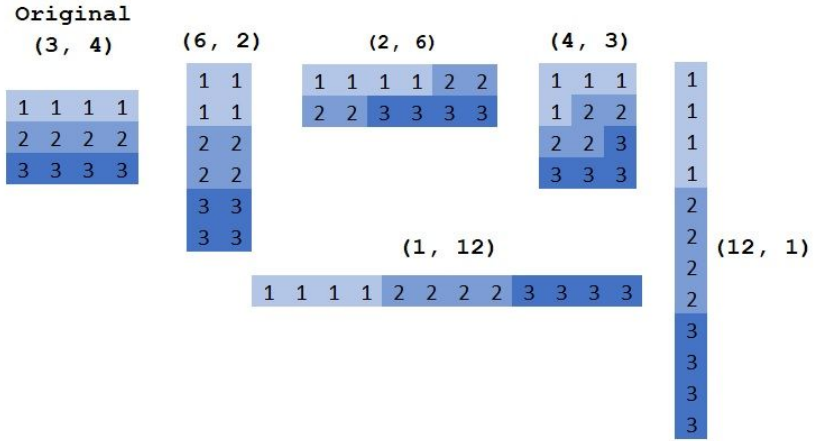
Tener en cuenta especificar el tipo de dato y el shape cuando sea necesario.

Reshaping

Ya vimos que el tipo de dato es muy importante en un array. Ahora nos vamos a centrar en el atributo **shape.** Éste determina la forma del array, lo que implica la dimensión y el paso.

Los datos se almacenan en un espacio contiguo de memoria y el shape determina cómo debe ser interpretada la forma del array: cuántos elementos debe contener cada dimensión. Se especifica como una tupla de números enteros. Estos números deben ser compatibles con la cantidad de elementos.

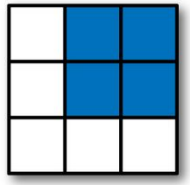
Para cambiar la forma del array se usa el método **reshape.** El método **flatten** transforma el array en unidimensional.



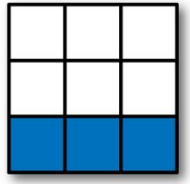
Indexing

Se conoce como ***indexing*** a cualquier operación que seleccione elementos de una array a través de corchetes (`[]`).

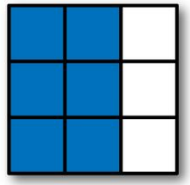
Para un array unidimensional la indexación funciona igual que con las listas. Sin embargo, para los arrays multidimensionales es posible especificar los índices de cada dimensión dentro de los mismos corchetes.



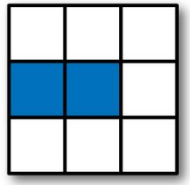
```
arr[:2, 1:]
```



```
arr[2]
arr[2, :]
arr[2:, :]
```



```
arr[:, :2]
```



```
arr[1, :2]
arr[1:2, :2]
```

Una característica interesante de Numpy es que permite seleccionar elementos en base a una condición. Esto es posible porque se puede usar un array de booleanos para seleccionar elementos de otro array, lo que se conoce como **enmascaramiento**.

```
1 numeros = np.array([10, 20, 30, 40])
2 booleanos = np.array([True, False, False, True])
3 numeros[booleanos]
```

```
array([10, 40])
```

Para construir arrays de booleanos se pueden usar expresiones condicionales con el mismo array (lo que profundizaremos más adelante). Luego, estos arrays se pueden usar para seleccionar elementos para los que se verifique la condición.

```
1 arr = np.array([10, 20, 30, 40, 50])
2 arr > 30
```

```
array([False, False, False,  True,  True])
```

```
1 arr[arr > 30]
```

```
array([40, 50])
```

Asignaciones

Con la notación de corchetes no sólo es posible elegir elementos de un array, sino que también se puede reasignar sus valores.

```
1  b = np.array([[1,2,3,4],[5,6,7,8]])
2  b[:,2] = 0
3  b
```

```
array([[1, 2, 0, 4],
       [5, 6, 0, 8]])
```

```
1  b[b < 3] = 11
2  b
```

```
array([[11, 11, 11,  4],
       [ 5,  6, 11,  8]])
```

Revisión

1. Repasar las características de un array.
2. Crear arrays a partir de listas y de funciones.
3. Modificar la forma (shape) de los arrays.
4. Seleccionar elementos por slices y por condiciones.
5. Cambiar los elementos de un array.

¡Muchas gracias!

¡Sigamos trabajando!