Creado por:

Isabel Maniega

-6- Numpy

```
In [3]: import numpy as np
```

Numeros dentro de un rango:

Generación de números con numpy en un rango

```
In [2]: a = np.arange(6)
        a

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

```
In [3]: type(a)

Out[3]: numpy.ndarray
```

Formas de imprimir la información

```
In [4]: a

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

```
In [5]: for i in a:
            print(i)

        0
        1
        2
        3
        4
        5
```

Longitud, forma, tamaño

```
In [6]: a

Out[6]: array([0, 1, 2, 3, 4, 5])
```

```
In [7]: len(a)

Out[7]: 6
```

```
In [8]: a.shape

Out[8]: (6,)
```

```
In [9]: a.size

Out[9]: 6
```

Máximos y mínimos

```
In [10]: a

Out[10]: array([0, 1, 2, 3, 4, 5])
```

```
In [11]: max(a)

Out[11]: 5
```

```
In [12]: min(a)
```

Out[12]:
```
0
```

Comprobación de elementos en el array

In [13]:
```
a
```

Out[13]:
```
array([0, 1, 2, 3, 4, 5])
```

In [14]:
```
25 in a
```

Out[14]:
```
False
```

In [15]:
```
0 in a
```

Out[15]:
```
True
```

In [16]:
```
25 not in a
```

Out[16]:
```
True
```

In [17]:
```
0 not in a
```

Out[17]:
```
False
```

Redefinir el tamaño

In [18]:
```
a
```

Out[18]:
```
array([0, 1, 2, 3, 4, 5])
```

In [19]:
```
a1 = a.reshape(2, 3)
a1
```

Out[19]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

Generar números en un intervalo

In [20]:
```
# sin especificar va de 1 en 1
b = np.arange(2,7) # 2, 3, 4, 5, 6
b
```

Out[20]:
```
array([2, 3, 4, 5, 6])
```

Generar números en un intervalo con salto

In [21]:
```
c = np.arange(10, 40, 5)
c
```

Out[21]:
```
array([10, 15, 20, 25, 30, 35])
```

In [22]:
```
d = np.arange(10, 41, 5)
d
```

Out[22]:
```
array([10, 15, 20, 25, 30, 35, 40])
```

Un array de varias filas y columnas

In [39]:
```
e = np.array([
    [1, 2],
    [3, 4],
    [5, 6]
])
e
```

Out[39]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [24]:
```
len(e)
```
Out[24]:
```
3
```
In [25]:
```
e.shape
```
Out[25]:
```
(3, 2)
```
In [26]:
```
e.size
```
Out[26]:
```
6
```
In [31]:
```
e[0]
```
Out[31]:
```
array([1, 2])
```
In [42]:
```
for i in range(len(e)):
    # print(e[i])
    for j in range(len(e[i])):
        print(e[i][j])
```
```
1
2
3
4
5
6
```

Linspace

In [30]:
```
# Recogemos una muestra de los datos, especificamos: min, max, y cada tantos recoja un valor
```
In [27]:
```
f = np.linspace(10, 20, 2) # de 10 a 20 con 2 elementos
f
```
Out[27]:
```
array([10., 20.])
```
In [28]:
```
g = np.linspace(10, 20, 5) # de 10 a 20 muestra 5
g
```
Out[28]:
```
array([10. , 12.5, 15. , 17.5, 20. ])
```
In [29]:
```
g1 = np.linspace(10, 20, 3) # de 10 a 20 muestra 3
g1
```
Out[29]:
```
array([10., 15., 20.])
```

Matrices básicas en numpy

Matriz Identidad: Diagonal principal llena de 1, resto 0

In [2]:
```
h = np.eye(3) # de 3 filas y 3 columnas --> matriz identidad
h
```
Out[2]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```
In [3]:
```
i = np.eye(5) # Matriz de 5 filas y 5 columnas
i
```
Out[3]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Matriz identidad multiplicada por un valor

In [4]:

```
j = 5 * i
j
```

Out[4]:

```
array([[5., 0., 0., 0., 0.],
       [0., 5., 0., 0., 0.],
       [0., 0., 5., 0., 0.],
       [0., 0., 0., 5., 0.],
       [0., 0., 0., 0., 5.]])
```

Matriz de todo 1

In [5]:

```
k = np.ones((3, 4)) # Matriz de 3 filas por 4 columnas --> valores 1
k
```

Out[5]:

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Matriz de todo 0

In [6]:

```
l = np.zeros((3, 4)) # Matriz de 0s --> 3 filas por 4 columnas
l
```

Out[6]:

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [7]:

```
l2 = np.zeros((6, 2))
l2
```

Out[7]:

```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

Transpuesta de una matriz

Intercambio de filas por columnas

In [9]:

```
m = np.array([[1, 2, 3],
             [4, 5, 6]])
m
```

Out[9]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [10]:

```
# Opción 1
m.transpose()
```

Out[10]:

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [11]:

```
# Opción 2
m.T
```

Out[11]:

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

ALL y ANY

In [12]:

```
n = np.array([[1, 2, 3],
             [4, 5, 6]])
n
```

Out[12]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [13]:

```
# ALL --> ¿Todos los elementos son mayores de 0? --> True/False
np.all(n>0)
```

Out[13]:

```
True
```

In [14]:

```
np.all(n>2)
```

Out[14]:

```
False
```

In [15]:

```
# ANY --> ¿Algún elemento son mayores de 2?
np.any(n>2)
```

Out[15]:

```
True
```

Función Ravel

In [16]:

```
# Pone en una sola dimensión una matriz
```

In [17]:

```
p = np.array([[1, 2, 3],
             [4, 5, 6]])
p
```

Out[17]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [23]:

```
# np.ravel(matriz a modificar)
np.ravel(p)
```

Out[23]:

```
array([1, 2, 3, 4, 5, 6])
```

In [19]:

```
p1 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
p1
```

Out[19]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [20]:

```
np.ravel(p1)
```

Out[20]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Flatten

In [ ]:

```
# Es una copia del array pero en 1 sola dimensión
```

In [4]:

```
matriz = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
matriz
```

Out[4]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [5]:

```
# nombre matriz + flatten()
matriz.flatten()
```

Out[5]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [7]:

```
m = matriz.flatten()
```

In [9]:

```
m.shape
```

Out[9]:

```
(9,)
```

Roll

In [25]:

```
# np.roll(array, desplazamiento, eje)
# Desplaza los elementos de manera circular a través de una dimensión
```

In [26]:
```
b = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
b
```

Out[26]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [27]:
```
# Desplazamiento= 1 y eje horizontal
np.roll(b, 1, axis=0)
```

Out[27]:
```
array([[ 9, 10, 11, 12],
       [ 1,  2,  3,  4],
       [ 5,  6,  7,  8]])
```

In [28]:
```
# Desplazamiento = 1 y eje vertical
np.roll(b, 1, axis=1)
```

Out[28]:
```
array([[ 4,  1,  2,  3],
       [ 8,  5,  6,  7],
       [12,  9, 10, 11]])
```

In [29]:
```
# Desplazamiento= -1 y eje horizontal
np.roll(b, -1, axis=0)
```

Out[29]:
```
array([[ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4]])
```

In [30]:
```
# Desplazamiento = -1 y eje vertical
np.roll(b, -1, axis=1)
```

Out[30]:
```
array([[ 2,  3,  4,  1],
       [ 6,  7,  8,  5],
       [10, 11, 12,  9]])
```

logspace

In [ ]:
```
# Array de elementos logarítmicos espaciados
```

In [32]:
```
# np.logspace(10^inicio, 10^fin, divisiones(elementos))
# como en linspace se incluye los extremos (inicios-->fin)
c = np.logspace(0, 1, 3)
c
```

Out[32]:
```
array([ 1.        ,  3.16227766, 10.        ])
```

In [35]:
```
# 10^0 = 1 ; 10^1 = 1; 3 divisiones(elementos)
```

$$10^0 = 1 \;\; y \;\; 10^1 = 1 \;\; 3 divisiones(elementos)$$

Acceso a un elemento de un array

In [36]:
```
q = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
q
```

Out[36]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [37]:
```
# Opción 1
q[2][1] # --> fila 2 y columna 1 (listas 0, 1, 2)
```

Out[37]:
```
8
```

In [38]:
```
q[0][2]
```

Out[38]:
```
3
```

In [39]:

```
# Opción 2
q[2, 1]
```

Out[39]:

```
8
```

In [40]:

```
# dos primeras filas (: --> todas)
q[:2]
```

Out[40]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [41]:

```
q[2:]
```

Out[41]:

```
array([[7, 8, 9]])
```

In [56]:

```
# Filtrar por columnas
q[:,[0]]
```

Out[56]:

```
array([[1],
       [4],
       [7]])
```

In [57]:

```
# Filtrar por columnas
q[:,[0,1]]
```

Out[57]:

```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

Algunas operaciones

In [50]:

```
# Potencias
```

In [51]:

```
r = np.array([1, 2, 3, 4])
r
```

Out[51]:

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

In [52]:

```
# Método 1
```

In [53]:

```
r**2 # 1^1, 2^2, 3^3, 4^4
```

Out[53]:

```
array([ 1,  4,  9, 16])
```

In [ ]:

```
# Método 2
```

In [54]:

```
pow(r, 2)
```

Out[54]:

```
array([ 1,  4,  9, 16])
```

Comparación entre Arrays

In [ ]:

```
# Creamos los arrays
```

In [10]:

```
s = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
s
```

Out[10]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [11]:

```
t = np.array([
    [100, 200, 3],
    [400, 5, 6]
])
t
```

Out[11]:

```
array([[100, 200,   3],
       [400,   5,   6]])
```

Los comparo

np.where(condicion, si es cierto, si es falso)

In [12]:

```
np.where(s==t, "True", "False")
```

Out[12]:

```
array([['False', 'False', 'True'],
       ['False', 'True', 'True']], dtype='<U5')
```

In [13]:

```
np.where(s==t, "Si", "No")
```

Out[13]:

```
array([['No', 'No', 'Si'],
       ['No', 'Si', 'Si']], dtype='<U2')
```

In [14]:

```
np.where(s==t, 1, 0)
```

Out[14]:

```
array([[0, 0, 1],
       [0, 1, 1]])
```

Producto escalar y producto vectorial de 2 vectores

In [22]:

```
w = np.array([1, 2, 3])
w
```

Out[22]:

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

In [23]:

```
x = np.array([2, 5, -4])
x
```

Out[23]:

```
array([ 2,  5, -4])
```

Producto escalar:

In [24]:

```
# w * x = ((1*2) + (2*5) + (3*-4))
```

In [25]:

```
# np.dot(matriz1, matriz2)
np.dot(w,x)
```

Out[25]:

```
0
```

Producto Vectorial:

In [ ]:

```
## Producto Vectorial

# i j k
# 1 2 3
# 2 5 -4

# y se opera:
# -8i+5K+6j -(-4k-4j+15i) = -23i+10j+1k --> (-23, 10, 1)
```

In [26]:

```
np.cross(w, x)
```

Out[26]:

```
array([-23,  10,   1])
```

Concatenación de arrays

Crear los arrays

In [27]:

```
y = np.array([
    [1, 2],
    [3, 4]
])
y
```

Out[27]:

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

In [29]:

```
z = np.array([
    [5, 6]
])
```

Concatenación por filas

In [30]:

```
np.concatenate((y,z), axis=0)
```

Out[30]:

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Concatenación por colunmas

In [31]:

```
z1 = z.transpose()
z1
```

Out[31]:

```
array([[5],
       [6]])
```

In [32]:

```
np.concatenate((y,z1), axis=1)
```

Out[32]:

```
array([[1, 2, 5],
       [3, 4, 6]])
```

Matriz con "matrix"

In [33]:

```
# 4 filas 4 columnas
u = np.matrix([
    [4, -3, 11, 1],
    [5, 9, 7, 2],
    [2, 3, 4, 1],
    [5, 3, -5, -9]
])
u
```

Out[33]:

```
matrix([[ 4, -3, 11,  1],
        [ 5,  9,  7,  2],
        [ 2,  3,  4,  1],
        [ 5,  3, -5, -9]])
```

In [34]:

```
# 1 fila y 4 columnas
v = np.matrix([4, 9, 1, 3])
v
```

Out[34]:

```
matrix([[4, 9, 1, 3]])
```

Suma

In [35]:

```
u + v
```

Out[35]:

```
matrix([[ 8,  6, 12,  4],
        [ 9, 18,  8,  5],
        [ 6, 12,  5,  4],
        [ 9, 12, -4, -6]])
```

Resta

In [37]:

```
u - v
```

Out[37]:

```
matrix([[  0, -12,  10,  -2],
        [  1,   0,   6,  -1],
        [ -2,  -6,   3,  -2],
        [  1,  -6,  -6, -12]])
```

Producto

In [38]:
```
u * v
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-af2d39225de8> in <module>
----> 1 u * v

~/.local/lib/python3.8/site-packages/numpy/matrixlib/defmatrix.py in __mul__(self, other)
    216         if isinstance(other, (N.ndarray, list, tuple)) :
    217             # This promotes 1-D vectors to row vectors
--> 218             return N.dot(self, asmatrix(other))
    219         if isscalar(other) or not hasattr(other, '__rmul__') :
    220             return N.dot(self, other)

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (4,4) and (1,4) not aligned: 4 (dim 1) != 1 (dim 0)
```

In [39]:
```
# ValueError --> es necesario realizar la transpuesta para este caso, ya que las dimensiones No son las adecua
```

Opción 1:

In [40]:
```
u*v.transpose()
```

Out[40]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Opción 2:

In [42]:
```
u*v.T
```

Out[42]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Opción 3:

In [43]:
```
np.dot(u, v.T)
```

Out[43]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Traza de una matriz

(suma de los elementos de la diagonal principal)

In [44]:
```
u -v
```

Out[44]:
```
matrix([[  0, -12,  10,  -2],
        [  1,   0,   6,  -1],
        [ -2,  -6,   3,  -2],
        [  1,  -6,  -6, -12]])
```

In [45]:
```
type(u-v)
```

Out[45]:
```
numpy.matrix
```

In [47]:
```
np.trace(u-v) # 0 + 0 + 3 + (-12) = -9 (suma de los elementos de la diagonal principal)
```

Out[47]:
```
-9
```

Creado por:

Isabel Maniega