NUMPY_ORIGINAL

June 20, 2021

1 NUMPY BY DUSTIN

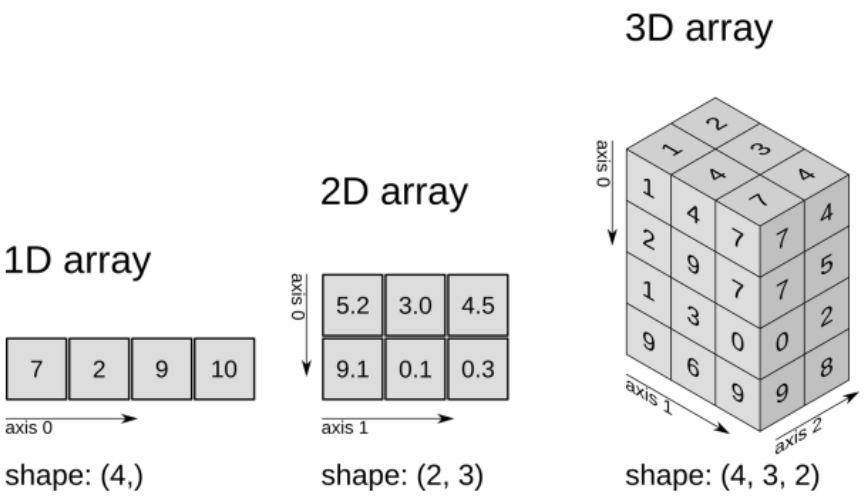
[2]:
```
import numpy as np
```

1.1 CREATING ARRAYS

[3]:
```
print(np.__version__)
```

```
1.20.1
```

[1]:
```
from IPython.display import display, Image
display(Image(filename="arrays.png"))
```



[11]:
```
a = np.array([1, 2, 3, 4, 5])
b = np.array((1,2,3,4,5))
print(a==b)
print(type(a))
```

```
[ True  True  True  True  True]
<class 'numpy.ndarray'>
```

[102]:
```
#cada dimension debe tener un numero igual de elementos en cada subset

#1D
a = np.array([1,2,3,4,5])
#2D
b = np.array([[1, 2, 3], [4, 5, 6]])
#3D
c = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

a.ndim
b.ndim
c.ndim
```

[102]:
```
3
```

[106]:
```
#establecer el numero de dimensiones
np.array([1,2,3,4,5],ndmin=5)
```

[106]:
```
array([[[[[1, 2, 3, 4, 5]]]]])
```

[93]:
```
np.arange(1,50)
```

[93]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
       35, 37, 39, 41, 43, 45, 47, 49])
```

[58]:
```
np.zeros((3,4))
```

[58]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[39]:
```
a = np.ones((3,4),dtype= "i")
print(a)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

[41]:
```
np.linspace(0,2,8)  #rango de 0 a 2 con 8 valores
```

[41]:
```
array([0.        , 0.28571429, 0.57142857, 0.85714286, 1.14285714,
       1.42857143, 1.71428571, 2.        ])
```

[66]:
```
np.full((2,2),7)
```

[66]:
```
array([[7, 7],
       [7, 7]])
```

[74]:
```
np.diag([1,2,3])
```

[74]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

[75]:
```
np.eye(2)
```

[75]:
```
array([[1., 0.],
       [0., 1.]])
```

[78]:
```
np.random.random((2,2))
```

[78]:
```
array([[0.33521563, 0.34343237],
       [0.51881228, 0.46389614]])
```

[90]:
```
np.empty((2,3))
```

[90]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

1.2 SAVIND AND LOADING DATA

[210]:
```
#working with
a = np.array(["dustin", "leona","buenda"])
np.save("my_file",a) #formato npy
```

[212]:
```
np.savez("my_file22",a) #formato npz
```

[112]:
```
np.load("my_file.npy")
```

[112]:
```
array(['dustin', 'leona', 'buenda'], dtype='<U6')
```

[216]:
```
#working with text and csv files
a = np.array([[1,2,3,4,5,6],[1,2,3,4,5,6]],dtype= int)
np.savetxt("my_files.txt", a,delimiter=" ")
```

[135]:
```
np.loadtxt("my_files.txt")
```

[135]:
```
array([[1., 2., 3., 4., 5., 6.],
       [1., 2., 3., 4., 5., 6.]])
```

[42]:
```
# as you can see %d will truncate to integer, %s will maintain formatting, %f␣
 ↪will print as float and %g is used for generic number
a = np.array([[11, 12, 13, 22], [21, 7, 23, 14], [31, 10, 33, 7]])
```

```
np.savetxt('my_files4.csv', a, delimiter=',', fmt='%d')
```

[60]:
```
import numpy as np
my_data = np.genfromtxt('Salary.csv', delimiter=',')

my_data
```

[60]:
```
array([[1.10000e+00, 3.93430e+04],
       [1.30000e+00, 4.62050e+04],
       [1.50000e+00, 3.77310e+04],
       [2.00000e+00, 4.35250e+04],
       [2.20000e+00, 3.98910e+04],
       [2.90000e+00, 5.66420e+04],
       [3.00000e+00, 6.01500e+04],
       [3.20000e+00, 5.44450e+04],
       [3.20000e+00, 6.44450e+04],
       [3.70000e+00, 5.71890e+04],
       [3.90000e+00, 6.32180e+04],
       [4.00000e+00, 5.57940e+04],
       [4.00000e+00, 5.69570e+04],
       [4.10000e+00, 5.70810e+04],
       [4.50000e+00, 6.11110e+04],
       [4.90000e+00, 6.79380e+04],
       [5.10000e+00, 6.60290e+04],
       [5.30000e+00, 8.30880e+04],
       [5.90000e+00, 8.13630e+04],
       [6.00000e+00, 9.39400e+04],
       [6.80000e+00, 9.17380e+04],
       [7.10000e+00, 9.82730e+04],
       [7.90000e+00, 1.01302e+05],
       [8.20000e+00, 1.13812e+05],
       [8.70000e+00, 1.09431e+05],
       [9.00000e+00, 1.05582e+05],
       [9.50000e+00, 1.16969e+05],
       [9.60000e+00, 1.12635e+05],
       [1.03000e+01, 1.22391e+05],
       [1.05000e+01, 1.21872e+05]])
```

1.3 INSPECTING YOUR ARRAY

[22]:
```
a = np.array([1,2,3])
b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
c = np.array([[(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6),]], dtype = int)
print(c)
```

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

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

[16]:
```
b.shape
c.ndim
```

[16]: 3

[68]:
```
len(a)
```

[68]: 3

[69]:
```
c.ndim
```

[69]: 3

[73]:
```
b.size
```

[73]: 6

[ ]:
```
"""Data Types in Python
By default Python have these data types:

strings - used to represent text data, the text is given under quote marks. e.g.
 ↪ "ABCD"
integer - used to represent integer numbers. e.g. -1, -2, -3
float - used to represent real numbers. e.g. 1.2, 42.42
boolean - used to represent True or False.
complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

NumPy has some extra data types, and refer to data types with one character,␣
 ↪like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent␣
 ↪them.

i - integer
b - boolean
u - unsigned integer
f - float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory for other type ( void )"""
```

[74]:
```
b.dtype
```

[74]:
```
dtype('float64')
```

[77]:
```
b.dtype.name
```

[77]:
```
'float64'
```

[78]:
```
b.astype(int64)
```

[78]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

1.4 ARRAYS MATHEMATICS

[276]:
```
a = np.array([(1.5,2,3)], dtype = float, ndmin=2)
b = np.array([(1,2,3), (9,8,7)], dtype = float)

a - b
```

[276]:
```
array([[ 0.5,  0. ,  0. ],
       [-7.5, -6. , -4. ]])
```

[87]:
```
a + b
```

[87]:
```
array([[ 2.5,  4. ,  6. ],
       [13. , 13. , 13. ]])
```

[88]:
```
a/b
```

[88]:
```
array([[1.5       , 1.        , 1.        ],
       [0.44444444, 0.625     , 0.85714286]])
```

[89]:
```
a*b
```

[89]:
```
array([[ 1.5,  4. ,  9. ],
       [36. , 40. , 42. ]])
```

[97]:
```
np.exp(a)
```

[97]:
```
array([[ 4.48168907,  7.3890561 , 20.08553692]])
```

[98]:
```
np.sqrt(a)
```

[98]:
```
array([[1.22474487, 1.41421356, 1.73205081]])
```

[99]:
```
np.sin(a)
np.cos(a)
np.tan(a)
```

[99]:
```
array([[14.10141995, -2.18503986, -0.14254654]])
```

[100]:
```
np.log(a)
```

[100]:
```
array([[0.40546511, 0.69314718, 1.09861229]])
```

[233]:
```
f = np.array([1,2])
g = np.array([4,5])
### 1*4+2*5
f.dot(g)

"""np.dot is the dot product of two matrices.

|A B| . |E F| = |A*E+B*G A*F+B*H|
|C D|   |G H|   |C*E+D*G C*F+D*H|

Whereas np.multiply does an element-wise multiplication of two matrices.

|A B|   |E F| = |A*E B*F|
|C D|   |G H|   |C*G D*H|"""
```

[233]:
```
'np.dot is the dot product of two matrices.\n\n|A B| . |E F| = |A*E+B*G
A*F+B*H|\n|C D|   |G H|   |C*E+D*G C*F+D*H|\n\nWhereas np.multiply does an
element-wise multiplication of two matrices.\n\n|A B|   |E F| = |A*E B*F|\n|C D|
|G H|   |C*G D*H|'
```

1.5 COMPARING

[110]:
```
a = np.array([(1.5,2,3)], dtype = float)
b = np.array([(1.5,2,3)], dtype = float)
c = np.array_equal(a,b)

a == b
```

[110]:
```
array([[ True,  True,  True]])
```

1.6 AGREGATE FUNCTION

[3]:
```
a = np.array([[1.5,2,3],[1,2,3]], dtype = float)
print(a)
```

```
[[1.5 2.  3. ]
 [1.  2.  3. ]]
```

[117]:
```
a.sum()
```

[117]:
```
12.5
```

[119]:
```
a.min()
```

[119]:
```
1.0
```

[120]:
```
a.max()
```

[120]:
```
3.0
```

[127]:
```
a.max(axis = 1) #maximun value from an arrow
```

[127]:
```
array([3., 3.])
```

[9]:
```
a.cumsum()
```

[9]:
```
array([ 1.5,  3.5,  6.5,  7.5,  9.5, 12.5])
```

[8]:
```
a.mean()
```

[8]:
```
2.0833333333333335
```

[145]:
```
np.median(a)
```

[145]:
```
2.0
```

[148]:
```
np.corrcoef(a)
```

[148]:
```
array([[1.        , 0.98198051],
       [0.98198051, 1.        ]])
```

[149]:
```
np.std(a)
```

[149]:
```
0.731247032282677
```

[130]:
```
np.var(a)
```

[130]:
```
0.5347222222222223
```

[231]:
```
a = np.array([1,2,3,np.nan,4,5,6])
print(a)
np.nansum(a)
```

```
[ 1.  2.  3. nan  4.  5.  6.]
```

[231]:
```
array([1., 2., 3., 0., 4., 5., 6.])
```

1.7 Copying and sorting arrays

[11]:
```
a = np.array([[1.5,2,3],[1,2,3]], dtype = int)
print(a)
```

```
[[1 2 3]
 [1 2 3]]
```

[50]:
```
x = np.arange(10)
z = x.copy()  #crea una copia y comparten informacion pero no propiedades
print(z)

z[0] = 10
print(z)
print(x)
```

```
[0 1 2 3 4 5 6 7 8 9]
[10  1  2  3  4  5  6  7  8  9]
[0 1 2 3 4 5 6 7 8 9]
```

[54]:
```
x = np.arange(10)
t = x.view()  #comparte informacion y pero no propiedades, si un elemento es␣
 ↪alterado en x, t no lo muestra
print(t)

t[0] = 10
print(t)
print(x)
```

```
[0 1 2 3 4 5 6 7 8 9]
[10  1  2  3  4  5  6  7  8  9]
[10  1  2  3  4  5  6  7  8  9]
```

[52]:
```
x = np.array([7,6,54,8,7,6,52,9,])
b = x.sort()
print(x)
```

```
[ 6  6  7  7  8  9 52 54]
```

1.8 FILTER

[34]:
```
x = np.arange(51,151)
b = x.reshape(10,2,5)
print(b)
print(b.ndim)
print(b.base)

#-1 en un reshape como valor desconocido que se calcula en numpy, solo en un␣
 ↪parametro
```

```
print(b.reshape(-1)) #1 dimension
```

```
[[[ 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]]]
3
[ 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]
[ 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]
```

[139]:
```
x[3]
```

[139]: 54

[146]:
```
#filtro en 2 dimensiones
x = np.array([[1,2,3,4,5,6],[9,8,7,6,5,4]])
x
#x.ndim
x[0,4]
```

[146]: 5

[156]:
```
#filter en 3 dimensiones
x = np.array([[[1,2,3,4,5,6],[9,8,7,6,5,4],[8,7,6,5,4,3]]])
x[0,0,5]
```

[156]: 6

[163]:
```
y = np.array(range(0,100), ndmin = 2).reshape(20,5)
y[4,0]
```

[163]: 20

[181]:
```
#slicing
x = np.arange(51,151)
x.reshape(20,5)

x[2:30]
```

[181]:
```
array([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])
```

[57]:
```
y = np.array(range(0,100), ndmin = 2).reshape(20,5)
y[2:5,:]
```

[57]:
```
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

[197]:
```
y[y>50]
```

[197]:
```
array([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])
```

[206]:
```
y[[2,3,4], :]
```

[206]:
```
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

[62]:
```
z = np.array(["a", "b", "c"])
for i in np.nditer(z):
    print(i)
```

```
a
b
c
```

[70]:
```
for ind,i in np.ndenumerate(z):
    print("indice",ind,"valor",i)
```

```
indice (0,) valor a
indice (1,) valor b
indice (2,) valor c
```

1.9 MANIPULATION

[215]:
```
y = np.transpose(y)
y.shape
y.T
```

[215]:
```
array([[ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75,
        80, 85, 90, 95],
       [ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76,
        81, 86, 91, 96],
       [ 2,  7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62, 67, 72, 77,
        82, 87, 92, 97],
       [ 3,  8, 13, 18, 23, 28, 33, 38, 43, 48, 53, 58, 63, 68, 73, 78,
        83, 88, 93, 98],
       [ 4,  9, 14, 19, 24, 29, 34, 39, 44, 49, 54, 59, 64, 69, 74, 79,
        84, 89, 94, 99]])
```

[225]:
```
x = np.array([[[1,2,3,4,5,6],[9,8,7,6,5,4],[8,7,6,5,4,3]]])
x.ravel() #aplana las matrices o se puede usar np.reshape(-1)
```

[225]:
```
1
```

[237]:
```
a = np.array(range(0,40)).reshape(10,4)
a.resize((4,10))
print(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]]
```

[237]:
```
2
```

[245]:
```
np.append(a,[1,2,3,4,5,6,7,8])
```

[245]:
```
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,  1,  2,  3,  4,  5,  6,  7,  8])
```

[46]:
```
a = np.append(a,[1,2,3,4,5,6,7,8])
a.astype("i1")
```

[46]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2,
       3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8,
       1, 2, 3, 4, 5, 6, 7, 8], dtype=int8)
```

[241]:
```
np.insert(a,1,3) #inserta con un orden
```

[241]:
```
array([ 0,  3,  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])
```

[242]:
```
np.delete(a,[1])
```

[242]:
```
array([ 0,  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])
```

[98]:
```
#combining
a = np.array([[1.5,2,3],[1,2,3]], dtype = float)
b = np.array([[1.5,2,3],[1,2,3]],ndmin=2)

np.concatenate((a,b), axis=1) #axis = 0
```

[98]:
```
array([[1.5, 2. , 3. , 1.5, 2. , 3. ],
       [1. , 2. , 3. , 1. , 2. , 3. ]])
```

[108]:
```
np.stack((a,b), axis = 2) #sube una dimension se puede usar 0, 1, 2
```

[108]:
```
array([[[1.5, 1.5],
        [2. , 2. ],
        [3. , 3. ]],

       [[1. , 1. ],
        [2. , 2. ],
        [3. , 3. ]]])
```

[255]:
```
np.vstack((a,b))
```

[255]:
```
array([[1.5, 2. , 3. ],
       [1. , 2. , 3. ],
       [1. , 2. , 3. ]])
```

[128]:
```
a = np.array([[1.5,2,3],[1,2,3]], dtype = float)
b = np.array([[1.5,2,3],[9,8,7]],ndmin=2)
np.hstack((a,b))
```

[128]:
```
array([[1.5, 2. , 3. , 1.5, 2. , 3. ],
       [1. , 2. , 3. , 9. , 8. , 7. ]])
```

[261]:
```
np.column_stack((a,b))
```

[261]:
```
array([[1.5, 2. , 3. , 1.5, 2. , 3. ],
       [1. , 2. , 3. , 9. , 8. , 7. ]])
```

[269]:
```
np.c_[a,b]
```

[269]:
```
array([[1.5, 2. , 3. , 1.5, 2. , 3. ],
       [1. , 2. , 3. , 9. , 8. , 7. ]])
```

[121]:
```
np.array_split(a,2)
```

[121]:
```
[array([[1.5, 2. , 3. ]]), array([[1., 2., 3.]])]
```

[129]:
```
#splitting
np.hsplit(a,3)
```

[129]:
```
[array([[1.5],
        [1. ]]),
 array([[2.],
        [2.]]),
 array([[3.],
        [3.]])]
```

[272]:
```
np.vsplit(a,2)
```

[272]:
```
[array([[1.5, 2. , 3. ]]), array([[1., 2., 3.]])]
```