NUMPY

Creando Arreglos

Una sola dimensión

```
In [ ]: a = np.array([4,7,9])
In [ ]: a

Out[ ]: array([4, 7, 9])

In [ ]: a = np.array([4.5,7.32,9.01])
In [ ]: a
Out[ ]: array([4.5 , 7.32, 9.01])
```

Varias dimensiones

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

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

Cambiando tipos de datos

```
In [ ]: a = np.array([33,40,5],dtype=complex)
In [ ]: a
Out[ ]: array([33.+0.j, 40.+0.j, 5.+0.j])

In [ ]: a = np.array([33,40,5],dtype=float)
In [ ]: a
Out[ ]: array([33., 40., 5.])

In [ ]: a = np.array([4.5,7.32,9.01],dtype = int)
In [ ]: a
Out[ ]: array([4, 7, 9])
```

Creando Arreglos - ingreso de datos automáticamente

Zeros

```
In [ ]: c = np.zeros((5,5))
In [ ]: c
Out[ ]:
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.]])

In [ ]: c = np.zeros((5,5), dtype=int)
In [ ]: c
Out[ ]:
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]])
```

Unos

```
In [ ]: d = np.ones((5,5))
In [ ]: d
Out[ ]:
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., 1.]])
```

Vacio

```
In [ ]: g = np.empty((3,3))
In [ ]: g
Out[ ]:
array([[0.0000000e+000, 7.7074241e-321, 2.4703282e-323],
       [1.1595974e-311, 0.0000000e+000, 6.4761631e-319],
       [1.1595982e-311, 0.0000000e+000, 2.4703282e-323]])

In [ ]: h = np.empty((2,2))
In [ ]: h
```

```
Out[ ]:
array([[1.15956883e-311, 8.45599366e-307],
         [1.37962117e-306, 2.37667317e-312]])
```

Llenar con el mismo número

```
In [ ]: b = np.full((3,3),5)
In [ ]: b
Out[ ]:
array([[5, 5, 5],
         [5, 5, 5],
         [5, 5, 5]])
```

Llenar con números aleatorios

```
In [ ]: b = np.random.random((3,3))
In [ ]: b
Out[ ]:
array([[0.67397832, 0.96811363, 0.69040877],
        [0.58196599, 0.73273725, 0.42511847],
        [0.49239954, 0.99404957, 0.37236421]])
```

Matriz idéntica

```
In [ ]: m = np.eye(5,5)
In [ ]: m
Out[ ]:
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.]])

In [ ]: m = np.eye(5,3)
In [ ]: m
Out[ ]:
array([[1., 0., 0.],
         [0., 1., 0.],
         [0., 0., 1.],
         [0., 0., 0.],
         [0., 0., 0.]])

In [ ]: m = np.eye(3,5)
In [ ]: m
Out[ ]:
array([[1., 0., 0., 0., 0.],
      [0., 1., 0., 0., 0.],
     [0., 0., 1., 0., 0.]])
```

Rangos

```
In [ ]: n = np.arange(0,20,2)
In [ ]: n
Out[ ]: array([ 0, 2, 4, 6, 8, 10, 12, 14, 16, 18])
```

```
In [32]: n = np.arange(0,5,0.3)
In [33]: n
Out[33]:
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. , 3.3, 3.6,
3.9, 4.2, 4.5, 4.8])
```

Espacios lineales

```
In [ ]: n = np.linspace(0,4,9)
In [ ]: n
Out[ ]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ])
```

```
In [ ]: x = np.linspace( 0, 2*pi, 20 )
In [ ]: f = np.sin(x)
In [ ]: f
Out[ ]:
array([ 0.00000000e+00, 3.24699469e-01, 6.14212713e-01, 8.37166478e-01,
9.69400266e-01, 9.96584493e-01, 9.15773327e-01, 7.35723911e-01,
4.75947393e-01, 1.64594590e-01, -1.64594590e-01, -4.75947393e-01,
-7.35723911e-01, -9.15773327e-01, -9.96584493e-01, -9.69400266e-01,
-8.37166478e-01, -6.14212713e-01, -3.24699469e-01, -2.44929360e-16])
```

Reformar

```
In [ ]: n = np.arange(20).reshape(5,4)
In [ ]: print(n)
[[ 0 1 2 3]
[ 4 5 6 7]
[ 8 9 10 11]
[12 13 14 15]
[16 17 18 19]]
```

```
In [ ]: n = np.arange(24).reshape(2,4,3)
In [ ]: print(n)
[[[ 0 1 2]
[ 3 4 5]
[ 6 7 8]
[ 9 10 11]]

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

```
In [ ]: a = np.reshape(n,(12,2))
In [ ]: a
Out[ ]:
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]])
```

Redimensionar

```
In [ ]: b = np.resize(a,(4))
In [ ]: b
Out[ ]: array([0, 1, 2, 3])

In [ ]: c = np.resize(b,(2,3))
In [ ]: c
Out[ ]:
array([[0, 1, 2],
[3, 0, 1]])
```

Atributos de los Arreglos

ndarray.ndim
ndarray.shape
ndarray.size
ndarray.dtype
ndarray.itemsize
Ndarray.data

Indexing - Slicing

#1D

```
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

# INDEX 1D

```
print(x[3])
print(x[-2])
```

# SLICING 1D

```
print(x[1:7:2])
print(x[-2:10])
print(x[-3:3:-1])
```

#2D

```
y = np.arange(20).reshape(5,4)
print(y)
```

# INDEX 2D

```
#simple
print(y[2,1])

#double
n = y[[1,2,3],[0,3,1]]
print(n)
```

# SLICING 2D

```
# 2D->1D
print(y[1:2, 1:3])

# 2D->2D
print(y[1:3, 1:3])
```

#3D

```
z = np.arange(40).reshape(2,5,4)
print(z)
```

# INDEX 3D

```
#simple
print(z[1,3,2])


#triple
print(z[0:2,0:3,0:3])
```

TALLER

- Generar un arreglo tridimensional de tamaño 9, 9, 9 con los número enteros del 0 al 728.
- Imaginar que el arreglo se divide en 27 arreglos de 3x3; desarrollar una función que permita intercambiar la posición de dos de estos bloques, pasándole como argumento únicamente la identificación de los bloques a intercambiar (los bloques se deben identificar con números del 0 al 26)