Indexación de ndarray

"Indexación de ndarray" © 2021,2022 by Francisco José Madrid Cuevas @ Universidad de Córdoba.España is licensed under CC BY-NC-SA 4.0. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/.

Para acceder a los elementos de un ndarray deberemos indexarlos usando el tradicional operador `[]`.

Este cuaderno se basa en la documentación oficial que puedes consultar aqui.

Configuración del entorno.

Como siempre, comenzamos cargando el módulo y visualizando su versión ya que este dato es importante para consultar la ayuda.

En el momento de escribir este tutorial la versión de Numpy con la que se trabaja es 1.22.4:

In [1]:
```
import numpy as np
print('Numpy version: {}'.format(np.__version__))
```

```
Numpy version: 1.22.4
```

Indexación básica.

Vamos a crear un array de una dimensión:

In [2]:
```
a = np.arange(10)
print("a.shape= ", a.shape)
```

```
a.shape=  (10,)
```

Para acceder a los elementos del array usaremos el operador '[]'. Hay que recordar que el ndarray usa la convención de que el primer elemento de una dimensión tendrá el índice 0.

Ejercicio 01: Imprimir el primer y el último elemento de un array de 10 valores.

La salida debería ser algo como:

```
Array:  [0 1 2 3 4 5 6 7 8 9]
El primer elemento ... 0
El último elemento ... 9
```

In [3]:
```
a = np.arange(10)
print("Array: ", a)
pri=None
ult=None
#pon tu código aquí

#
print(f'El primer elemento ... {pri}')
print(f'El último elemento ... {ult}')
```

```
Array:  [0 1 2 3 4 5 6 7 8 9]
El primer elemento ... None
El último elemento ... None
```

También podemos indicar índices negativos para contar desde el último elemento, por ejemplo, el índice -1 indica el último elemento, -2 el penúltimo y así sucesivamente.

Ejercicio 02: Imprimir, usando índices negativos, el último y el penúltimo elemento de un array.

La salida debería ser algo como:

```
Array:  [0 1 2 3 4 5 6 7 8 9]
El último elemento    ... 9
El penúltimo elemento ... 8
```

In [5]:
```
a = np.arange(10)
print("Array: ", a)
ult=None
pen=None

#pon tu código aquí

#

print(f'El último elemento    ... {ult}')
print(f'El penúltimo elemento ... {pen}')
```

```
Array:  [0 1 2 3 4 5 6 7 8 9]
El último elemento    ... None
El penúltimo elemento ... None
```

Si tenemos un ndarray con más de una dimensión, para seleccionar ún elemento necesitaremos indicar el índice en cada dimensión.

Ejercicio 03: Dada una matriz de tamaño 3x4, imprimir el elemento situado en la segunda fila tercera columna.

La salida debería ser algo como:

```
Matriz:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Elemento segunda fila, tercera columna 6
```

In [6]:
```
a = np.arange(12).reshape(3,4)
print("Matriz: \n", a)
v = None
#pon tu código aquí

#
print('Elemento segunda fila, tercera columna {}'.format(v))
```

```
Matriz:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Elemento segunda fila, tercera columna None
```

Además de usar constantes literales para indicar un índice, se puede usar cualquier expresión que se evalúa a un número entero.

Ejercicio 04: Dado un array de tamaño N, acceder al elemento central, es decir aquel con índice N dividido entre 2.

La salida debería ser algo como:

```
Array:
 [ 0  1  2  3  4  5  6  7  8  9 10]
Elemento central es 5
```

In [7]:
```
N = 11
a = np.arange(N)
print("Array: \n", a)
v = None
#pon tu código aquí
#Sugerencia: asegúrate de poner una expresión que se evalúe
# a un número entero.

#
print('Elemento central es {}'.format(v))
```

```
Array:
 [ 0  1  2  3  4  5  6  7  8  9 10]
Elemento central es None
```

Si intentamos acceder a un elemento con un índice fuera del rango permitido, se genera una excepción `IndexError` que podremos capturar y procesar convenientemente.

In [8]:
```
a = np.arange(10)
try:
    v = a[11]
except IndexError:
    print("Error al acceder fuera de rango.")
```

```
Error al acceder fuera de rango.
```

Selección básica de rangos.

En lugar de indexar un elemento particular, también podemos obtener rangos (ver slicing) usando la notación `inicio:fin:paso`.

Por defecto el paso es de 1 si no se indica. Así por ejemplo `1:3` define el rango desde el elemento `[1]` hasta el `[2]` (el índice fin no es incluido).

Si no se indica inicio, se cuenta desde 0. Si no se indica fin se cuenta hasta el último elemento inclusive, por lo tanto el rango `:` obtiene todos los elementos y `1::2` obtiene todos los elementos con índice impar.

Ejercicio 05: Dado un array de tamaño N, obtener el rango de elementos desde el primero hasta el medio (N//2) inclusive.

El resultado debería ser parecido a:

```
El rango es:  [0 1 2 3 4 5]
```

In [10]:
```
N = 11
a = np.arange(N)
b = None
#Pon tu código aquí.

#
print('El rango es: ', b)
```

```
El rango es:  None
```

Ejercicio 06: Dado un array de tamaño N, obtener el rango de elementos desde el medio (N//2) inclusive hasta el final.

El resultado debería ser parecido a:

```
El rango es:  [ 5  6  7  8  9 10]
```

In [11]:
```
N = 11
a = np.arange(N)
b = None
#Pon tu código aquí.

#
print('El rango es: ', b)
```

```
El rango es:  None
```

Ejercicio 07: Dado un array de tamaño N, obtener el rango de elementos desde el medio -1 al elemento medio +1 inclusive.

El resultado debería ser parecido a:

```
El rango es:  [4 5 6]
```

In [12]:
```
N = 11
a = np.arange(N)
b = None
#Pon tu código aquí.

#
print('El rango es: ', b)
```

```
El rango es:  None
```

Ejercicio 08: Dado un array de tamaño N, obtener el rango de elementos que corresponden con las posiciones pares.

El resultado debería ser parecido a:

```
El rango es:  [ 0  2  4  6  8 10]
```

In [13]:
```
N = 11
a = np.arange(N)
b = None
#Pon tu código aquí.
```

```
#
print('El rango es: ', b)
```

```
El rango es:  None
```

Ejercicio 09: Dado un array de tamaño N, obtener el rango de elementos que corresponden con las posiciones impares.

El resultado debería ser parecido a:

```
El rango es:  [1 3 5 7 9]
```

In [14]:
```
N = 11
a = np.arange(N)
b = None
#Pon tu código aquí.

#
print('El rango es: ', b)
```

```
El rango es:  None
```

Rangos para más de una dimensión.

Si el ndarray tiene más de una dimensión al igual que ocurre con los índices, se puede indicar un rango para cada dimensión, separados por `,`. El orden sería `[primera-dimensión, segunda-dimensión, ...]`.

Ejercicio 10: Dada una matriz `a` con forma (3,4) obtener la submatriz `b` con forma (2,2) cuyo primer elemento es el elemento 0,0 de la matriz `a`.

El resultado debería ser parecido a:

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
b:
 [[0 1]
 [4 5]]
```

In [15]:
```
a = np.arange(12).reshape(3,4)
print('a: \n', a)
b = None
#Pon tu código aquí.

#
print('b: \n', b)
```

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
b:
 None
```

Ejercicio 11: Dada una matriz `a`, obtener la submatriz `b` con forma `(F,C)` cuyo primer elemento es el

elemento `f,c` de la matriz `a`.

El resultado debería ser parecido a:

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

In [16]:
```
a = np.arange(20).reshape(4,5)
print('a: \n', a)
F=3
C=2
f=1
c=1
b = None
#Pon tu código aquí.

#
print('b: \n', b)
```

```
a:
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
b:
 None
```

Ejercicio 12: Dada una matriz `a`, obtener la submatriz `b` con todos los elementos de `a` situados en una posición par de fila y columna.

El resultado debería ser parecido a:

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 [[ 0  2]
 [ 8 10]]
```

In [17]:
```
a = np.arange(16).reshape(4,4)
print('a: \n', a)
b = None
#Pon tu código aquí.

#
print('b: \n', b)
```

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
```

```
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 None
```

Ejercicio 13: Dada una matriz `a`, obtener la submatriz `b` con todos los elementos de `a` situados en una posición impar de fila y columna.

El resultado debería ser parecido a:

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 [[ 5  7]
 [13 15]]
```

In [18]:
```
a = np.arange(16).reshape(4,4)
print('a: \n', a)
b = None
#Pon tu código aquí.

#
print('b: \n', b)
```

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 None
```

Reducir una dimensión.

En ocasiones cuando cuando trabajamos con un ndarray con varias dimensiones puede ser útil obtener un vista del mismo reduciendo una de sus dimensiones, por ejemplo, si tenemos una matriz podemos obtener un vector fila o columna, si tenemos un tensor con tres dimensiones, obtener una matriz con el primer plano (el primer valor de la tripleta que forma cada elemento del tensor).

Para obtener estas vistas, las dimensiones que no se reducen se indican con el rango completo `:` para esa dimensión, y la dimensión que se reduce se indica con el rángo o índice deseado.

Ejercicio 14: Dada una matriz `a` obtener el vector correspondiente a la segunda fila.

El resultado debería ser parecido a:

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
```

```
 [4 5 6 7]
b.shape:  (4,)
```

In [19]:
```
a = np.arange(16).reshape(4,4)
print('a: \n', a)
b = np.empty(1)
#Pon tu código aquí.

#
print('b: \n', b)
print('b.shape: ', b.shape)
```

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 [0.]
b.shape:  (1,)
```

Ejercicio 15: Dada una matriz `a` obtener el vector correspondiente a la tercera columna.

El resultado debería ser parecido a:

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 [ 2  6 10 14]
b.shape:  (4,)
```

In [20]:
```
a = np.arange(16).reshape(4,4)
print('a: \n', a)
b = np.empty(1)
#Pon tu código aquí.

#
print('b: \n', b)
print('b.shape: ', b.shape)
```

```
a:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
b:
 [0.]
b.shape:  (1,)
```

Ejercicio 16: Dada una imagen digital RGB codificada como un tensor `a` con forma (filas, columnas, 3) matriz `a` obtener la matriz `b` correspondiente al plano R.

El resultado debería ser parecido a:

```
a:
 [[[ 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]
     [27 28 29]
     [30 31 32]
     [33 34 35]]

    [[36 37 38]
     [39 40 41]
     [42 43 44]
     [45 46 47]]]
   b:
    [[ 0  3  6  9]
    [12 15 18 21]
    [24 27 30 33]
    [36 39 42 45]]
   b.shape:  (4, 4)
```

In [21]:

```
a = np.arange(48, dtype='B').reshape(4,4,3)
print('a: \n', a)
b = np.empty(1)
#Pon tu código aquí.

#
print('b: \n', b)
print('b.shape: ', b.shape)
```

```
a:
 [[[ 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]
  [27 28 29]
  [30 31 32]
  [33 34 35]]

 [[36 37 38]
  [39 40 41]
  [42 43 44]
  [45 46 47]]]
b:
 [0.]
b.shape:  (1,)
```

Aplanando las dimensiones.

Cuando se trabaja con más de una dimensión, puede ser interesante indexar el ndarray como si tuviera una sola dimensión. Esto se conoce como "aplanar el ndarray".

Ya hemos visto que la función ravel() recibe un ndarray y devuelve una copia "aplanada".

Si no queremos duplicar la memoria y sólo necesitamos acceder a los elementos del ndarray como si tuviera una sóla dimensión podemos utilizar el atributo ndarray.flat para obtener una vista plana del ndarray. De forma alternativa, el tipo `ndarray` proporciona un método ndarray.flatten() con la misma funcionalidad.

El resultado del aplanado dependerá de como se almacenen las dimensiones en memoria, por defecto se utiliza el formato usado por el lenguaje de programación 'C'. Si queremos otro tipo de ordenación (por ejemplo la usada en el lenguaje Fortran) necesitaremos obtener una copia usando las funciones `ravel()` o `flatten()`.

Ejercicio 17: Dada una imagen digital RGB codificada como un tensor `a` con tipo byte y con forma (filas, columnas, 3), obtener el valor del byte almacenado en la posición de memoria 24 relativa al comienzo del bloque de memoria que almacena los datos de la imagen.

El resultado debería ser parecido a:

```
a:
 [[[ 7  8  9]
  [10 11 12]
  [13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]
  [25 26 27]
  [28 29 30]]

 [[31 32 33]
  [34 35 36]
  [37 38 39]
  [40 41 42]]

 [[43 44 45]
  [46 47 48]
  [49 50 51]
  [52 53 54]]]
Valor almacenado en la posición 24:  31
```

In [22]:
```
a = np.arange(7, 7+48, dtype='B').reshape(4,4,3)
print('a: \n', a)
b = None
#Pon tu código aquí.
#Sugerencia: como no queremos duplicar la memoria, utiliza el atributo
# flat para obtener una vista aplanada.

#
print('Valor almacenado en la posición 24: ', b)
```

```
a:
 [[[ 7  8  9]
  [10 11 12]
  [13 14 15]
  [16 17 18]]

 [[19 20 21]
```

```
  [22 23 24]
  [25 26 27]
  [28 29 30]]

 [[31 32 33]
  [34 35 36]
  [37 38 39]
  [40 41 42]]

 [[43 44 45]
  [46 47 48]
  [49 50 51]
  [52 53 54]]]
Valor almacenado en la posición 24:  None
```

Añadir una dimensión.

A la hora de indexar un ndarray necesitamos que la vista obtenida tenga más dimesiones que la original. Para ellos podemos utilizar el objeto `numpy.newaxis` para generar esta nueva dimensión ("axis") con un tamaño de 1.

Por ejemplo si tenemos una array 1-D pasaremos a tener un array 2-D:

In [23]:
```
a = np.arange(10)
print(f'A con forma original           : {a.shape}')
print(a)
print('')
print(f'A con forma extendida (filas)   :{a[np.newaxis, :].shape}:')
print(a[np.newaxis, :])
print('')
print(f'A con forma extendida (columnas): {a[:, np.newaxis].shape}')
print(a[:, np.newaxis])
```

```
A con forma original           : (10,)
[0 1 2 3 4 5 6 7 8 9]

A con forma extendida (filas)   :(1, 10):
[[0 1 2 3 4 5 6 7 8 9]]

A con forma extendida (columnas): (10, 1)
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
```

Los rangos son "vistas".

Un aspecto a tener siempre en cuenta es que, en general, los rangos obtenidos son "vistas" del ndarray y no copias, por lo que si modificamos el valor en un rango estaremos modificando el valor en el ndarray correspondiente.

Ejercicio 18: Comprobar que si modificamos el primer valor del rango obtenido de un array seleccionando las posiciones pares, el valor del ndarray también se modifica.

El resultado debería ser parecido a:

```
a antes  :  [ 0  1  2  3  4  5  6  7  8  9 10]
b antes  :  [ 0  2  4  6  8 10]
b después:  [-1  2  4  6  8 10]
a después:  [-1  1  2  3  4  5  6  7  8  9 10]
```

In [24]:
```
a = np.arange(11)
b = [0]
print('a antes  : ', a)
#Pon tu código aquí.
#Sugerencia: usar una selección de rango básica.

#
print('b antes  : ', b)
b[0]=-1
print('b después: ', b)
print('a después: ', a)
```

```
a antes  :  [ 0  1  2  3  4  5  6  7  8  9 10]
b antes  :  [0]
b después:  [-1]
a después:  [ 0  1  2  3  4  5  6  7  8  9 10]
```

Ejercicio 19: Comprobar que si obtenemos una copia a partir de un rango correspondiente a un array seleccionando las posiciones pares y, posteriormente, modificamos el primer valor de la copia, el correspondiente valor en el ndarray no se ha modificado.

El resultado debería ser parecido a:

```
a antes  :  [ 0  1  2  3  4  5  6  7  8  9 10]
b antes  :  [ 0  2  4  6  8 10]
b después:  [-1  2  4  6  8 10]
a después:  [ 0  1  2  3  4  5  6  7  8  9 10]
```

In [25]:
```
a = np.arange(11)
print('a antes  : ', a)
b = [0]
#Pon tu código aquí.
#Sugerencia: utiliza np.copy ya que se busca que la varible 'b'
#  sea una copia del rango.

#
print('b antes  : ', b)
b[0]=-1
print('b después: ', b)
print('a después: ', a)
```

```
a antes  :  [ 0  1  2  3  4  5  6  7  8  9 10]
b antes  :  [0]
b después:  [-1]
a después:  [ 0  1  2  3  4  5  6  7  8  9 10]
```

In [ ]:
```

```