Fundamentos de Aprendizaje Automático

Introducción a Python/Numpy

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Montevideo, 2021

Tabla de contenido

Consideraciones previas

- Esta clase no sustituye la lectura del material de referencia provisto
  - https://cs231n.github.io/python-numpy-tutorial/
- El objetivo de la clase es complementar el material, destacando aquellos aspectos que consideramos mas relevantes para el curso
- Para un mejor aprovechamiento se recomienda
  1. Tener configurado el ambiente de desarrollo y un notebook abierto
  2. Acceder a Google Colaboratory y crear un nuevo notebook.
- Verificar el correcto funcionamiento ejecutando

```
>>> import numpy as np
```

Python

- Es un lenguaje de programación muy amigable
- En el curso usaremos la versión 3.8

Python

- Es un lenguaje de programación muy amigable
- En el curso usaremos la versión 3.8
- Como todo lenguaje de programación
  - Define una serie de tipos de datos básicos, por ej: int, float, str

```
>>> x = 3
>>> print(type(x)) # Prints "<class 'int'>"
```

  - Permite definir funciones

```
>>> def suma(a,b):
>>>     c = a + b
>>>     return c
```

  - Provee estructuras de control de flujo: for, while, if, if-else
  - Tiene contenedores: listas, diccionarios, sets

Listas

- Las listas son uno de los contenedores que tiene *Python*

```
>>> xs = [3, 1, 2]    # Se crea una lista
>>> print(xs, xs[2])  # Prints "[3, 1, 2] 2"
>>> print(xs[-1])     # Se accede al último elemento de la lista
>>> xs[2] = 'foo'     # Se modifica un elemento de la lista
>>> print(xs)         # Muestra "[3, 1, 'foo']"
>>> xs.append('bar')  # Se agrega un elemento al final de la lista
>>> print(xs)         # Muestra "[3, 1, 'foo', 'bar']"
```

Listas

- Las listas son uno de los contenedores que tiene *Python*

```
>>> xs = [3, 1, 2]    # Se crea una lista
>>> print(xs, xs[2])  # Prints "[3, 1, 2] 2"
>>> print(xs[-1])     # Se accede al último elemento de la lista
>>> xs[2] = 'foo'     # Se modifica un elemento de la lista
>>> print(xs)         # Muestra "[3, 1, 'foo']"
>>> xs.append('bar')  # Se agrega un elemento al final de la lista
>>> print(xs)         # Muestra "[3, 1, 'foo', 'bar']"
```

- Recorrer y mostrar elementos de una lista

```
# Muestra "cat", "dog", "monkey", uno en cada línea
>>> animals = ['cat', 'dog', 'monkey']
>>> for animal in animals:
>>>     print(animal)
cat
dog
monkey
```

Listas

- Las listas son uno de los contenedores que tiene *Python*

```
>>> xs = [3, 1, 2]    # Se crea una lista
>>> print(xs, xs[2])  # Prints "[3, 1, 2] 2"
>>> print(xs[-1])     # Se accede al último elemento de la lista
>>> xs[2] = 'foo'     # Se modifica un elemento de la lista
>>> print(xs)         # Muestra "[3, 1, 'foo']"
>>> xs.append('bar')  # Se agrega un elemento al final de la lista
>>> print(xs)         # Muestra "[3, 1, 'foo', 'bar']"
```

- Recorrer y mostrar elementos de una lista

```
# Muestra "0 cat", "1 dog", "2 monkey", uno en cada línea
>>> animals = ['cat', 'dog', 'monkey']
>>> for i, animal in enumerate(animals):
>>>     print(i, animal)
0 cat
1 dog
2 monkey
```

Suma de elementos de una lista

- Supongamos que tenemos dos listas y queremos hacer una operación elemento a elemento, por ejemplo la suma

```
>>> L1 = [1,2,3]
>>> L2 = [4,5,6]
```

- Hacer L1 + L2 no funciona

```
>>> print(L1+L2)  #concatena los elementos de la lista
[1, 2, 3, 4, 5, 6]
```

Suma de elementos de una lista

- Supongamos que tenemos dos listas y queremos hacer una operación elemento a elemento, por ejemplo la suma

```
>>> L1 = [1,2,3]
>>> L2 = [4,5,6]
```

- Hacer L1 + L2 no funciona

```
>>> print(L1+L2)  #concatena los elementos de la lista
[1, 2, 3, 4, 5, 6]
```

- Es necesario hacer algo un poco más sofisticado (y muy ineficiente)

```
# una forma de hacer la suma elemento a elemento
>>> suma_listas = []
>>> for e1, e2 in zip(L1, L2):
>>>     suma_listas.append(e1+e2)
>>> print(suma_listas)
[5,7,9]
```

NumPy

- NumPy (Numerical Python) es el paquete fundamental para el cálculo científico en *Python*.
- NumPy es fuertemente utilizada por muchas bibliotecas de mas alto nivel, por ejemplo Pandas, SciPy, **Matplotlib**, **scikit-learn**, scikit-image.
- Para acceder a las funciones provistas por NumPy hay que importarlo

```
>>> import numpy as np
```

Creación de arreglos

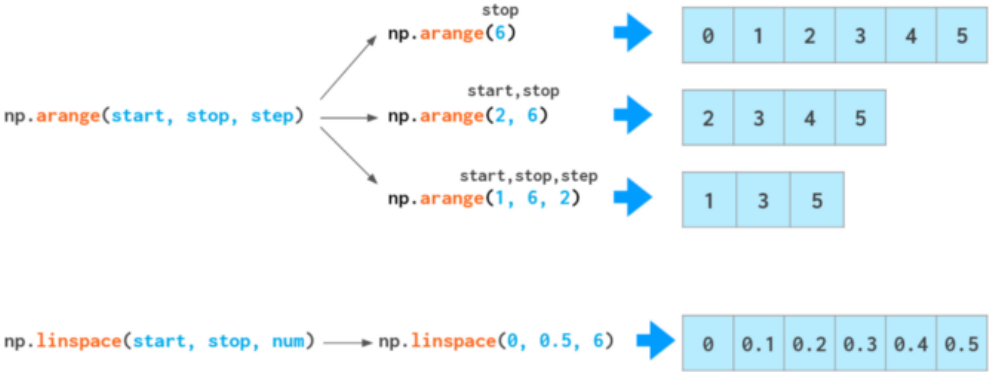
- Arreglos unidimensionales

```
>>> a = np.array([2, 3, 4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')

>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
>>> b.shape
(3,)
```

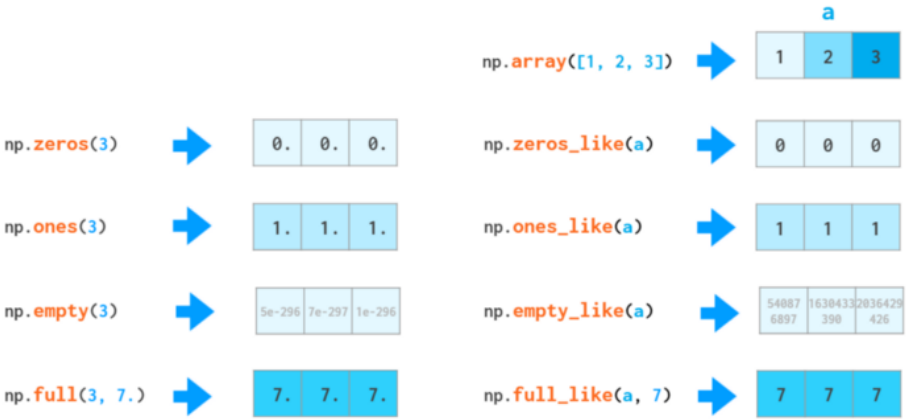
- Arreglos n-dimensionales

```
>>> b = np.array([(1.5, 2, 3), (4, 5, 6)])
>>> b
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
>> b.dtype
dtype('float64')

>> b.shape
(2,3)
```

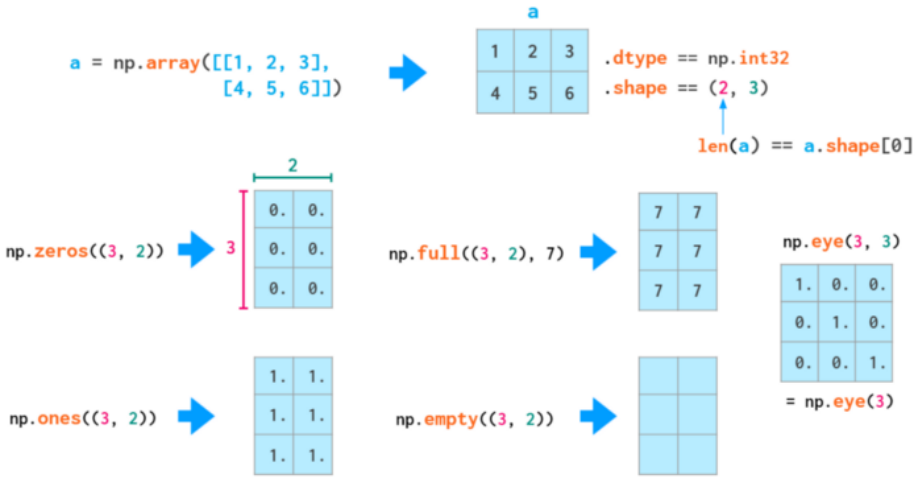
Creación de arreglos unidimensionales



Creación de arreglos unidimensionales

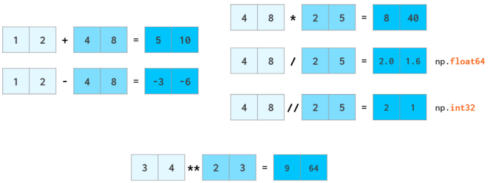


Creación de arreglos bidimensionales

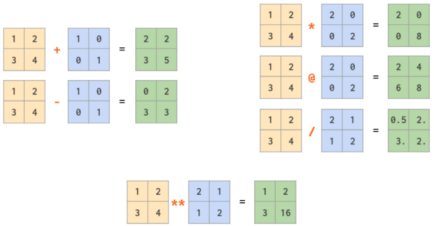


Operaciones básicas

- Vectores



- Matrices



Operaciones vectoriales y matriciales

- Vectoriales

```
>>> a = np.array([2, 3, 4])
>>> b = np.arange(3)   #[0,1,2]

>>> a.dot(b)  # producto escalar
11

>>> a @ b
11
```

- Matriciales

```
>>> A = np.array([[1, 1],
...               [0, 1]])
>>> B = np.array([[2, 0],
...               [3, 4]])

>>> A.dot(B)  # producto entre matrices
array([[5, 4],
       [3, 4]])
>>> A @ B     # producto entre matrices
array([[5, 4],
       [3, 4]])
```

Ejercicio

```
>>> x = np.arange(4)
>>> xx = x.reshape(4,1)
>>> y = np.ones(5)
>>> z = np.ones((3,4))

# Ejecutar y explicar la salida de

>>> z @ xx

>>> z * xx

>>> (xx.T).shape

>>> x + xx.T

>>> xx + y
```

Broadcasting

- Es el mecanismo por el que *numpy* permite realizar operaciones aritméticas simples entre elementos de distinto tamaño
- El ejemplo mas simple

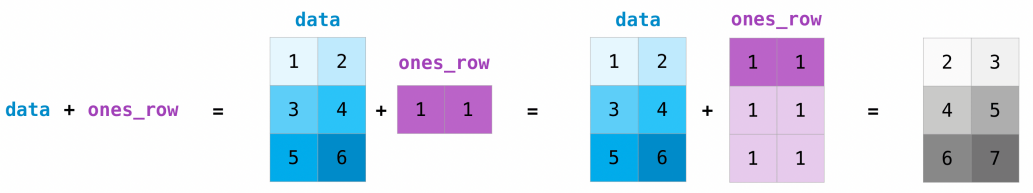
```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = 2.0

>>> a * b
array([ 2.,  4.,  6.])
```

Broadcasting

- Es el mecanismo por el que *numpy* permite realizar operaciones aritméticas simples entre elementos de distinto tamaño
- El ejemplo mas simple

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = 2.0

>>> a * b
array([ 2.,  4.,  6.])
```

- Un poco mas complejo



Condiciones del broadcasting

- La operación entre dos arreglos con el *mismo número de dimensiones* se realiza elemento a elemento si éstos tienen dimensiones compatibles, es decir, se cumple una de las siguientes:
  1. El tamaño de las dimensiones son iguales
  2. Cuando los tamaños de alguna dimensión son distintos, una tiene tamaño 1

Condiciones del broadcasting

- La operación entre dos arreglos con el *mismo número de dimensiones* se realiza elemento a elemento si éstos tienen dimensiones compatibles, es decir, se cumple una de las siguientes:
  1. El tamaño de las dimensiones son iguales
  2. Cuando los tamaños de alguna dimensión son distintos, una tiene tamaño 1

- Cuando el *número de dimensiones es distinto* se agregan dimensiones a la izquierda del arreglo de menor dimensión hasta *igualar el número de dimensiones*. Luego se evalúan lan condiciones descritas arriba.

$$
\begin{aligned}
A(4darray): &\quad 8x1x6x1 \\
B(3darray): &\quad 7x6x5 \\
Result(4darray): &\quad 8x7x6x5
\end{aligned}
$$

Condiciones del broadcasting

- La operación entre dos arreglos con el *mismo número de dimensiones* se realiza elemento a elemento si éstos tienen dimensiones compatibles, es decir, se cumple una de las siguientes:
  1. El tamaño de las dimensiones son iguales
  2. Cuando los tamaños de alguna dimensión son distintos, una tiene tamaño 1

- Cuando el *número de dimensiones es distinto* se agregan dimensiones a la izquierda del arreglo de menor dimensión hasta *igualar el número de dimensiones*. Luego se evalúan lan condiciones descritas arriba.

$A(4darray)$:	$8x1x6x1$
$B(3darray)$:	$1x7x6x5$
$Result(4darray)$:	$8x7x6x5$

Ejemplo broadcasting

```
>>> x = np.arange(4)
>>> xx = x.reshape(4,1)
>>> y = np.ones(5)
>>> z = np.ones((3,4))

>>> x.shape
(4,)

>>> y.shape
(5,)

>>> x + y
ValueError: operands could
not be broadcast together
with shapes (4,) (5,)
```

Ejemplo broadcasting

```
>>> x = np.arange(4)
>>> xx = x.reshape(4,1)
>>> y = np.ones(5)
>>> z = np.ones((3,4))

>>> x.shape
(4,)

>>> y.shape
(5,)

>>> x + y
ValueError: operands could
not be broadcast together
with shapes (4,) (5,)
```

```
>>> x.shape
(4,)

>>> z.shape
(3, 4)

>>> (x + z).shape
(3, 4)

>>> x + z
array([[ 1.,  2.,  3.,  4.],
       [ 1.,  2.,  3.,  4.],
       [ 1.,  2.,  3.,  4.]])
```

Ejemplo broadcasting

```
>>> x = np.arange(4)
>>> xx = x.reshape(4,1)
>>> y = np.ones(5)
>>> z = np.ones((3,4))

>>> x.shape
(4,)

>>> y.shape
(5,)

>>> x + y
ValueError: operands could
not be broadcast together
with shapes (4,) (5,)
```

```
>>> x.shape
(4,)

>>> z.shape
(3, 4)

>>> (x + z).shape
(3, 4)

>>> x + z
array([[ 1.,  2.,  3.,  4.],
       [ 1.,  2.,  3.,  4.],
       [ 1.,  2.,  3.,  4.]])
```

```
>>> xx.shape
(4, 1)

>>> (xx + y).shape
(4, 5)

>>> xx + y
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.,  4.]])
```

Funciones universales

- Son funciones matemáticas habituales (sin, cos, exp) que operan elemento a elemento. Lista completa acá.

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([1.        , 2.71828183, 7.3890561 ])
>>> np.sqrt(B)
array([0.        , 1.        , 1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([2., 0., 6.])
```

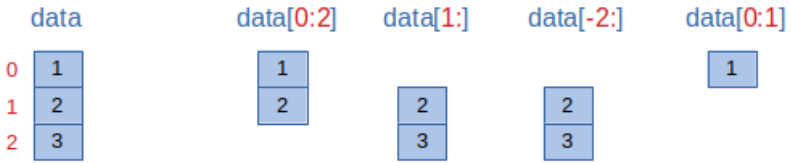
- Realizan *casteo* de tipo y *broadcasting* automático.

Indexing

- Es el mecanismo por el cual se accede a los elementos de una arreglo.
- Existen dos variantes principales
  - Slicing: se extrae una porción del arreglo especificando los índices inicial, final y el paso. El índice inicial es inclusivo mientras que el final no. Se trabaja sobre los mismos datos que el arreglo original (se obtiene una vista).
  - Fancy indexing: se obtiene la porción que interesa a partir de la *lista de índices* o una *máscara*. Se genera un nuevo arreglo con los datos seleccionados.

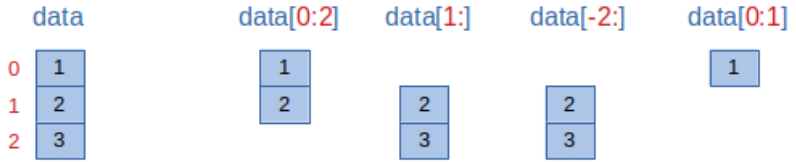
Slicing

- Slicing de arreglos unidimensionales

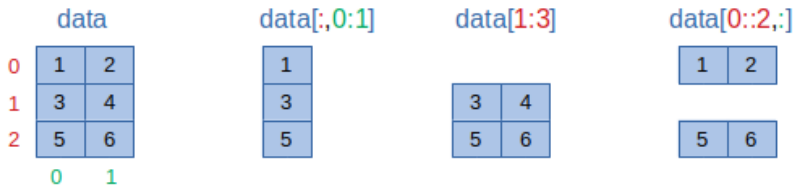


Slicing

- Slicing de arreglos unidimensionales



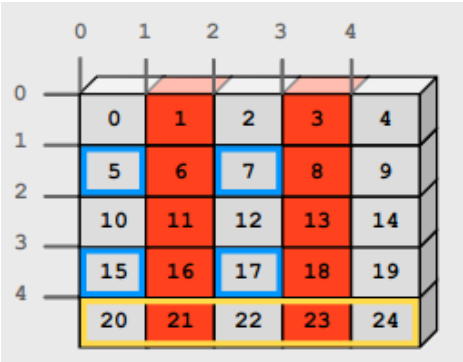
- Slicing de matrices



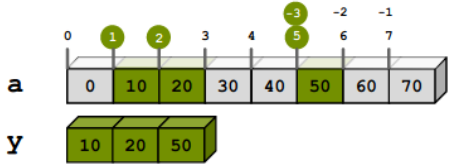
Ejercicio Slicing

- A partir del arreglo *a*, obtener los valores indicados en el diagrama.

```
a = np.arange(25).reshape(5, 5)
```



Fancy indexing



- Acceso mediante índices

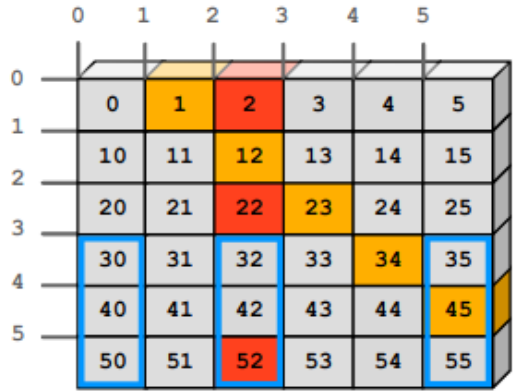
```
>>> a = np.arange(0, 80, 10)
# fancy indexing
>>> indices = [1, 2, -3]
>>> y = a[indices]
>>> y
array([10, 20, 50])
# se cambian los valores de los indices
>>> a[indices] = 99
>>> a
array([ 0, 99, 99, 30, 40, 99,
60, 70])
# los valores de y no cambian
```

- Acceso mediante una máscara

```
# acceso a elementos mediante mascaras
>>> mask = np.array(
...     [0, 1, 1, 0, 0, 1, 0, 0],
...     dtype=bool)
# fancy indexing
>>> y = a[mask]
>>> y
array([99, 99, 99])
```

Fancy indexing 2D

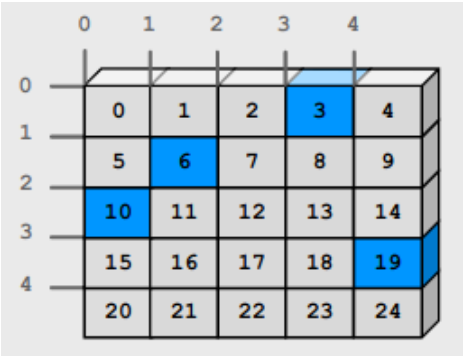
```
>>> a[[0, 1, 2, 3, 4],
...   [1, 2, 3, 4, 5]]
array([ 1, 12, 23, 34, 45])

>>> a[3:, [0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array(
...     [1, 0, 1, 0, 0, 1],
...     dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



Ejercicio Fancy indexing

Crear el arreglo bi-dimensional del diagrama y luego extraer

1. un arreglo que contenga los elementos en azul
2. los números divisibles por tres



Copias de arreglos

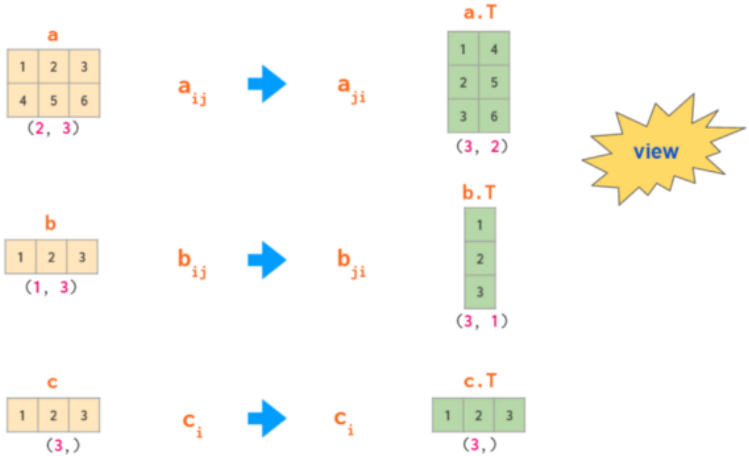
- Es una fuente habitual de confusiones
  - En este caso no se copian los datos

```
>>> a = np.array([[ 0,  1,  2,  3],
...               [ 4,  5,  6,  7],
...               [ 8,  9, 10, 11]])
>>> b = a            # no se crea un nuevo objeto
>>> b is a           # a y b son dos nombres para el mismo ndarray
True
```

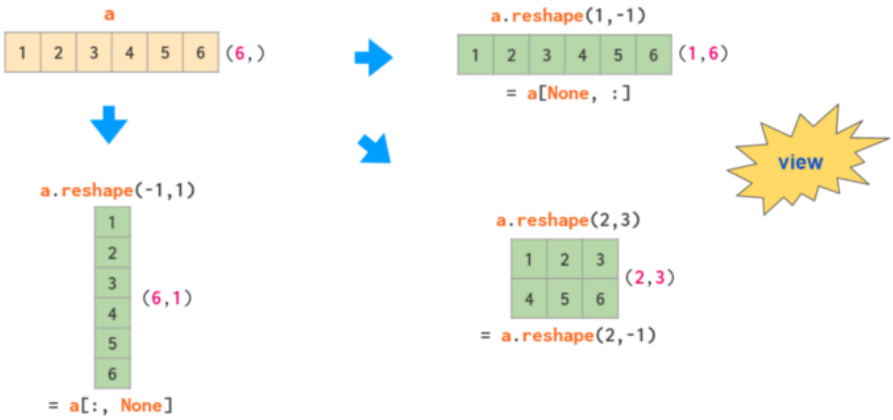
  - En este caso se copian

```
>>> c = a.copy()  # se crea un nuevo arreglo con nuevos datos
>>> c is a
False
```

Transformaciones de arreglos



Transformaciones de arreglos



1. Python
2. NumPy
3. Referencias

Referencias

- Python Numpy Tutorial, CS231n Standford Course
  https://cs231n.github.io/python-numpy-tutorial/
- NumPy user guide
  https://numpy.org/doc/stable/user/index.html
- SciPy 2021 Tutorial: Introduction to Numerical Computing With NumPy
  https://github.com/enthought/Numpy-Tutorial-SciPyConf-2021
- NumPy Illustrated: The Visual Guide to NumPy
  https://betterprogramming.pub/numpy-illustrated-the-visual-guide-to-numpy-3b1d4976de1d