PYTHON
ARRAYS CON NUMPY

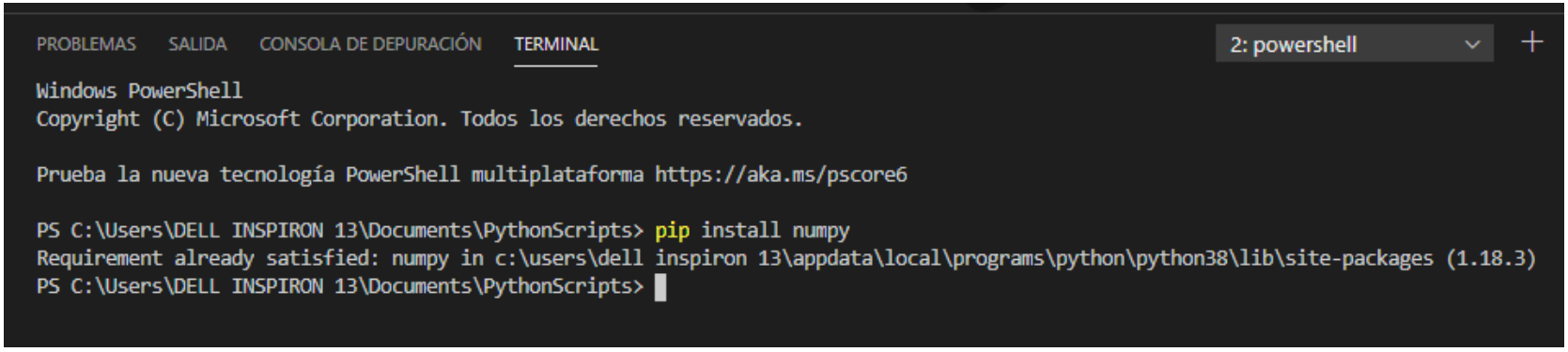
NumPy es el paquete fundamental para la computación científica en Python. Numpy Significa 'Python numérico'. Es una biblioteca que proporciona matrices de distintas dimensiones y una variedad de rutinas de operaciones.

Operaciones usando NumPy

•Operaciones matemáticas y lógicas en matrices.
•Transformadas de Fourier y rutinas para la manipulación de formas.
•Operaciones relacionadas con álgebra lineal. NumPy tiene funciones integradas para álgebra lineal y generación de números aleatorios.

La distribución estándar de Python no viene incluida con el módulo NumPy.

pip install numpy



```
# Ejemplo 1
import numpy as np

x = np.array([1,2,3,4,5], dtype = np.int8)
print (x.itemsize)

x = np.array([1,2,3,4,5], dtype = np.float32)
print (x.itemsize)

print (x.size)

x = np.array([1,2,3,4,5])
print (x.flags)
```

```
>>> x = np.array([1,2,3,4,5], dtype = np.int8)
>>> print (x.itemsize)
1
>>> x = np.array([1,2,3,4,5], dtype = np.float32)
>>> print (x.itemsize)
4
>>> print (x.size)
5
>>> x = np.array([1,2,3,4,5])
>>> print (x.flags)
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

itemsize devuelve en bytes la longitud de cada elemento de la matriz.
size: cantidad de elementos del array
flags: atributos del objeto ndarray. Sus valores actuales son devueltos por esta función.

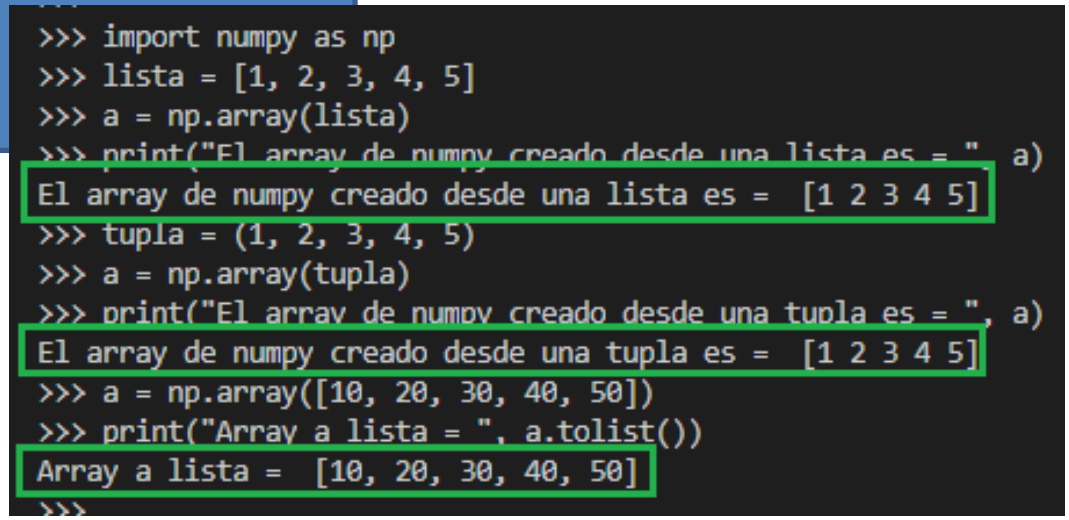
```
# Ejemplo 2
import numpy as np

m = np.linspace(10, 40, 4)
print (m)

m = np.linspace(10, 40, 4, dtype = "int")
print (m)

m = np.logspace(2, 3, 10)
print (m)

m = np.linspace(2, 3, 10)
print (m)

np.power(10, m)
```

```
>>>
>>> import numpy as np
>>> m = np.linspace(10, 40, 4)
>>> print (m)
[10. 20. 30. 40.]
>>> m = np.linspace(10, 40, 4, dtype = "int")
>>> print (m)
[10 20 30 40]
>>> m = np.logspace(2, 3, 10)
>>> print (m)
[ 100.          129.1549665   166.81005372  215.443469    278.25594022
  359.38136638  464.15888336  599.48425032  774.26368268 1000.        ]
>>> m = np.linspace(2, 3, 10)
>>> print (m)
[2.         2.11111111 2.22222222 2.33333333 2.44444444 2.55555556
 2.66666667 2.77777778 2.88888889 3.        ]
>>> np.power(10, m)
array([ 100.        ,  129.1549665 ,  166.81005372,  215.443469  ,
        278.25594022,  359.38136638,  464.15888336,  599.48425032,
        774.26368268, 1000.        ])
>>>
```

linspace genera un array formado por n números equiespaciados entre 2 números dados.
logspace genera un array formado también por n números entre 2 dados, pero en una escala logarít mica. La base a aplicar (por defecto 10) puede especificarse en el argumento base.

```
# Ejemplo 3
import numpy as np

lista = [1, 2, 3, 4, 5]

a = np.array(lista)

print("El array de numpy creado desde una lista es = ", a)

tupla = (1, 2, 3, 4, 5)

a = np.array(tupla)

print("El array de numpy creado desde una tupla es = ", a)

a = np.array([10, 20, 30, 40, 50])

print("Array a lista = ", a.tolist())
```

Se pueden crear arrays desde una lista o tupla.



```
# Ejemplo 4
import numpy as np

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

a.shape = (3,2)
print (a)
print(a.shape)

a = np.arange(24)
print (a)

a.ndim

b = a.reshape(2,4,3)
print (b)
```

```
>>> print (a)
[[1 2 3]
 [4 5 6]]
>>> print(a.shape)
(2, 3)
>>> a.shape = (3,2)
>>> print (a)
[[1 2]
 [3 4]
 [5 6]]
>>> print(a.shape)
(3, 2)
>>> a = np.arange(24)
>>> 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]
>>> a.ndim
1
>>> b = a.reshape(2,4,3)
>>> print (b)
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]

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

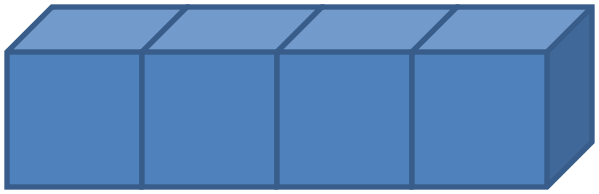
array: Es la forma más simple de crear un array a partir de un iterador como una simple lista (el iterador también puede ser otro array NumPy
shape: devuelve una tupla que consta de dimensiones de matriz. También se puede usar para cambiar el tamaño de la matriz.
ndim: devuelve el número de dimensiones de la matriz o los ejes.
reshape: función para cambiar el tamaño de una matriz.
arange genera un conjunto de números entre un valor de inicio y uno final, pudiendo especificar un incremento entre los valores

```
# Ejemplo 5
import numpy as np

a = np.full((2,3), fill_value = '-2')
print (a)

a = np.eye(5, dtype=int)
print (a)

a = np.identity(5)
print (a)
```

```
>>> import numpy as np
>>> a = np.full((2,3), fill_value = '-2')
>>> print (a)
[['-2' '-2' '-2']
 ['-2' '-2' '-2']]
>>> a = np.eye(5, dtype=int)
>>> print (a)
[[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]]
>>> a = np.identity(5)
>>> print (a)
[[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.]]
```

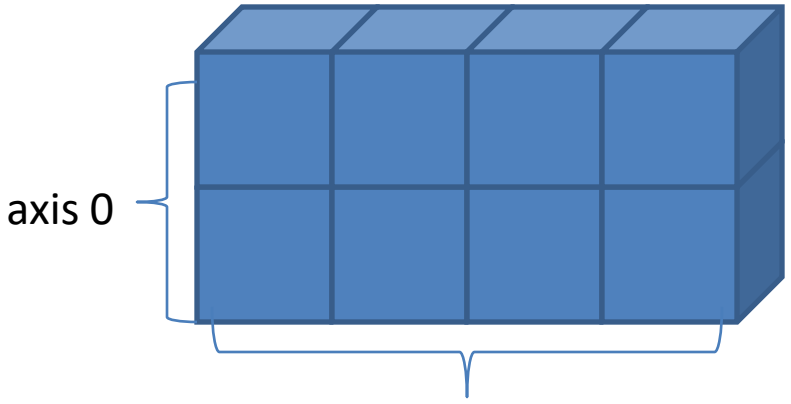
full() nos permite crear un array de cierto tamaño y tipo completarlo con un valor concreto:
eye() devuelve un array de dos dimensiones con unos en la diagonal principal y ceros en el resto del array.
identity() devuelve un array cuadrado con unos en la diagonal principal y ceros en el resto del array

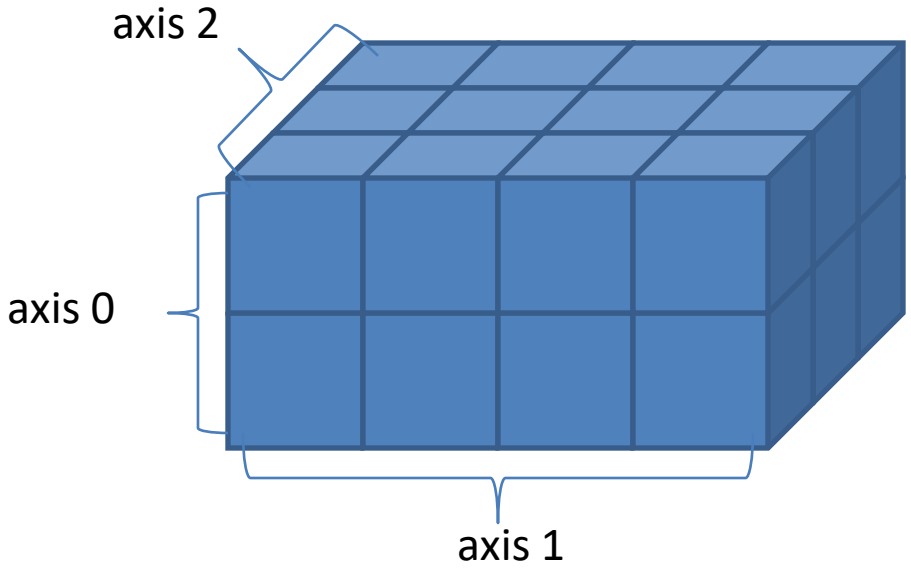
OPERACIONES BÁSICAS



1 Dimensión

2 Dimensiones





3 Dimensiones

ARRAYS UNIDIMENSIONALES

```
# Ejemplo 6

import numpy as np

lista=[25,12,15,66,12.5]
vector=np.array(lista)
print(vector)

vector = np.append (vector, [10, 11, 12])

print(vector)

vector = np.insert(vector, 1, 90)

print(vector)

print("Array ordenado = ", np.sort(vector))

print("- vector original")
print(vector)

vector = np.delete(vector, 2, axis = 0)

print(vector)
```

```
>>> import numpy as np
>>> lista=[25,12,15,66,12.5]
>>> vector=np.array(lista)
>>> print(vector)
[25.  12.  15.  66.  12.5]
>>> vector = np.append (vector, [10, 11, 12])
>>> print(vector)
[25.  12.  15.  66.  12.5 10.  11.  12. ]
>>> vector = np.insert(vector, 1, 90)
>>> print(vector)
[25.  90.  12.  15.  66.  12.5 10.  11.  12. ]
>>> print("Array ordenado = ", np.sort(vector))
Array ordenado =  [10.  11.  12.  12.  12.5 15.  25.  66.  90. ]
>>> print("- vector original")
- vector original
>>> print(vector)
[25.  90.  12.  15.  66.  12.5 10.  11.  12. ]
>>> vector = np.delete(vector, 2, axis = 0)
>>> print(vector)
[25.  90.  15.  66.  12.5 10.  11.  12. ]
>>>
```

```
# Ejemplo 7
import numpy as np
lista=[25,12,15,66,12.5,7]
vector=np.array(lista)
print(vector)
print("Nos quedamos con los >= 15", vector >= 15)
print( vector % 2 == 0)
print( vector ** 2)
print("- sumarle 1 a cada elemento del vector:", (vector + 1))
print("- multiplicar por 5 cada elemento del vector:", (vector * 5))
print("- suma de los elementos:", np.sum(vector))
print("- promedio (media) de los elementos:", np.mean(vector))
print("- El máximo de los elementos:", np.max(vector))
print("- El mínimo de los elementos:", np.min(vector))
```

```
>>> print(vector)
[25.  12.  15.  66.  12.5  7. ]
>>> print("Nos quedamos con los >= 15", vector >= 15)
Nos quedamos con los >= 15 [ True False  True  True False False]
>>> print( vector % 2 == 0)
[False  True False  True False False]
>>> print( vector ** 2)
[ 625.    144.    225.   4356.    156.25   49.  ]
>>> print("- sumarle 1 a cada elemento del vector:", (vector + 1))
- sumarle 1 a cada elemento del vector: [26.  13.  16.  67.  13.5  8. ]
>>> print("- multiplicar por 5 cada elemento del vector:", (vector * 5))
- multiplicar por 5 cada elemento del vector: [125.   60.   75.  330.   62.5  35. ]
>>> print("- suma de los elementos:", np.sum(vector))
- suma de los elementos: 137.5
>>> print("- promedio (media) de los elementos:", np.mean(vector))
- promedio (media) de los elementos: 22.916666666666668
>>> print("- El máximo de los elementos:", np.max(vector))
- El máximo de los elementos: 66.0
>>> print("- El mínimo de los elementos:", np.min(vector))
- El mínimo de los elementos: 7.0
```

```
# Ejemplo 8
import numpy as np

lista=[25,12,15,66,12.5,7]
vector=np.array(lista)
print(vector)

print("-
 el vector sumado a si mismo:", (vector + vector))
print("- suma de vectores vector1 y vector2 :")
vector2=np.array([11, 55, 1.2, 7.4, -8, 32])
print(vector + vector2)
print(vector - vector2)
sumo_dos = lambda x: x + 2

print("Array después de la función sumo_dos: ", sum
o_dos(vector))
print(vector)
```

```
>>> print("- el vector sumado a si mismo:", (vector + vector))
- el vector sumado a si mismo: [ 50.  24.  30. 132.  25.  14.]
>>> print("- suma de vectores vector1 y vector2 (mismo tamaño):")
- suma de vectores vector1 y vector2 (mismo tamaño):
>>> vector2=np.array([11, 55, 1.2, 7.4, -8, 32])
>>> print(vector + vector2)
[36.  67.  16.2 73.4  4.5 39. ]
>>> print(vector - vector2)
[ 14.  -43.   13.8  58.6  20.5 -25. ]
>>> sumo_dos = lambda x: x + 2
>>> print("Array después de la función sumo_dos: ", sumo_dos(vector))
Array después de la función sumo_dos:  [27.  14.  17.  68.  14.5  9. ]
```

Qué pasa cuando aplicamos una operación entre 2 arrays de una dimensión (1-D) con diferentes tamaños?

- Tendremos un mensaje de error, específicamente un ValueError
- la ejecución trata de emparejar desde el primero hasta el último elemento
- cuando un elemento no encuentra pareja, tendremos una error de emparejamiento .

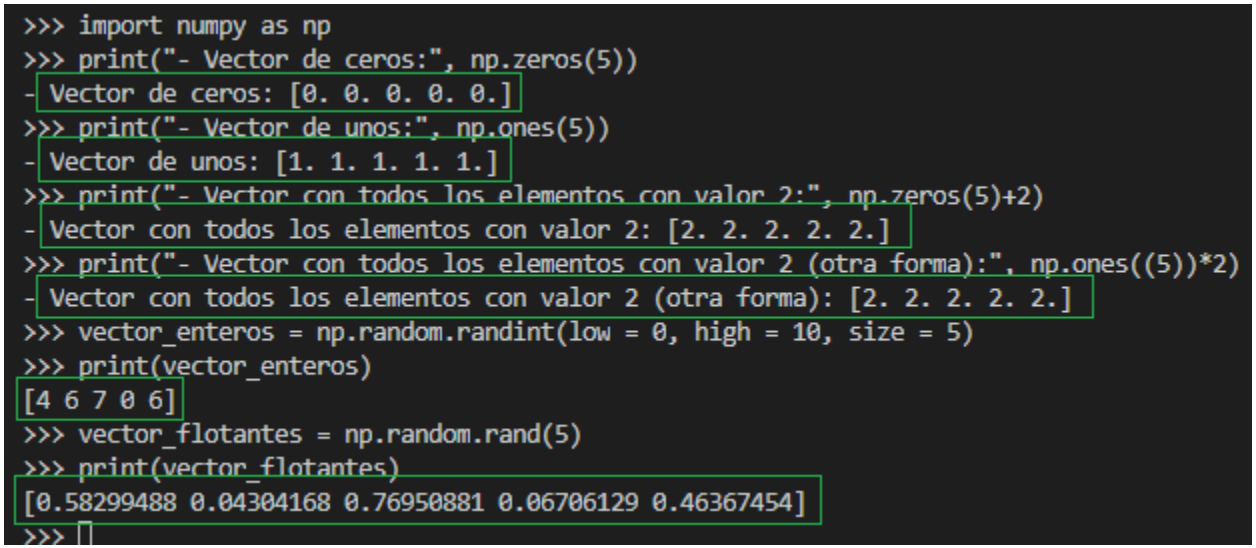
```
# Ejemplo 9
import numpy as np

vector=np.random.randint(low = 10, high = 100, siz
e = 5)
print(vector)

for elemento in vector:
   print(elemento)
```

```
>>> import numpy as np
>>> vector=np.random.randint(low = 10, high = 100, size = 5)
>>> print(vector)
[76 99 65 51 62]
>>>
>>> for elemento in vector:
...     print(elemento)
...
76
99
65
51
62
```

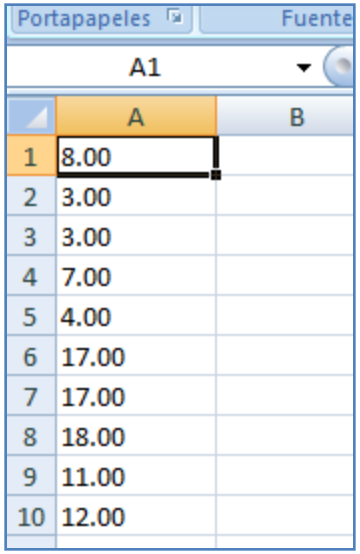
```python
# Ejemplo 10
import numpy as np

lista=[25, 12, 15, 66, 12.5, 7]
vector=np.array(lista)
print(vector)

print("Un subset del vector = ", vector[-3:])
index = np.where(vector == 15)
print("Dónde está el número 15?: ", index)
print("Otro subset del vector = ", vector[2:5])

print(vector[3])
print(vector[1:4])
print(vector[1:])
print(vector[:4])
print(vector[:])
```

```
>>> import numpy as np
>>> lista=[25, 12, 15, 66, 12.5, 7]
>>> vector=np.array(lista)
>>> print(vector)
[25.  12.  15.  66.  12.5  7. ]
>>> print("Un subset del vector = ", vector[-3:])
Un subset del vector =  [66.  12.5  7. ]
>>> index = np.where(vector == 15)
>>> print("Dónde está el número 15?: ", index)
Dónde está el número 15?:  (array([2], dtype=int64),)
>>> print("Otro subset del vector = ", vector[2:5])
Otro subset del vector =  [15.  66.  12.5]
>>> print(vector[3])
66.0
>>> print(vector[1:4])
[12. 15. 66.]
>>> print(vector[1:])
[12.  15.  66.  12.5  7. ]
>>> print(vector[:4])
[25. 12. 15. 66.]
>>> print(vector[:])
[25.  12.  15.  66.  12.5  7. ]
```

```
# Ejemplo 11
import numpy as np

print("- Vector de ceros:", np.zeros(5))
print("- Vector de unos:", np.ones(5))
print("- Vector con todos los elementos con valor 2:", np.zeros(5)+2)
print("- Vector con todos los elementos con valor 2 (otra forma):", np.ones((5))*2)

vector_enteros = np.random.randint(low = 0, high = 10, size = 5)
print(vector_enteros)

vector_flotantes = np.random.rand(5)
print(vector_flotantes)
```



```
# Ejemplo 12
import numpy as np

vector1=np.random.randint(low = 0, high = 10, size = 5)
print(vector1)

vector2=np.random.randint(low = 11, high = 20, size = 5)
print(vector2)

juntos = np.stack ((vector1, vector2), axis=0)
print(juntos)

separados = np.split(juntos, 2)
print(separados)

nuevo_Array = np.append(vector1, vector2)
print(nuevo_Array)
np.savetxt("myArray.csv", nuevo_Array)

np.savetxt("myArray.csv", nuevo_Array, fmt='%.2f')
```




```
>>> import numpy as np
>>> vector1=np.random.randint(low = 0, high = 10, size = 5)
>>> print(vector1)
[8 3 3 7 4]
>>> vector2=np.random.randint(low = 11, high = 20, size = 5)
>>> print(vector2)
[17 17 18 11 12]
>>> juntos = np.stack ((vector1, vector2), axis=0)
>>> print(juntos)
[[ 8  3  3  7  4]
 [17 17 18 11 12]]
>>> separados = np.split(juntos, 2)
>>> print(separados)
[array([[8, 3, 3, 7, 4]]), array([[17, 17, 18, 11, 12]])]
>>> nuevo_Array = np.append(vector1, vector2)
>>> print(nuevo_Array)
[ 8  3  3  7  4 17 17 18 11 12]
>>> np.savetxt("ejemplos/numpy/myArray.csv", nuevo_Array)
>>> np.savetxt("ejemplos/numpy/myArray.csv", nuevo_Array, fmt='%.2f')
>>>
```

stack une una secuencia de matrices a lo largo de un nuevo eje.
split divide el array

ARRAYS BIDIMENSIONALES

```
# Ejemplo 13
import numpy as np
lista_de_listas=[ [1  ,-4],
                  [12 , 3],
                  [7.2, 5]]

matriz = np.array(lista_de_listas)
print(matriz)

col = np.array([[400], [800], [260]])
print(col)

matriz = np.append(matriz, col, axis = 1)

print(matriz)

matriz = np.append(matriz, [[50, 60, 70]], axis = 0)

print(matriz)

matriz = np.delete(matriz, 0, axis = 0)

print(matriz)

matriz = np.delete(matriz, 1, axis = 1)

print(matriz)
```

```
>>> matriz = np.array(lista_de_listas)
>>> print(matriz)
[[ 1.  -4. ]
 [12.   3. ]
 [ 7.2  5. ]]
>>> col = np.array([[400], [800], [260]])
>>> print(col)
[[400]
 [800]
 [260]]
>>> matriz = np.append(matriz, col, axis = 1)
>>> print(matriz)
[[  1.   -4.  400. ]
 [ 12.    3.  800. ]
 [  7.2   5.  260. ]]
>>> matriz = np.append(matriz, [[50, 60, 70]], axis = 0)
>>> print(matriz)
[[  1.   -4.  400. ]
 [ 12.    3.  800. ]
 [  7.2   5.  260. ]
 [ 50.   60.   70. ]]
>>> matriz = np.delete(matriz, 0, axis = 0)
>>> print(matriz)
[[ 12.    3.  800. ]
 [  7.2   5.  260. ]
 [ 50.   60.   70. ]]
>>> matriz = np.delete(matriz, 1, axis = 1)
>>> print(matriz)
[[ 12.  800. ]
 [  7.2 260. ]
 [ 50.   70. ]]
>>>
```

```
# Ejemplo 14
import numpy as np
lista_de_listas=[ [1  ,-4],
                  [12 , 3],
                  [7.2, 5]]

matriz = np.array(lista_de_listas)
print(matriz)

print("Matriz ordenada = ", np.sort(matriz))

print("Suma por columna = ", matriz.sum(axis=0))
print("Suma por fila = ", matriz.sum(axis=1))
print("Máximo = ", matriz.max())
print("Mínimo = ", matriz.min())
```

```
>>> print(matriz)
[[ 1.  -4. ]
 [12.   3. ]
 [ 7.2  5. ]]
>>> print("Matriz ordenada = ", np.sort(matriz))
Matriz ordenada =  [[-4.   1. ]
 [ 3.  12. ]
 [ 5.   7.2]]
>>> print("Suma por columna = ", matriz.sum(axis=0))
Suma por columna =  [20.2  4. ]
>>> print("Suma por fila = ", matriz.sum(axis=1))
Suma por fila =  [-3.  15.  12.2]
>>> print("Máximo = ", matriz.max())
Máximo =  12.0
>>> print("Mínimo = ", matriz.min())
Mínimo =  -4.0
```

```
# Ejemplo 15
import numpy as np

matriz = np.random.randint(100, size=(2, 4))
print(matriz)

for elemento in matriz:
   print(elemento)
```

```
>>> import numpy as np
>>> matriz = np.random.randint(100, size=(2, 4))
>>> print(matriz)
[[49 13 59 89]
 [46 83 30 69]]
>>> for elemento in matriz:
...     print(elemento)
...
[49 13 59 89]
[46 83 30 69]
```

```python
# Ejemplo 16
import numpy as np
lista_de_listas=[ [1  ,-4],
                  [12 , 3],
                  [7.2, 5]]

matriz = np.array(lista_de_listas)
print(matriz)

print("Elemento en los índices [1][1] = ", matriz[1][1])
print("Elementos individuales")
print(matriz[0,1])
print(matriz[2,1])

print("Vector de elementos de la fila 1")
print(matriz[1,:])

print("Vector de elementos de la columna 0")
print(matriz[:,0])

print("Submatriz de 2x2 con las primeras dos filas")
print(matriz[0:2,:])

print("Submatriz de 2x2 con las ultimas dos filas")
print(matriz[1:3,:])
```

```
>>> print("Elemento en los índices [1][1] = ", matriz[1][1])
Elemento en los índices [1][1] =  3.0
>>> print("Elementos individuales")
Elementos individuales
>>> print(matriz[0,1])
-4.0
>>> print(matriz[2,1])
5.0
>>> print("Vector de elementos de la fila 1")
Vector de elementos de la fila 1
>>> print(matriz[1,:])
[12.  3.]
>>> print("Vector de elementos de la columna 0")
Vector de elementos de la columna 0
>>> print(matriz[:,0])
[ 1.  12.   7.2]
>>> print("Submatriz de 2x2 con las primeras dos filas")
Submatriz de 2x2 con las primeras dos filas
>>> print(matriz[0:2,:])
[[ 1. -4.]
 [12.  3.]]
>>> print("Submatriz de 2x2 con las ultimas dos filas")
Submatriz de 2x2 con las ultimas dos filas
>>> print(matriz[1:3,:])
[[12.   3. ]
 [ 7.2  5. ]]
```

```
# Ejemplo 17
import numpy as np
lista_de_listas=[ [1  ,-4],
                  [12 , 3],
                  [7.2, 5]]

matriz = np.array(lista_de_listas)
print(matriz)
print("- Le asignamos el valor 4
 a los elementos de la columna 0:")
matriz[:,0] = 4
print(matriz)
print("- Dividimos por 3 la columna 1:")
matriz[:,1]=matriz[:,1]/3.0
print(matriz)
print("- Multiplicamos por 5 la fila 1:")
matriz[1,:]=matriz[1,:]*5
print(matriz)
print("- Le sumamos 1 a toda la matriz:")
matriz= matriz + 1
print(matriz)
print(matriz.transpose())
print(matriz.ravel())
```

transpose invierte o permuta los ejes de una matriz; devuelve la matriz modificada
ravel devuelve la matriz aplanada.

```
- Le asignamos el valor 4 a los elementos de la columna 0:
>>> matriz[:,0] = 4
>>> print(matriz)
[[ 4. -4.]
 [ 4.  3.]
 [ 4.  5.]]
>>> print("- Dividimos por 3 la columna 1:")
- Dividimos por 3 la columna 1:
>>> matriz[:,1]=matriz[:,1]/3.0
>>> print(matriz)
[[ 4.         -1.33333333]
 [ 4.          1.        ]
 [ 4.          1.66666667]]
>>> print("- Multiplicamos por 5 la fila 1:")
- Multiplicamos por 5 la fila 1:
>>> matriz[1,:]=matriz[1,:]*5
>>> print(matriz)
[[ 4.         -1.33333333]
 [20.          5.        ]
 [ 4.          1.66666667]]
>>> print("- Le sumamos 1 a toda la matriz:")
- Le sumamos 1 a toda la matriz:
>>> matriz= matriz + 1
>>> print(matriz)
[[ 5.         -0.33333333]
 [21.          6.        ]
 [ 5.          2.66666667]]
>>> print(matriz.transpose())
[[ 5.         21.         5.        ]
 [-0.33333333  6.         2.66666667]]
>>> print(matriz.ravel())
[ 5.         -0.33333333 21.         6.         5.         2.66666667]
```

```python
# Ejemplo 18
import numpy as np

print("- Matriz creada con np.zeros:")
matriz_ceros = np.zeros((2,3))
print(matriz_ceros)

print("- Matriz creada con np.ones:")
matriz_unos = np.ones((3,2))
print(matriz_unos)

print("- Copia de la matriz creada con np.ones:")
matriz_unos_copia=np.copy(matriz_unos)
print(matriz_unos_copia)

matriz_enteros = np.random.randint(100, size=(2, 4))
print(matriz_enteros)

matriz_flotantes = np.random.rand(2,3)
print(matriz_flotantes)
```

```
>>> matriz_ceros = np.zeros((2,3))
>>> print(matriz_ceros)
[[0. 0. 0.]
 [0. 0. 0.]]
>>> print("- Matriz creada con np.ones:")
- Matriz creada con np.ones:
>>> matriz_unos = np.ones((3,2))
>>> print(matriz_unos)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
>>> print("- Copia de la matriz creada con np.ones:")
- Copia de la matriz creada con np.ones:
>>> matriz_unos_copia=np.copy(matriz_unos)
>>> print(matriz_unos_copia)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
>>> matriz_enteros = np.random.randint(100, size=(2, 4))
>>> print(matriz_enteros)
[[ 7 19 57 69]
 [53 54 97 67]]
>>> matriz_flotantes = np.random.rand(2,3)
>>> print(matriz_flotantes)
[[0.47603169 0.91825951 0.53427126]
 [0.73624122 0.18526288 0.03372684]]
>>>
```

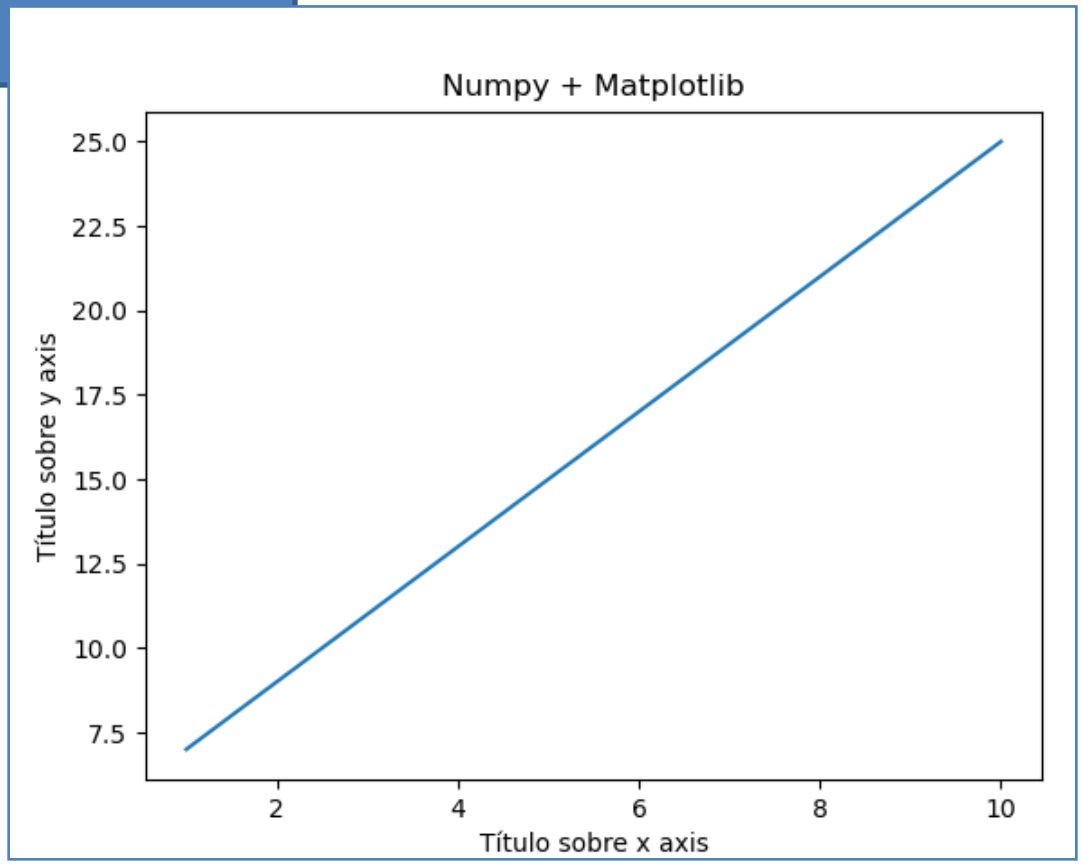
```python
# Ejemplo 19
import numpy as np

m1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
m2 = np.array([[9, 10, 11, 12], [13, 14, 15, 16]])

print("Matriz 1:\n", m1)
print("Matriz 2:\n", m2)
print("Mezcla Vertical:")
print(np.vstack((m1, m2)))
print("Mezcla Horizontal:")
print(np.hstack((m1, m2)))
print("División vertical m1 en 2 piezas:")
r = np.vsplit(m1, 2)
print(r[0])
print(r[1])
print(r)

print("División horizontal m1 en 2 piezas:")
r = np.hsplit(m1, 2)
print(r[0])
print(r[1])

print(m2)
np.savetxt("myMatrix.csv", m2, fmt='%.2f')
```

```
Mezcla Vertical:
>>> print(np.vstack((m1, m2)))
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
>>> print("Mezcla Horizontal:")
Mezcla Horizontal:
>>> print(np.hstack((m1, m2)))
[[ 1  2  3  4  9 10 11 12]
 [ 5  6  7  8 13 14 15 16]]
>>> print("División vertical m1 en 2 piezas:")
División vertical m1 en 2 piezas:
>>> r = np.vsplit(m1, 2)
>>> print(r[0])
[[1 2 3 4]]
>>> print(r[1])
[[5 6 7 8]]
>>> print(r)
[array([[1, 2, 3, 4]]), array([[5, 6, 7, 8]])]
>>> print("División horizontal m1 en 2 piezas:")
División horizontal m1 en 2 piezas:
>>> r = np.hsplit(m1, 2)
>>> print(r[0])
[[1 2]
 [5 6]]
>>> print(r[1])
[[3 4]
 [7 8]]
```

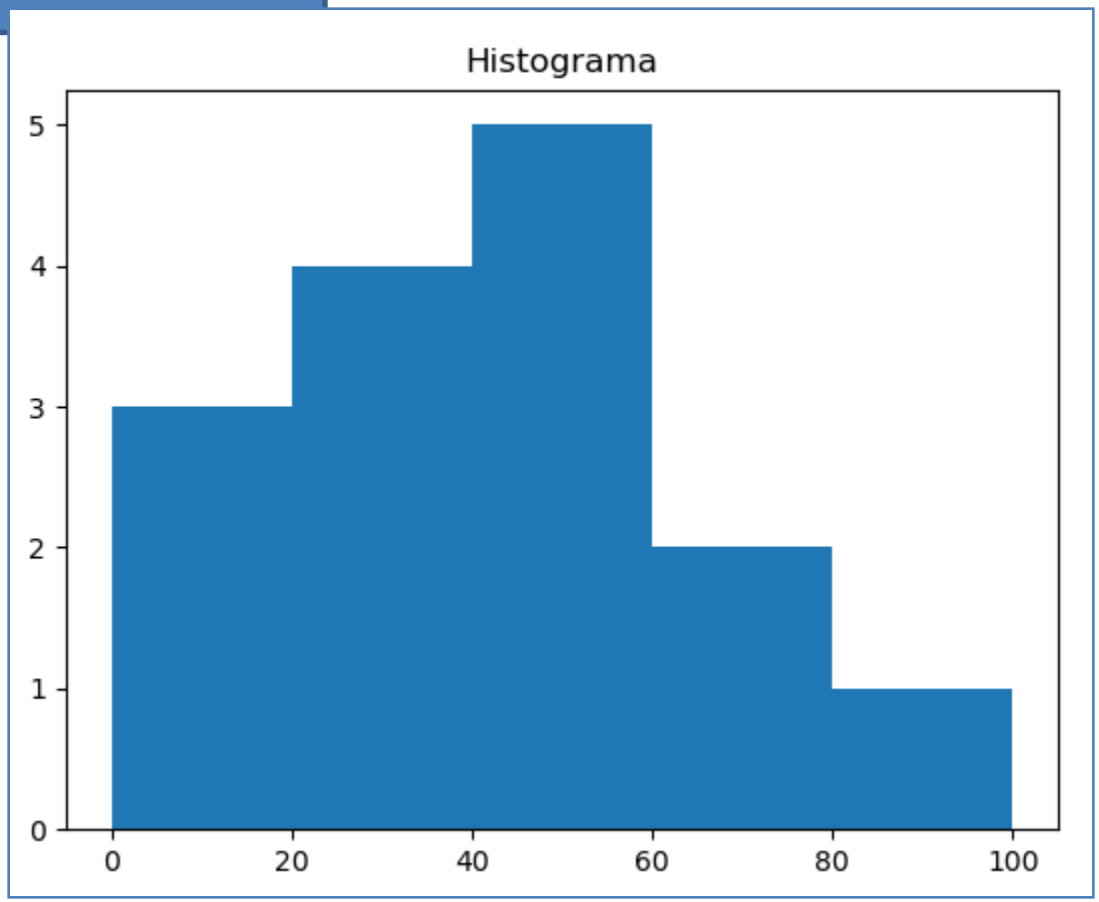
vstack apila matrices verticalmente en cuanto a filas
hstack apila matrices horizontalmente, en cuanto a columnas.
vsplit y **hsplit** dividen la matriz, en cuanto a filas y columnas, respectivamente.

GRÁFICOS
NUMPY + MATPLOTLIB

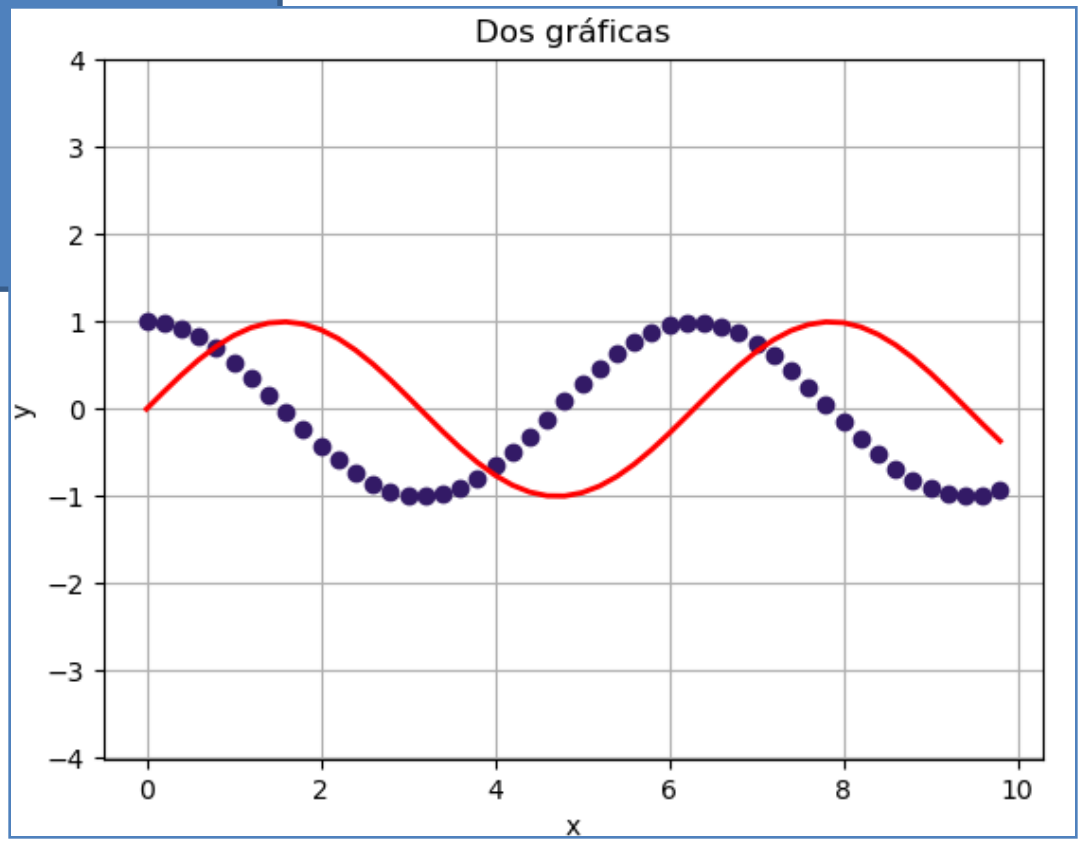
```
# Ejemplo 20
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(1,11)
y = 2 * x + 5
plt.title("Numpy + Matplotlib")
plt.xlabel("Título sobre x axis")
plt.ylabel("Título sobre y axis")
plt.plot(x,y)
plt.show()
```



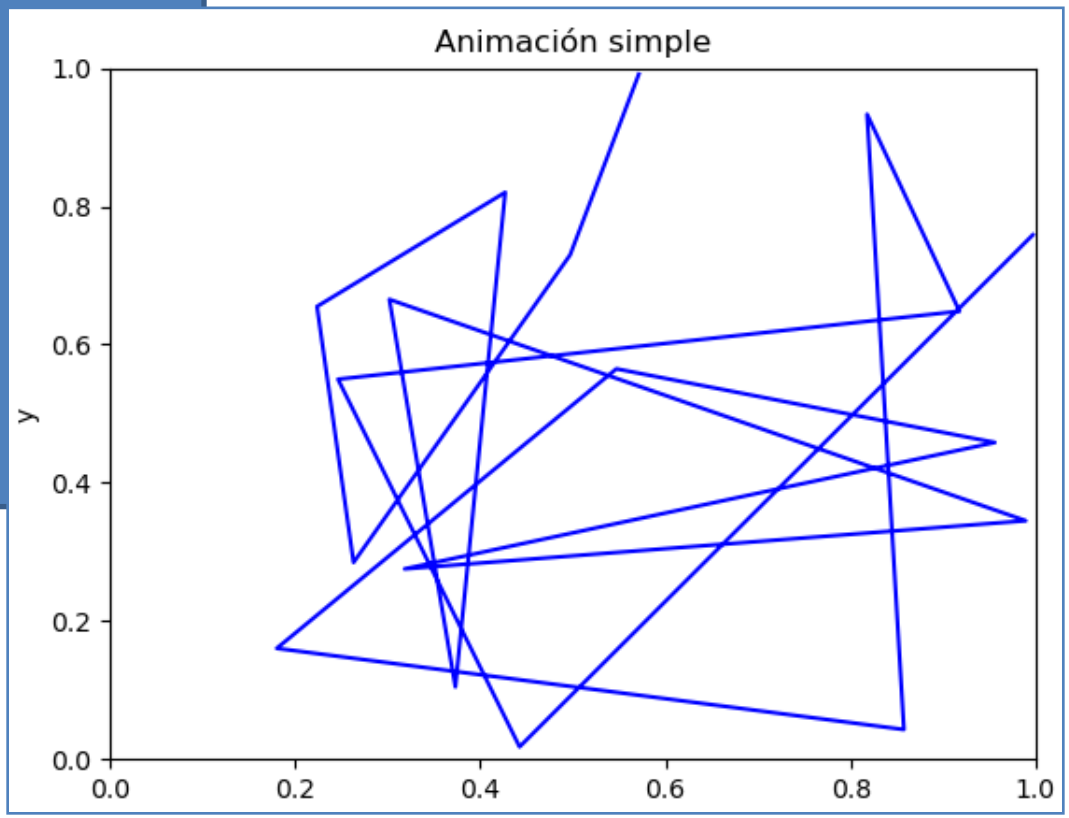
```
# Ejemplo 21
Import matplotlib.pyplot as plt
import numpy as np

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("Histograma")

plt.show()
```



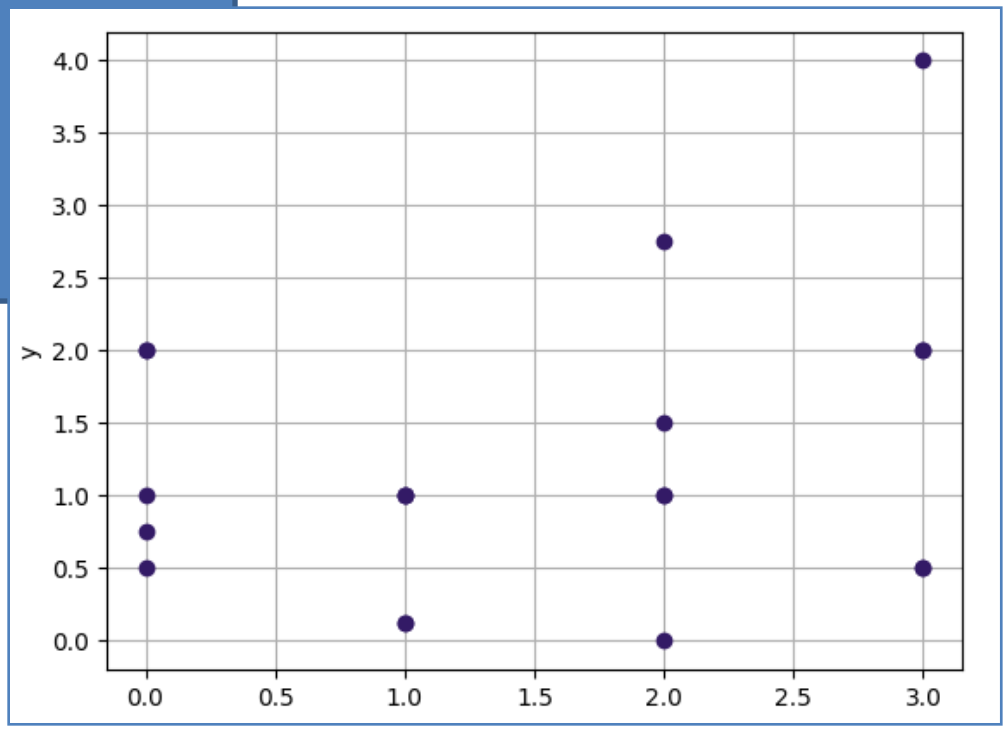
```
# Ejemplo 22
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0,10,0.2)
y1 = np.cos(x)
y2 = np.sin(x)

plt.plot(x,y1,'o',linewidth=3,color=(0.2,0.1,0.4))
plt.plot(x,y2,'-',linewidth=2,color='r')
plt.grid()
plt.axis('equal')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Dos gráficas')
plt.show()
```



```
# Ejemplo 23
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np

def update_line(num, data, line):
    line.set_data(data[...,:num])
    return line,

fig1 = plt.figure()

data = np.random.rand(2, 25)
l, = plt.plot([], [], 'b-')
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Animación simple')
line_ani = animation.FuncAnimation(fig1,
update_line, 25, fargs=(data, l), interval=50,
blit=True)
plt.show()
```



```
# Ejemplo 24
# Creamos un archivo .csv con los siguientes datos:
# 2,0.75,2,1,0.5
# 1,0.125,1,1,0.125
# 2.75,1.5,1,0,1
# 4,0.5,2,2,0.5

import numpy as np

csv_array = np.genfromtxt('archivo.csv', delimiter = ',')
print(csv_array)


import matplotlib.pyplot as plt

plt.plot(csv_array,'o',linewidth=3,color=(0.2,0.1,0.4))
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



https://numpy.org/

https://numpy.org/community/

https://numpy.org/doc/stable/user/index.html

numpy · PyPI

FIN