FUNDAMENTOS DE PROGRAMACIÓN

UNIDAD 5:

ARREGLOS N-DIMENSIONALES

Arreglos N-dimensionales

Agrupación de elementos del mismo tipo de dato (homogéneos) y con un tamaño definido.

Dimensión 1: Arreglos

```
[2 3 4]
```

Dimensión 2: Matriz

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

Numpy

Librería de Python que permite realizar operaciones con arreglos n-dimensionales (arreglos y matrices).

```
import numpy as np
```

Creación de arreglos

Creando un arreglo a partir de una lista de elementos:

```
>>> a = np.array([2,5,9])
>>> a
array([2, 5, 9])
>>> a.dtype
dtype('int32')
>>>
>>> b = np.array([4,0.6,3])
>>> b
array([ 4. ,  0.6,  3. ])
>>> b.dtype
dtype('float64')
```

Creación de arreglos

Cuando se crea un arreglo se puede especificar el tipo de dato de los elementos que tendrá el arreglo:

```
>>> a = np.array([1,2,3,5],float)
>>> a
array([ 1.,  2.,  3.,  5.])
>>> b = np.array([1,2,3,5],str)
>>> b
array(['1', '2', '3', '5'],
      dtype='<U1')
```

Creación de Matrices

Creando una matriz a partir de una lista de elementos:

```
>>> a = np.array([[2,3,5],[7,8,9]])
>>> a
array([[2, 3, 5],
       [7, 8, 9]])
>>> a.dtype
dtype('int32')
>>> b = np.array([[7,8,2.5],[3.2,8,9]])
>>> b
array([[ 7. ,  8. ,  2.5],
       [ 3.2,  8. ,  9. ]])
>>> b.dtype
dtype('float64')
```

Creación de Matrices

Cuando se crea una matriz se puede especificar el tipo de dato de los elementos que tendrá la matriz:

```
>>> a = np.array([[2,3,5],[7,8,9]], float)
>>> a
array([[ 2.,  3.,  5.],
       [ 7.,  8.,  9.]])
>>> b = np.array([[7,8,2.5],[3.2,8,9]], str)
>>> b
array([['7', '8', '2.5'],
       ['3.2', '8', '9']],
      dtype='<U3')
```

Propiedades de Arreglos N-dimensionales

Propiedad	Descripción
ndarray.ndim	Retorna el valor de la dimensión del arreglo N-dimensional (int) . • arreglo: 1 dimensión • matriz: 2 dimensiones
ndarray.shape	Retorna una tupla (*n,m*) que contiene el número de filas "*n*" y columnas "*m*" del arreglo n-dimensional.
ndarray.size	Retorna el número de elementos totales de un arreglo n-dimensional.
ndarray.dtype	Retorna el tipo de dato de los elementos del arreglo n-dimensional.

Propiedades de los arreglos

```
>>> a = np.array([2,3,4])
>>> print("Dimension de un arreglo: ",a.ndim)
Dimension de un arreglo:  1
>>> print("Filas y columnas del arreglo: ",a.shape)
Filas y columnas del arreglo:  (3,)
>>> print("# de elementos del arreglo: ",a.size)
# de elementos del arreglo:  3
>>> print("Tipo de dato de los elementos del arreglo: ",a.dtype)
Tipo de dato de los elementos del arreglo:  int32
>>>
```

Propiedades de las matrices

```
>>> b=np.array([[2,3,4],[3,5,7]])
>>> print("Dimension de una matriz: ",b.ndim)
Dimension de una matriz:  2
>>> print("Filas y columnas de la matriz: ",b.shape)
Filas y columnas de la matriz:  (2, 3)
>>> print("# de elementos de la matriz: ",b.size)
# de elementos de la matriz:  6
>>> print("Tipo de dato de los elementos de la matriz: ",b.dtype)
Tipo de dato de los elementos de la matriz:  int32
```

Inicialización de arreglos n-dimensionales

Propiedad	Descripción
np.zeros	Crea un arreglo de ceros.
np.ones	Crea un arreglo de unos.
np.empty	Crea un arreglo cuyos elementos son valores aleatorios.
np.full	Crea un arreglo en base a una constante.
np.eye	Crea una matriz identidad.
np.random.random	Crea un arreglo y lo llena con números aleatorios (0 - 1).

Inicialización de arreglos

```
>>> a = np.zeros(2,int)
>>> a
array([0, 0])
>>> b = np.ones((3),int)
>>> b
array([1, 1, 1])
>>> c = np.full((4),7,int)
>>> c
array([7, 7, 7, 7])
>>> d = np.empty((2),int)
>>> d
array([         -1, 2147483647])
>>> e = np.random.random(3)
>>> e
array([ 0.24326825,  0.1410721 ,  0.53921591])
```

Inicialización de matrices

```
>>> a = np.zeros((3,4),int)
>>> a
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
>>> b = np.ones((2,3))
>>> b
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> c= np.empty((3,4))
>>> c
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

Inicialización de matrices

```
>>> d = np.full((3,3),5,int)
>>> d
array([[5, 5, 5],
       [5, 5, 5],
       [5, 5, 5]])
>>> e = np.eye(2)
>>> e
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> f = np.random.random((2,3))
>>> f
array([[ 0.97772636,  0.70214669,  0.37990848],
       [ 0.45607978,  0.28254567,  0.27442132]])
```

Función arange: arreglos

```
>>> a = np.arange(6)
>>> a
array([0, 1, 2, 3, 4, 5])
>>>
>>> b = np.arange(2,10)
>>> b
array([2, 3, 4, 5, 6, 7, 8, 9])
```

Transformar un arreglo en matriz: reshape

La función reshape recibe la forma del nuevo arreglo n dimensional. La forma debe corresponder a la cantidad de elementos que tiene el arreglo.

```
>>> a = np.arange(10)
>>> b = a.reshape(5,2)
>>> b
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

Función reshape

```
>>> a = np.arange(12).reshape(4,3)
>>> a
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

Transformar una matriz en un arreglo: ravel

La función ravel transforma cualquier arreglo n-dimensional en un arreglo de una dimensión.

```
>>> a = np.array([[2,8,9],[6,9,10]])
>>> a
array([[ 2,  8,  9],
       [ 6,  9, 10]])
>>> b = a.ravel()
>>> b
array([ 2,  8,  9,  6,  9, 10])
```

Operaciones con escalares

Se puede realizar operaciones aritméticas entre los arreglos n-dimensionales y escalares. El resultado de esta operación afecta a los elementos del arreglo n-dimensional.

	Descripción
+	Suma
-	Resta.
*	Producto
/	División.
//	División entera.
**	Potencia.

Operaciones con escalares

```
>>> a = np.arange(10).reshape(2,5)
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a=a*10
>>> a
array([[ 0, 10, 20, 30, 40],
       [50, 60, 70, 80, 90]])
>>> a=a-5
>>> a
array([[-5,  5, 15, 25, 35],
       [45, 55, 65, 75, 85]])
>>> a=a//5
>>> a
array([[-1,  1,  3,  5,  7],
       [ 9, 11, 13, 15, 17]], dtype=int32)
>>> a=a/2
>>> a
array([[-0.5,  0.5,  1.5,  2.5,  3.5],
       [ 4.5,  5.5,  6.5,  7.5,  8.5]])
```

Operaciones en arreglos n-dimensionales

Propiedad	Descripción
a + b np.add(a,b)	Suma los elementos uno a uno de los arreglos.
a – b np.subtract(a,b)	Resta los elementos uno a uno de los arreglos.
a * b np.multiply(a,b)	Multiplica los elementos uno a uno de los arreglos.
a .dot(b)	Producto de la matriz.
a / b np.divide(a,b)	Divide los elementos uno a uno de los arreglos.
np.sqrt(x)	Raíz cuadrada de los elementos del arreglo.

Operaciones en arreglos

```
>>> a = np.array([1,5,3])
>>> b = np.array([2,6,4])
>>> a+b
array([ 3, 11,  7])
>>>
>>> a-b
array([-1, -1, -1])
>>>
>>> a*b
array([ 2, 30, 12])
```

Operaciones en arreglos

```
>>> a = np.array([1,5,3])
>>> b = np.array([2,6,4])
>>> a/b
array([ 0.5       ,  0.83333333,  0.75      ])
>>> np.sqrt(a)
array([ 1.        ,  2.23606798,  1.73205081])
```

Operaciones en matrices

```
>>> a = np.array([[1,3],[2,5]])
>>> b = np.array([[6,8],[4,2]])
>>> a+b
array([[ 7, 11],
       [ 6,  7]])
>>>
>>> a-b
array([[-5, -5],
       [-2,  3]])
```

Operaciones en matrices

```
>>> a = np.array([[1,3],[2,5]])
>>> b = np.array([[6,8],[4,2]])
>>>
>>> a*b
array([[ 6, 24],
       [ 8, 10]])
>>>
>>> a.dot(b)
array([[18, 14],
       [32, 26]])
```

Operaciones en matrices

```
>>> a = np.array([[1,3],[2,5]])
>>> b = np.array([[6,8],[4,2]])

>>> a/b
array([[ 0.16666667,  0.375     ],
       [ 0.5       ,  2.5       ]])
>>>
>>> np.sqrt(a)
array([[ 1.        ,  1.73205081],
       [ 1.41421356,  2.23606798]])
```

Funciones en arreglos n-dimensionales

Propiedad	Descripción
sum	Retorna la suma de los elementos del arreglo de n-dimensiones.
min	Retorna el menor elemento del arreglo de n-dimensiones.
max	Retorna el mayor elemento del arreglo de n-dimensiones.

Funciones en arreglos

```
>>> a = np.array([1,5,3,4,2])
>>> np.sum(a)
15
>>>
>>> np.min(a)
1
>>>
>>> np.max(a)
5
```

Funciones en matrices

```
>>> a = np.array([[2,3],[1,5]])
>>> np.sum(a)
11
>>>
>>> np.min(a)
1
>>>
>>> np.max(a)
5
```

Funciones en matrices (copy)

```
>>> a = np.array ([[1,3],[2,5],[6,4]])
>>> b = a.copy()
>>> b
array([[1, 3],
       [2, 5],
       [6, 4]])
```

Indexación numérica en arreglos

```
>>> a = np.array([1,5,3])
>>> a[0]=2
>>> a[-1]=4
>>> a
array([2, 5, 4])
```

Indexación numérica en matrices

```
>>> b = np.array([[1,5,3,7,2],[2,6,8,9,4]])
>>> b[0, 2] = 10
>>> b[1,-0] = 12
>>> b
array([[ 1,  5, 10,  7,  2],
       [12,  6,  8,  9,  4]])
```

Slicing en arreglos

```
>>> a = np.arange(6)**2
>>> a
array([ 0,  1,  4,  9, 16, 25])
>>> a[2]
4
>>> a[2:5]
array([ 4,  9, 16])
>>> a[::-1]
array([25, 16,  9,  4,  1,  0])
>>> a[:4:3]=50
>>> a
array([50,  1,  4, 50, 16, 25])
```

Slicing en matrices

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> b = a[:2, 1:3]
>>> b
array([[2, 3],
       [6, 7]])
>>> print (a[0, 1])
2
>>> b[0, 0] = 77
>>> print(a[0, 1])
77
```

Slicing en matrices

<array>[inicio:fin:step,inicio:fin:step]

<array>[filas,columnas]

Slicing en matrices

Utilizando un entero

```
>>> a=np.arange(12).reshape(3,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a[1,:2]
array([4, 5])
>>> a[1,:2].shape
(2,)
>>> a[1,:2].ndim
1
```

```
>>> a[1:2,:2]
array([[4, 5]])
>>> a[1:2,:2].shape
(1, 2)
>>> a[1:2,:2].ndim
2
```

Creación de Arreglos con Indexing

```
>>> b=np.array([[1,2],[3,4],[5,6]])
>>> b
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> c=np.array([b[0,0],b[1,1],b[2,0]])
>>> c
array([1, 4, 5])
```

```
>>> b=np.array([[1,2],[3,4],[5,6]])
>>> b
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> c=b[[0,1,2],[0,1,0]]
>>> c
array([1, 4, 5])
```

Utilizando los elementos de otro arreglo o matriz

Boolean Indexing

```
>>> a=np.array([[1,2],[3,4],[5,6]])
>>> a
array([[1, 2],
       [3, 4],
       [5, 6]])
>>>
>>>
>>> a[a>2]
array([3, 4, 5, 6])
>>> a[a>2].ndim
1
>>> a[a>2].shape
(4,)
```

Evaluando condiciones

Boolean Indexing

Crear un programa que sume los números pares de una matriz

```
>>> a=np.arange(10).reshape(2,5)
>>> suma=0
>>> r,c=a.shape
>>> for i in range(r):
...     for j in range(c):
...             if(a[i,j]%2==0):
...                     suma=suma+a[i,j]
...
>>> print(suma)
20
```

vs

```
>>> print(a[a%2==0].sum())
20
```

Where

np.where(condición)

```
>>> b=np.arange(10).reshape((2,5))
>>> b
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> np.where(b>5)
(array([1, 1, 1, 1]), array([1, 2, 3, 4]))
```

Retorna los índices de los elementos que cumplan con la condición

Any

np.any(condición)

```
>>> a
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 0]])
>>> np.any(a==1)
True
```

```
>>> a
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
>>> np.any(a==1)
False
```

Retorna True cuando al MENOS UNO de los elementos cumple con la condición

All

np.all(condición)

```
>>> a=np.zeros((3,3),dtype=int)
>>> a
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
>>> np.all(a==0)
True
```

```
>>> a[1,1]=1
>>> a
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 0]])
>>> np.all(a==0)
False
```

Retorna True si TODOS los elementos cumplen con la condición

Ejercicio