T14_Numpy

January 30, 2021

1 Python de cero a experto

Autor: Luis Miguel de la Cruz Salas

1.1 Numpy

Es una biblioteca de Python que permite crear y gestionar arreglos multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel que operan sobre estos arreglos. El sitio oficial es https://numpy.org/

Para usar todas las herramientas de numpy debemos importar la biblioteca como sigue:

[1]:
```
import numpy as np
np.version.version
```

[1]:
```
'1.19.2'
```

[2]:
```
# Función para obtener los atributos de arreglos
info_array = lambda x: print(f' tipo  : {type(x)} \n dtype : {x.dtype} \n dim  ␣
 ↪: {x.ndim} \n shape : {x.shape} \n size(bytes) : {x.itemsize} \n␣
 ↪size(elements) : {x.size}')
```

1.1.1 Creación de arreglos simples

Ejemplo 1. Crear un arreglo de números del 1 al 10 usando: `np.array`, `np.arange`, `np.linspace`, `np.zeros`, `np.ones`, `np.random.rand`

[3]:
```
x = np.array([1,2,3,4,5,6,7,8,9,10])
x
```

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

[4]:
```
info_array(x)
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 1
```

```
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[5]:
```
x = np.arange(10)
x
```

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

[6]:
```
x = np.arange(1,11,1)
x
```

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

[7]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : int64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

Ojo `np.arange()` acepta parámetros flotantes:

[8]:
```
xf = np.arange(1, 11, 1.)
xf
```

[8]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

[9]:
```
xf = np.arange(0.3, 0.7, 0.12)
xf
```

[9]:
```
array([0.3 , 0.42, 0.54, 0.66])
```

[10]:
```
x = np.linspace(1,10,10)
x
```

[10]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

[11]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

Ojo: con `np.linspace` es posible generar un número exacto de elementos, por ejemplo:

[12]:
```
xf = np.linspace(0.3, 0.7, 6)
xf
```

[12]:
```
array([0.3 , 0.38, 0.46, 0.54, 0.62, 0.7 ])
```

[13]:
```
x = np.zeros(10)
x
```

[13]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[14]:
```
for i,val in enumerate(x):
    x[i] = i+1
x
```

[14]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

[15]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[16]:
```
x = np.ones(10)
x
```

[16]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[17]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[18]:
```
for i,val in enumerate(x):
    x[i] = i+val
x
```

[18]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

[19]:
```
x *= 2
x
```

[19]:
```
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20.])
```

[20]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[21]:
```
x = np.random.rand(10)
info_array(x)
x
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[21]:
```
array([0.07002317, 0.36936648, 0.416455  , 0.15678583, 0.04222058,
       0.21495417, 0.21770447, 0.59670632, 0.25927431, 0.97652473])
```

[22]:
```
x = np.random.rand(2,5)
info_array(x)
x
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 2
shape : (2, 5)
size(bytes) : 8
size(elements) : 10
```

[22]:
```
array([[0.7694518 , 0.59450093, 0.82848694, 0.96239847, 0.78422521],
       [0.75219948, 0.25494304, 0.50426336, 0.79623103, 0.38727812]])
```

1.1.2 Modificar el tipo de dato de los elementos del arreglo

[23]:
```
x = np.linspace(1,10,10)
# La modificación afecta al arreglo 'y' pero no al arreglo 'x' (no es inplace)
y = x.astype(int)
```

[24]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[25]:
```
info_array(y)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : int64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[26]:
```
print(id(x), id(y))
```

```
140704685267184 140704685267024
```

[27]:
```
print(x)
print(y)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 1  2  3  4  5  6  7  8  9 10]
```

1.1.3 Arreglos multidimensionales

[28]:
```
x = np.array([[1,2.0],[0,0],(1+1j,3.)])
x
```

[28]:
```
array([[1.+0.j, 2.+0.j],
       [0.+0.j, 0.+0.j],
       [1.+1.j, 3.+0.j]])
```

[29]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : complex128
dim   : 2
shape : (3, 2)
size(bytes) : 16
size(elements) : 6
```

[30]:
```
x = np.array( [ [1,2], [3,4] ], dtype=complex )
x
```

[30]:
```
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
```

[31]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : complex128
dim   : 2
shape : (2, 2)
size(bytes) : 16
size(elements) : 4
```

[32]:
```
x = np.array( [ [[1,2], [3,4]], [[5,6], [7,8]] ])
x
```

[32]:
```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

[33]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : int64
dim   : 3
shape : (2, 2, 2)
size(bytes) : 8
size(elements) : 8
```

[34]:
```
x = np.zeros((10,10))
x
```

[34]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

[35]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
```

```
dim   : 2
shape : (10, 10)
size(bytes) : 8
size(elements) : 100
```

[36]:
```
x = np.ones((4,3,2))
x
```

[36]:
```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

[37]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 3
shape : (4, 3, 2)
size(bytes) : 8
size(elements) : 24
```

[38]:
```
x = np.empty((2,3,4))
x
```

[38]:
```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

[39]:
```
info_array(x)
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
```

```
 dim   : 3
 shape : (2, 3, 4)
 size(bytes) : 8
 size(elements) : 24
```

1.1.4 Cambiando el shape de los arreglos

Función reshape

[40]:
```
x = np.array([ [[ 1, 2, 3, 4],
                [ 5, 6, 7, 8],
                [ 9,10,11,12]],
               [[13,14,15,16],
                [17,16,19,20],
                [21,22,23,24]] ])
info_array(x)
print(f'x = \n {x}')
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 3
 shape : (2, 3, 4)
 size(bytes) : 8
 size(elements) : 24
x =
 [[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 16 19 20]
  [21 22 23 24]]]
```

[41]:
```
y = x.reshape(6,4)
info_array(y)
print(f'y = \n {y}')
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 2
 shape : (6, 4)
 size(bytes) : 8
 size(elements) : 24
y =
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 16 19 20]
```

```
 [21 22 23 24]]
```

[42]:
```
info_array(x)
print(f'x = \n {x} \n')
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 3
 shape : (2, 3, 4)
 size(bytes) : 8
 size(elements) : 24
x =
 [[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 16 19 20]
  [21 22 23 24]]]
```

[43]:
```
y = x.reshape(24)
info_array(y)
print(f'y = \n {y}')
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 1
 shape : (24,)
 size(bytes) : 8
 size(elements) : 24
y =
 [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 16 19 20 21 22 23 24]
```

[44]:
```
y = x.reshape(2,3,4)
info_array(y)
print(f'y = \n {y}')
print(f'x = \n {x}')
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 3
 shape : (2, 3, 4)
 size(bytes) : 8
 size(elements) : 24
y =
 [[[ 1  2  3  4]
  [ 5  6  7  8]
```

```
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 16 19 20]
  [21 22 23 24]]]
x =
 [[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 16 19 20]
  [21 22 23 24]]]
```

[45]:
```
# Otra manera
np.reshape(x, (2,3,4))
```

[45]:
```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 16, 19, 20],
        [21, 22, 23, 24]]])
```

Atributo shape (inplace)

[46]:
```
y.shape
```

[46]:
```
(2, 3, 4)
```

[47]:
```
y.shape = (6,4)
y.shape
```

[47]:
```
(6, 4)
```

[48]:
```
info_array(y)
print(f'y = \n {y}')
```

```
 tipo  : <class 'numpy.ndarray'>
 dtype : int64
 dim   : 2
 shape : (6, 4)
 size(bytes) : 8
 size(elements) : 24
y =
 [[ 1  2  3  4]
 [ 5  6  7  8]
```

```
 [ 9 10 11 12]
 [13 14 15 16]
 [17 16 19 20]
 [21 22 23 24]]
```

Creando un arreglo y modificando su shape al vuelo

[49]:
```
x = np.arange(24).reshape(2,3,4)
x
```

[49]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[50]:
```
x = np.arange(1,25,1).reshape(2,3,4)
info_array(x)
x
```

```
tipo  : <class 'numpy.ndarray'>
dtype : int64
dim   : 3
shape : (2, 3, 4)
size(bytes) : 8
size(elements) : 24
```

[50]:
```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
```

1.1.5 Copias y vistas de arreglos

[51]:
```
x = np.array([1,2,3,4])
z = x  # z es un sinónimo de x, no se crea una copia!
print(id(z), id(x))
print(z is x)
print(x is z)
```

```
140704685349904 140704685349904
True
True
```

Los objetos que son *mutables* se pasan por referencia a una función:

[52]:
```
def f(a):
    print(id(a))

print(id(x))
print(f(x))
```

```
140704685349904
140704685349904
None
```

Copia superficial o vista de un arreglo

[53]:
```
z = x.view()
print(id(z), id(x))
print(z is x)
print(x is z)
print(z.base is x) # Comparten la memoria
print(z.flags.owndata) # Propiedades de la memoria
print(x.flags.owndata) # Propiedades de la memoria
```

```
140704685348864 140704685349904
False
False
True
False
True
```

[54]:
```
print(z.flags)
```

```
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : False
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

[55]:
```
z.shape =(2,2)
print(z.shape, z, sep = '\n')
print(x.shape, x, sep = '\n')
```

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

[56]:
```
z[1,1] = 1000
print(z.shape, z, sep = '\n')
print(x.shape, x, sep = '\n')
```

```
(2, 2)
[[   1    2]
 [   3 1000]]
(4,)
[   1    2    3 1000]
```

Copia completa de arreglos

[57]:
```
z = x.copy()
print(id(z), id(x))
print(z is x)
print(x is z)
print(z.base is x) # Comparten la memoria
print(z.flags.owndata) # Propiedades de la memoria
print(x.flags.owndata) # Propiedades de la memoria
```

```
140704685356976 140704685349904
False
False
False
True
True
```

[58]:
```
print('z = ', z)
print('x = ', x)
```

```
z =  [   1    2    3 1000]
x =  [   1    2    3 1000]
```

[59]:
```
z[3] = 4
print('z = ', z)
print('x = ', x)
```

```
z =  [1 2 3 4]
x =  [   1    2    3 1000]
```

Las rebanadas son vistas de arreglos Las vistas de arreglos pueden ser útiles en ciertos casos, por ejemplo si tenemos un arreglo muy grande y solo deseamos mantener unos cuantos elementos del mismo, debemos hacer lo siguiente:

[60]:
```
a = np.arange(int(1e5)) # Arreglo de 100000 elementos
b = a[:200].copy()      # Copia completa de 200 elementos de 'a'
del a                   # Eliminar la memoria que usa 'a'
b
```

[60]:
```
array([  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,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
       169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
       182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
       195, 196, 197, 198, 199])
```

Pero si usamos rebanadas, el comportamiento es distinto:

[61]:
```
a = np.arange(int(1e5)) # Arreglo de 100000 elementos
b = a[:200]             # Vista de 200 elementos de 'a'
b[0] = 1000
print('b = ', b)
print('a = ', a)
```

```
b =  [1000    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   48   49   50   51   52   53   54   55
   56   57   58   59   60   61   62   63   64   65   66   67   68   69
   70   71   72   73   74   75   76   77   78   79   80   81   82   83
   84   85   86   87   88   89   90   91   92   93   94   95   96   97
   98   99  100  101  102  103  104  105  106  107  108  109  110  111
  112  113  114  115  116  117  118  119  120  121  122  123  124  125
  126  127  128  129  130  131  132  133  134  135  136  137  138  139
  140  141  142  143  144  145  146  147  148  149  150  151  152  153
  154  155  156  157  158  159  160  161  162  163  164  165  166  167
  168  169  170  171  172  173  174  175  176  177  178  179  180  181
  182  183  184  185  186  187  188  189  190  191  192  193  194  195
  196  197  198  199]
a =  [ 1000     1     2 … 99997 99998 99999]
```

1.1.6 Rebanadas (slicing)

[62]:
```
x = np.arange(0,10,1.)
x
```

[62]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

[63]:
```
x[:] # El arreglo completo
```

[63]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

[64]:
```
x[3:6] # Una sección del arreglo, de 3 a 5
```

[64]:
```
array([3., 4., 5.])
```

[65]:
```
x[2:9:2] # de 2 a 8, dando saltos de 2 en 2
```

[65]:
```
array([2., 4., 6., 8.])
```

[66]:
```
x[1:7:2] = 100 # modificando algunos elementos del arreglo
x
```

[66]:
```
array([  0., 100.,   2., 100.,   4., 100.,   6.,   7.,   8.,   9.])
```

[67]:
```
y = np.arange(36).reshape(6,6)
y
```

[67]:
```
array([[ 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]])
```

[68]:
```
y[1:4,:] # renglones de 1 a 3
```

[68]:
```
array([[ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])
```

[69]:
```
y[:,1:5] # columnas de 1 a 4
```

[69]:
```
array([[ 1,  2,  3,  4],
       [ 7,  8,  9, 10],
       [13, 14, 15, 16],
       [19, 20, 21, 22],
       [25, 26, 27, 28],
       [31, 32, 33, 34]])
```

[70]:
```
y[2:4,2:5] # seccion del arreglo
```

[70]:
```
array([[14, 15, 16],
       [20, 21, 22]])
```

[71]:
```
y[1:5:2,1:5:2] # sección del arreglo con saltos distintos de 1
```

[71]:
```
array([[ 7,  9],
       [19, 21]])
```

[72]:
```
y[1:5:2,1:5:2] = 0
y
```

[72]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  0,  8,  0, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18,  0, 20,  0, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

También es posible seleccionar elementos que cumplan cierto criterio.

[73]:
```
y[y<25] # Selecciona los elementos del arreglo que son menores que 25
```

[73]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  0,  8,  0, 10, 11, 12, 13, 14, 15, 16,
       17, 18,  0, 20,  0, 22, 23, 24])
```

[74]:
```
y[y%2==0] # Selecciona todos los elementos pares
```

[74]:
```
array([ 0,  2,  4,  6,  0,  8,  0, 10, 12, 14, 16, 18,  0, 20,  0, 22, 24,
       26, 28, 30, 32, 34])
```

[75]:
```
y[(y>8) & (y<20)] # Selecciona todos los elementos mayores que 8 y menores que␣
↪20
```

[75]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18])
```

[76]:
```
y[(y>8) & (y<20)] = 666
y
```

[76]:
```
array([[  0,   1,   2,   3,   4,   5],
       [  6,   0,   8,   0, 666, 666],
       [666, 666, 666, 666, 666, 666],
       [666,   0,  20,   0,  22,  23],
       [ 24,  25,  26,  27,  28,  29],
       [ 30,  31,  32,  33,  34,  35]])
```

[77]:
```
z = np.nonzero(y == 666) # Determina los renglones y las columnas donde se␣
↪cumple la condición.
z
```

[77]:
```
(array([1, 1, 2, 2, 2, 2, 2, 2, 3]), array([4, 5, 0, 1, 2, 3, 4, 5, 0]))
```

[78]:
```
indices = list(zip(z[0], z[1])) # Genera una lista de coordenadas donde se cumple la condición.
indices
```

[78]:
```
[(1, 4), (1, 5), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 0)]
```

[79]:
```
print(y[z]) # Imprime los elementos del arreglo 'y' usando las coordenadas de 'z'
```

```
[666 666 666 666 666 666 666 666 666]
```

1.1.7 Operaciones básicas entre arreglos

[80]:
```
v1 = np.array([2.3,3.1,9.6])
v2 = np.array([3.4,5.6,7.8])
```

[81]:
```
(1/3)*v1 # Escalar por arreglo
```

[81]:
```
array([0.76666667, 1.03333333, 3.2       ])
```

[82]:
```
v1+v2 # Suma de arreglos
```

[82]:
```
array([ 5.7,  8.7, 17.4])
```

[83]:
```
v1-v2 # Resta de arreglos
```

[83]:
```
array([-1.1, -2.5,  1.8])
```

[84]:
```
v1*v2 # Multiplicación elemento a elemento
```

[84]:
```
array([ 7.82, 17.36, 74.88])
```

[85]:
```
v1/v2 # División elemento a elemento
```

[85]:
```
array([0.67647059, 0.55357143, 1.23076923])
```

[86]:
```
v1 ** 2 # Potencia de un arreglo
```

[86]:
```
array([ 5.29,  9.61, 92.16])
```

[87]:
```
v1 % 2  # Modulo de un arreglo
```

[87]:
```
array([0.3, 1.1, 1.6])
```

[88]:
```
10 * np.sin(v1) # Aplicación de una función matemática a cada elemento del arreglo
```

[88]:
```
array([ 7.45705212,  0.41580662, -1.74326781])
```

[89]:
```
v1 > 3 # Operación de comparación, devuelve un arreglo Booleano
```

[89]:
```
array([False,  True,  True])
```

1.1.8 Operaciones entre arreglos Booleanos

[90]:
```
f = np.array([True, False, False, True])
r = np.array([False, True, False, True])
```

[91]:
```
f & r
```

[91]:
```
array([False, False, False,  True])
```

[92]:
```
f | r
```

[92]:
```
array([ True,  True, False,  True])
```

[93]:
```
~f
```

[93]:
```
array([False,  True,  True, False])
```

[94]:
```
b = np.arange(4)
b
```

[94]:
```
array([0, 1, 2, 3])
```

[95]:
```
b[f]
```

[95]:
```
array([0, 3])
```

[96]:
```
b[f] = 100
b
```

[96]:
```
array([100,   1,   2, 100])
```

1.1.9 Métodos de los arreglos

Existe una larga lista de métodos definidas para los arreglos, vea más información aquí.

[97]:
```
x = np.random.random(100) # arreglo de 100 números aleatorios entre 1 y 0
x
```

[97]:
```
array([0.85625036, 0.50539065, 0.88239252, 0.09338146, 0.34664493,
       0.29279555, 0.50460183, 0.5609502 , 0.40634518, 0.46953224,
       0.85486939, 0.64442876, 0.37226585, 0.68941874, 0.28695561,
```

```
       0.17592572, 0.16393506, 0.35950818, 0.99388413, 0.58195126,
       0.26085354, 0.66680653, 0.2713812 , 0.27355876, 0.84098559,
       0.62635967, 0.64460188, 0.13632262, 0.24998947, 0.302774  ,
       0.06855619, 0.00805434, 0.59477011, 0.45280032, 0.51291593,
       0.3174877 , 0.53378935, 0.1692721 , 0.86249364, 0.72266603,
       0.25692481, 0.85649479, 0.62100091, 0.71940878, 0.99893656,
       0.51833782, 0.35898922, 0.08743599, 0.53307403, 0.52676193,
       0.37164367, 0.83310402, 0.37916705, 0.47951049, 0.83065503,
       0.97591968, 0.51217404, 0.89493185, 0.50956744, 0.79483729,
       0.86111319, 0.37894106, 0.1381868 , 0.63363595, 0.01050854,
       0.75415779, 0.87098714, 0.88112281, 0.22749475, 0.5088999 ,
       0.40583075, 0.28834272, 0.98852758, 0.82901189, 0.74559152,
       0.4281683 , 0.8597571 , 0.79139404, 0.78267197, 0.10195268,
       0.21694582, 0.74100873, 0.56562619, 0.99644958, 0.63128338,
       0.89569805, 0.81094901, 0.73498545, 0.20229809, 0.76306927,
       0.1674949 , 0.20339313, 0.61351661, 0.47949596, 0.56742726,
       0.57247437, 0.84728614, 0.10513187, 0.57501568, 0.38616955])
```

[98]:
```
x.max()
```

[98]:
```
0.9989365621026909
```

[99]:
```
x.sum()
```

[99]:
```
53.17875748931071
```

[100]:
```
x = np.arange(10).reshape(2,5)
x
```

[100]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

[101]:
```
x.T
```

[101]:
```
array([[0, 5],
       [1, 6],
       [2, 7],
       [3, 8],
       [4, 9]])
```

[102]:
```
x.transpose()
```

[102]:
```
array([[0, 5],
       [1, 6],
       [2, 7],
       [3, 8],
       [4, 9]])
```

[103]:
```
np.transpose(x)
```

[103]:
```
array([[0, 5],
       [1, 6],
       [2, 7],
       [3, 8],
       [4, 9]])
```

[104]:
```
np.flip(x) # Cambiar el orden de los elementos del arreglo
```

[104]:
```
array([[9, 8, 7, 6, 5],
       [4, 3, 2, 1, 0]])
```

[105]:
```
np.flip(x, axis=0)
```

[105]:
```
array([[5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4]])
```

[106]:
```
f1 = x.flatten() # Aplanar un arreglo
f1[0] = 1000
print(x)
print(f1)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[1000    1    2    3    4    5    6    7    8    9]
```

[107]:
```
f2 = x.ravel() # Aplanar un arreglo
f2[0] = 1000
print(x)
print(f1)
```

```
[[1000    1    2    3    4]
 [   5    6    7    8    9]]
[1000    1    2    3    4    5    6    7    8    9]
```

Los arreglos deben ser compatibles para poder realizar las operaciones anteriores:

[108]:
```
a = np.arange(24).reshape(2,3,4)
b = np.arange(24).reshape(2,3,4)
a + b
```

[108]:
```
array([[[ 0,  2,  4,  6],
        [ 8, 10, 12, 14],
        [16, 18, 20, 22]],

       [[24, 26, 28, 30],
        [32, 34, 36, 38],
```

```
        [40, 42, 44, 46]]])
```

[109]:
```
c = np.arange(24).reshape(6,4)
a + c
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-109-84a41d0e3fbd> in <module>
      1 c = np.arange(24).reshape(6,4)
----> 2 a + c

ValueError: operands could not be broadcast together with shapes (2,3,4) (6,4)
```

1.1.10 Apilación y concatenación de arreglos

[110]:
```
a = np.arange(4).reshape(2,2)
b = np.arange(4,8,1).reshape(2,2)
print(a)
print(b)
```

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

[111]:
```
np.vstack( (a, b) ) # Apilación vertical
```

[111]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

[112]:
```
np.hstack( (a, b) ) # Apilación horizontal
```

[112]:
```
array([[0, 1, 4, 5],
       [2, 3, 6, 7]])
```

[113]:
```
x = np.arange(1,25,1).reshape(6,4)
x
```

[113]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

[114]:
```
np.hsplit(x, 2) # División vertical en dos arreglos
```

[114]:
```
[array([[ 1,  2],
        [ 5,  6],
        [ 9, 10],
        [13, 14],
        [17, 18],
        [21, 22]]),
 array([[ 3,  4],
        [ 7,  8],
        [11, 12],
        [15, 16],
        [19, 20],
        [23, 24]])]
```

[115]:
```
np.vsplit(x, 2) #  División horizontal en dos arreglos
```

[115]:
```
[array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]),
 array([[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]])]
```

Se recomienda revisar la función `np.concatenate` para ver más opciones

1.1.11 Agregando dimensiones al arreglo

[116]:
```
x = np.arange(1,11,1.)
info_array(x)
x
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 1
shape : (10,)
size(bytes) : 8
size(elements) : 10
```

[116]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

[117]:
```
x_row = x[np.newaxis, :]
info_array(x_row)
x_row
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 2
```

```
shape : (1, 10)
size(bytes) : 8
size(elements) : 10
```

[117]:
```
array([[ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]])
```

[118]:
```
x_col = x[:, np.newaxis]
info_array(x_col)
x_col
```

```
tipo  : <class 'numpy.ndarray'>
dtype : float64
dim   : 2
shape : (10, 1)
size(bytes) : 8
size(elements) : 10
```

[118]:
```
array([[ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.],
       [ 6.],
       [ 7.],
       [ 8.],
       [ 9.],
       [10.]])
```

[119]:
```
# Otra manera
x_row = np.expand_dims(x, axis=0)
x_col = np.expand_dims(x, axis=1)
print(x_row)
print(x_col)
```

```
[[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]]
[[ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]
 [ 6.]
 [ 7.]
 [ 8.]
 [ 9.]
 [10.]]
```

1.1.12 Constantes

[120]:
```
np.e
```

[120]:
```
2.718281828459045
```

[121]:
```
np.euler_gamma # Euler-Mascheroni constant
```

[121]:
```
0.5772156649015329
```

[122]:
```
np.pi
```

[122]:
```
3.141592653589793
```

[123]:
```
np.inf # Infinito
```

[123]:
```
inf
```

[124]:
```
#Por ejemplo
np.array([1]) / 0.
```

```
<ipython-input-124-616fdfdd880a>:2: RuntimeWarning: divide by zero encountered
in true_divide
  np.array([1]) / 0.
```

[124]:
```
array([inf])
```

[125]:
```
np.nan # Not a Number: Valor no definido o no representable
```

[125]:
```
nan
```

[126]:
```
# Por ejemplo
np.sqrt(-1)
```

```
<ipython-input-126-636e7e27cf34>:2: RuntimeWarning: invalid value encountered in
sqrt
  np.sqrt(-1)
```

[126]:
```
nan
```

[127]:
```
np.log([-1, 1, 2])
```

```
<ipython-input-127-a90bd0729868>:1: RuntimeWarning: invalid value encountered in
log
  np.log([-1, 1, 2])
```

[127]:
```
array([       nan, 0.        , 0.69314718])
```

[128]:
```
np.NINF  # Infinito negativo
```

[128]:
```
-inf
```

[129]:
```
# Por ejemplo
np.array([-1]) / 0.
```

```
<ipython-input-129-48a02a3d3ded>:2: RuntimeWarning: divide by zero encountered
in true_divide
  np.array([-1]) / 0.
```

[129]:
```
array([-inf])
```

[130]:
```
np.NZERO # Cero negativo
```

[130]:
```
-0.0
```

[131]:
```
np.PZERO # Cero positivo
```

[131]:
```
0.0
```

1.1.13 Exportando e importando arreglos a archivos

[132]:
```
x = np.arange(1,25,1.0).reshape(6,4)
print(x)
np.savetxt('arreglo.csv', x, fmt='%.2f', delimiter=',', header='1,  2,  3,  4')
```

```
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]
 [13. 14. 15. 16.]
 [17. 18. 19. 20.]
 [21. 22. 23. 24.]]
```

[133]:
```
#Usando la biblioteca Pandas
import pandas as pd
df = pd.DataFrame(x)
df
```

[133]:

	0	1	2	3
0	1.0	2.0	3.0	4.0
1	5.0	6.0	7.0	8.0
2	9.0	10.0	11.0	12.0
3	13.0	14.0	15.0	16.0
4	17.0	18.0	19.0	20.0
5	21.0	22.0	23.0	24.0

[134]:
```
df.to_csv('arreglo_PD.csv')
```

[135]:
```
y = pd.read_csv('arreglo_PD.csv')
y
```

[135]:

	Unnamed: 0	0	1	2	3
0	0	1.0	2.0	3.0	4.0
1	1	5.0	6.0	7.0	8.0
2	2	9.0	10.0	11.0	12.0
3	3	13.0	14.0	15.0	16.0
4	4	17.0	18.0	19.0	20.0
5	5	21.0	22.0	23.0	24.0

[ ]:
```

```