NUMPY

Lista de métodos:

- **numpy.max()**: obtener el valor máximo de un array. https://www.educba.com/numpy-max/ (https://www.educba.com/numpy-max/)
- **numpy.mean()**: sirve para calcular la MEDIA de un array https://numpy.org/doc/stable/reference/generated/numpy.mean.html (https://numpy.org/doc/stable/reference/generated/numpy.mean.html)
- **numpy.median()**: Sirve para calcular la ME-DIA-NA de un array. https://numpy.org/doc/stable/reference/generated/numpy.median.html (https://numpy.org/doc/stable/reference/generated/numpy.median.html)
- **np.std()**: Desviación estándar.
- **np.var()**: Calcula la varianza.
- **numpy.arange(n)**: crea un array con la extensión que indiques en n.
- **np.sum(x)**: suma todos los valores del array x.

Numpy es una librería de python que sirve para trabajar con **ARRAYS**

In [1]:
```
import numpy as np #importar numpy

arr = np.array([1, 2, 3, 4, 5]) #funcion para crear un array

print(arr)
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

- Un array se crea con la función *array()* y se le pasa una lista, tupla, etc.
- El objeto array se llama *ndarray*

DIMENSIONES

In [25]:
```
import numpy as np

#array 1-D unidimensional, formado por elementos 0-D
arr1 = np.array([1, 2, 3, 4, 5])

#array 2-D bidimensional, formado por elementos 1-D: REPRESENTA UNA MATRI
a = [1, 2]
b = [2, 3]
arr2 = np.array([a, b])

#array 3-D tridimensional, formado por elementos 2-D
```

```
c = [1, 2]
```

IMPORTANTE: LOS ARRAYS QUE CONFORMAN LOS ARRAYS BIDIMENSIONALES Y TRIDIMENSIONALES DEBEN SER DE LA MISMA LONGITUD

In [26]:
```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
```

```
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

INDEXING

ACCEDIENDO ELEMENTOS ARRAY 2-D / 3-D

Para acceder a un elemento de un array bidimensional utilizamos por ejemplo [0, 1], en el que 0 hace referencia a la dimensión y 1 hace referencia al elemento de dicha dimensión al cual queremos acceder.

Para un array tridimensional es el mismo proceso. Básicamente vamos atravesando dimensiones: [0, 1, 0].

In [33]:
```
print('Array bidimensional: ', arr2[1, 0])

print('Primera dimensión del array tridimensional: ', arr3[0])

print('Segunda dimensión del array tridimensional: ', arr3[0, 1])
```

```
Array bidimensional:  2
Primera dimensión del array tridimensional:  [[1 2]
 [2 3]]
Segunda dimensión del array tridimensional:  [2 3]
Tercera dimensión del array tridimensional:  2
```

SLICING

In [44]:
```
x = [1, 5, 4, 3, 2, 3]
z = [2, 3, 4, 1, 6, 8]

arra = np.array([x, z])
```

```
print(arra[1, 1:4])
print(arra[0, -5:-1])
print(arra[1, 2::2])
print(arra[0:2, ::2]) #también se puede slicear la primera dimensión del
                        #y a la z (índice 1)
```

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

DATA TYPES EN NUMPY

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)

In [1]:
```
#obtener el tipo de data que hay dentro de un array
import numpy as np

ints = np.array([1, 2, 3, 4])
strs = np.array(['apple', 'banana', 'cherry'])


print(ints.dtype)
```

```
int32
<U6
```

Definir el tipo de data esperado en un array

In [47]:
```
arr = np.array([1, 2, 3, 4], dtype='S') #este array no son integers sino

print(arr)
print(arr.dtype)
```

```
[b'1' b'2' b'3' b'4']
```

Si los valores que hay dentro del array no pueden convertirse al tipo de dato que nosotros hemos definido se producirá un

ValueError

Cambiar el tipo de data de un array ya existente con el método *astype()*

- *astype()* crea una copia del array
- Se le pasa como parámetro el tipo de data que quieres

In [62]:
```
a = np.array([1.2, 2.3, 4.5, 4.9])
print(a)

#float > integer
otroA = a.astype('i')
print(otroA)

#int > boolean
b = np.array([1, 0, 3])
unBoolean = b.astype(bool)
```

```
[1.2 2.3 4.5 4.9]
[1 2 4 4]
[ True False  True]
```

COPIA VS VISTA

- *.copy()*: Una copia de un array es un array nuevo independiente del original
- *.view()*: Una vista del array depende del original y cualquier modificación hecha al array original afectará a la vista

.base devuelve None si el array doesn't owns the data, de lo contrario se referirá al objeto original

In [63]:
```
arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()
y = arr.view()

print(x.base)
```

```
None
[1 2 3 4 5]
```

OBTENER EL SHAPE DE UN ARRAY

The shape of an array is the number of elements in each dimension.

In [64]:
```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

```
(2, 4)
```

The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

RESHAPING: *reshape()*

By reshaping we can add or remove dimensions or change number of elements in each dimension.

In [65]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

In [66]:
```
#1 dimensión > 3 dimensiones

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

newarr = arr.reshape(2, 3, 2)
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

IMPORTANTE: EL RESHAPE ES UNA **VISTA** DEL ARRAY

UNKNOWN DIMENSION

You are allowed to have one "unknown" dimension.

Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method.

Pass -1 as the value, and NumPy will calculate this number for you.

In [67]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)
```

```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

FLATTENING ARRAYS: convertir un array multidimensional a una sola dimensión

reshape(-1)

In [68]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)

print(newarr)
```

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

ITERAR

- Para iterar a través de los arrays se puede usar un for loop; para iterar a través de todos los elementos de cada dimensión se añade un for loop por cada dimensión

In [4]:
```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print("Iteración simple del array multidimensional: ")
for x in arr:
  print(x)

print("Iteración a través de las dimensiones para recorrer todos los elem
for x in arr:
  for y in x:
    for z in y:
      print(z)
```

```
Iteración simple del array multidimensional:
[[1 2 3]
 [4 5 6]]
[[ 7  8  9]
 [10 11 12]]
Iteración a través de las dimensiones para recorrer todos los elemento
s:
1
```

nditer()

In [9]:
```
print("Primera iteracion: ")
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for x in np.nditer(arr):
  print(x)
print("Segunda iteracion: ")
for x in np.nditer(arr[:, ::2]):
```

```
Primera iteracion:
1
2
3
4
5
6
7
8
Segunda iteracion:
1
3
5
7
```

We can use **op_dtypes** argument and pass it the expected datatype to change the datatype of elements while iterating.

NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in nditer() we pass **flags=['buffered'].**

In [6]:
```
for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
```

```
b'1'
b'2'
b'3'
b'4'
b'5'
b'6'
b'7'
b'8'
b'9'
b'10'
b'11'
b'12'
```

ndenumerate()

Sirve para enumerar las iteraciones del bucle, es decir, para obtener el índice de cada elemento.

In [16]:
```
#Una dimensión
arr = np.array([1, 2, 3])

for idx, x in np.ndenumerate(arr):
    print(idx, x)

#2 dimensiones
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for idx, x in np.ndenumerate(arr):
```

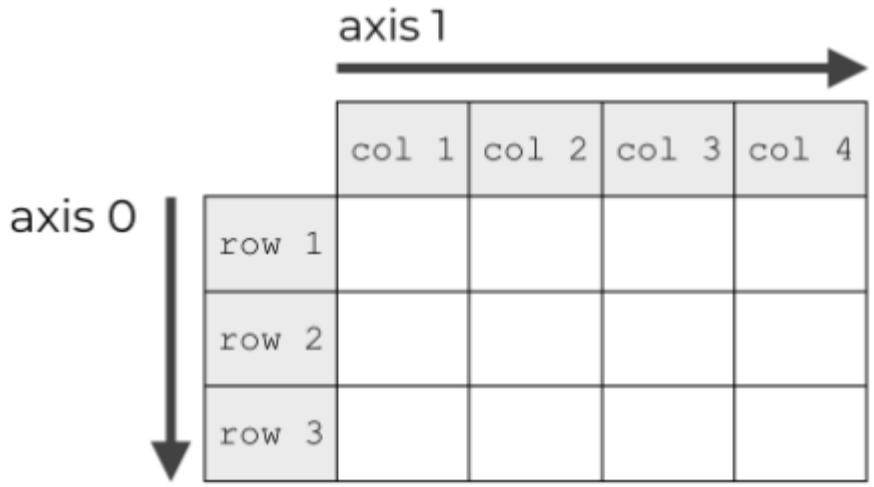
```
(0,) 1
(1,) 2
(2,) 3
(0, 0) 1
(0, 1) 2
(0, 2) 3
(0, 3) 4
(1, 0) 5
(1, 1) 6
(1, 2) 7
(1, 3) 8
```

ARRAY JOINING

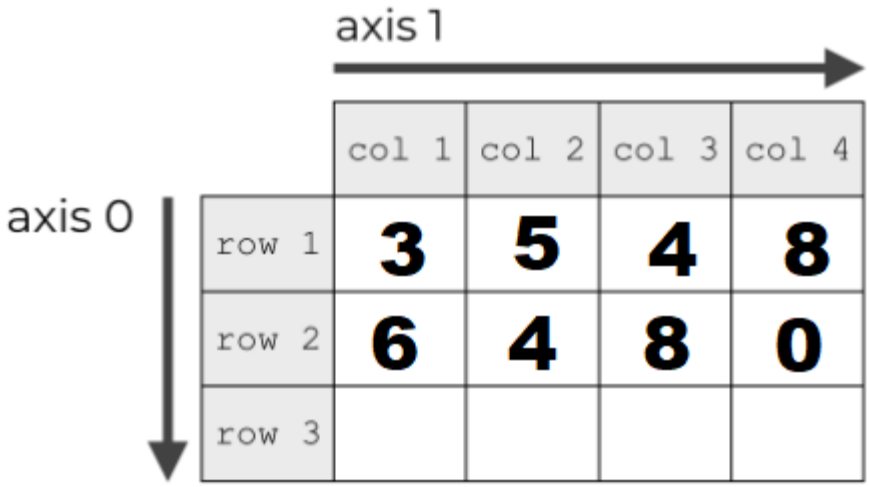
AXES

https://www.sharpsightlabs.com/blog/numpy-axes-explained/ (https://www.sharpsightlabs.com/blog/numpy-axes-explained/)

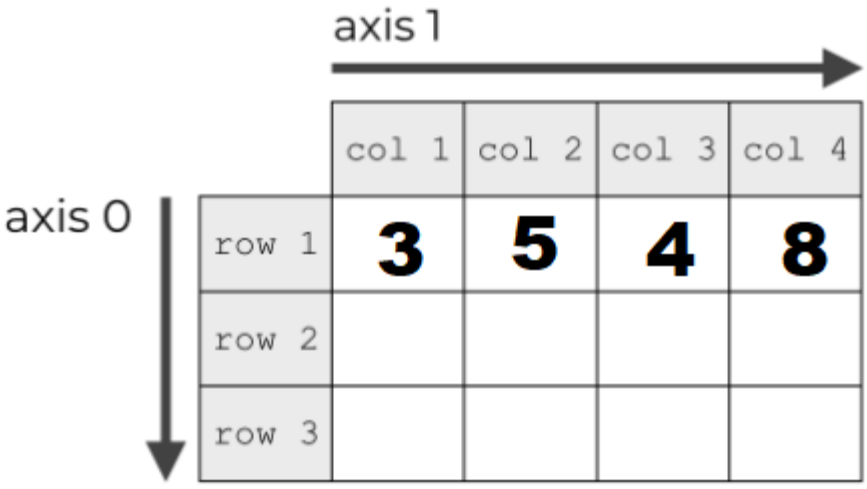
Un array de una dimensión es un vector, un array de dos dimensiones es una matriz con filas y columnas, o, mejor dicho, con dos axis.



Un array bidimensional [[3, 5, 4, 8], [6, 4, 8, 0]], se vería así:



CUIDADO: no muy accurate: Un array unidimensional [3, 5, 4, 8], se vería así:



(NOTA: en realidad existiría un único axis 0, puesto que es un vector)

concatenate()

In [3]:
```
import numpy as np

arr1 = np.array([[3, 5, 4, 8], [6, 4, 8, 0]])

arr2 = np.array([[6, 6, 6, 6], [2, 2, 2, 2]])

arr = np.concatenate((arr1, arr2))
arra = np.concatenate((arr1, arr2), axis=1)

print(arr)
print("")
```

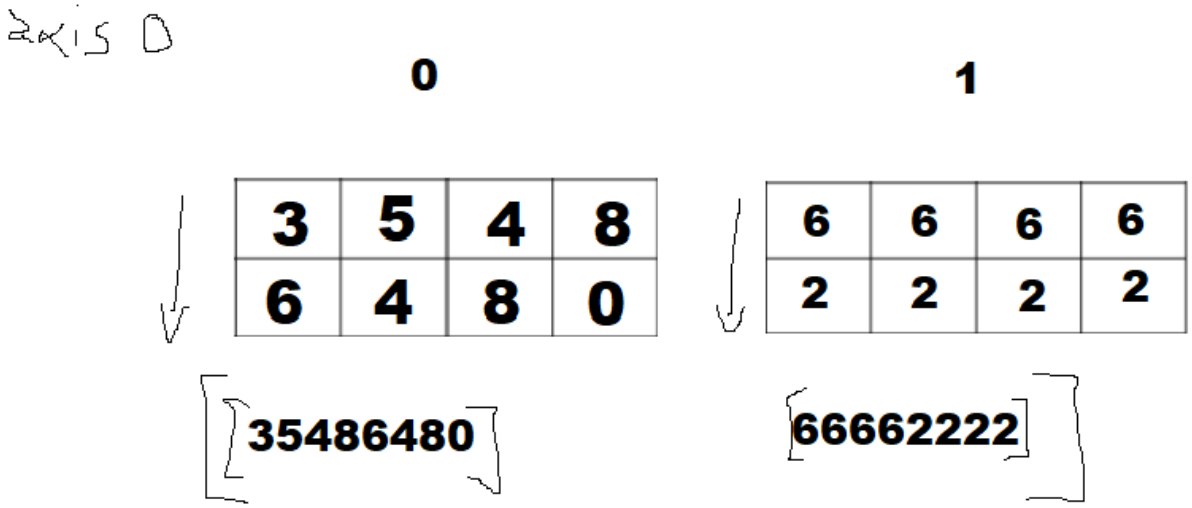
```
[[3 5 4 8]
 [6 4 8 0]
 [6 6 6 6]
 [2 2 2 2]]

[[3 5 4 8 6 6 6 6]
 [6 4 8 0 2 2 2 2]]
```

Tenemos dos arrays bidimensionales.

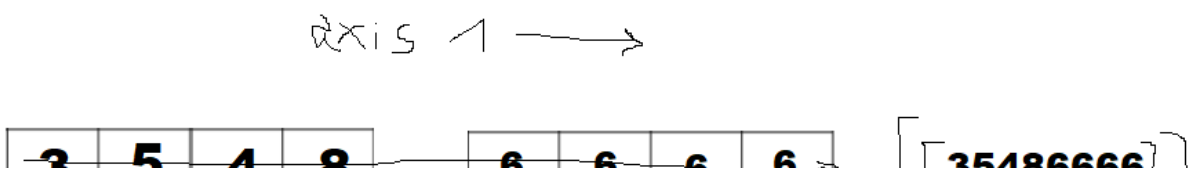
- En la primera concatenación (arr) los dos arrays se han concatenado en un solo array a partir del axis por defecto: axis 0.
- En la segunda concatenación (arra) los dos arrays han sido concatenados a partir del axis 1.

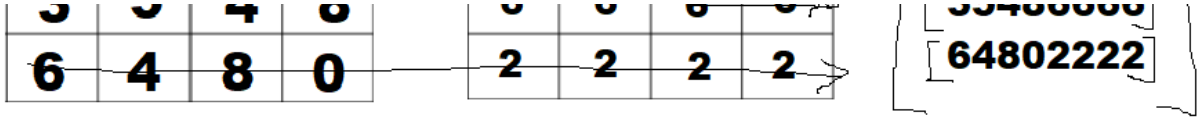
Veamos cómo funciona:

Concatenación axis 0:



Concatenación axis 1:





stack()

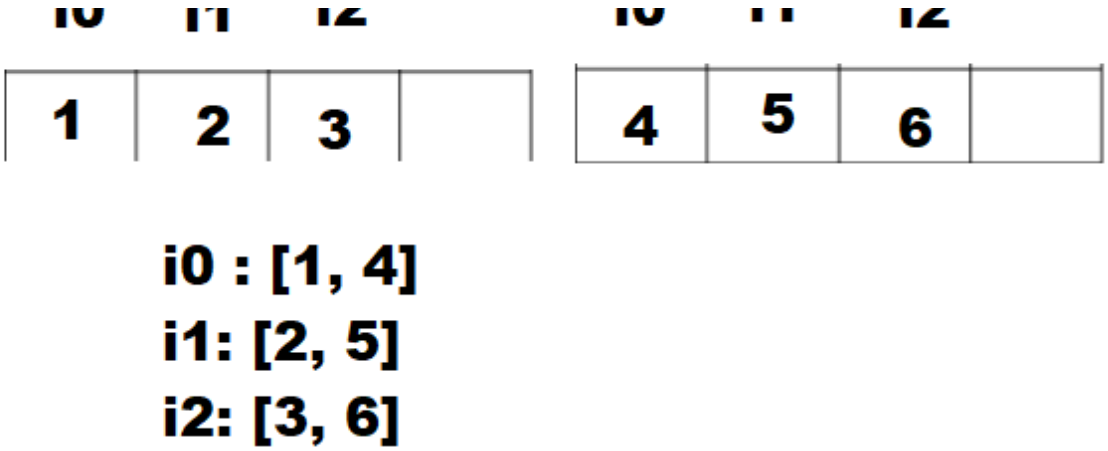
La función *stack()* es como concatenar pero lo que hace es poner un array encima de otro, no los "concatena".

- *hstack()* concatena a través de las filas
- *vstack()* concatena a través de columnas
- *dstack()* concatena a través de la altura/profundidad

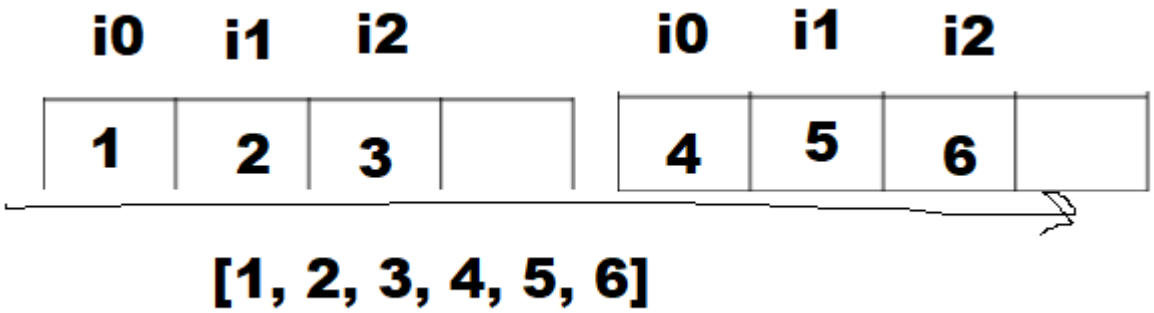
In [6]:
```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr_a = np.stack((arr1, arr2), axis=1)
arr_b = np.hstack((arr1, arr2))
arr_c = np.vstack((arr1, arr2))
arr_d = np.dstack((arr1, arr2))

print("Stack():")
print(arr_a)
print("")
print("hstack():")
print(arr_b)
print("")
print("vstack():")
print(arr_c)
print("")
print("dstack():")
```

```
Stack():
[[1 4]
 [2 5]
 [3 6]]

hstack():
[1 2 3 4 5 6]

vstack():
[[1 2 3]
 [4 5 6]]

dstack():
[[[1 4]
  [2 5]
  [3 6]]]
```

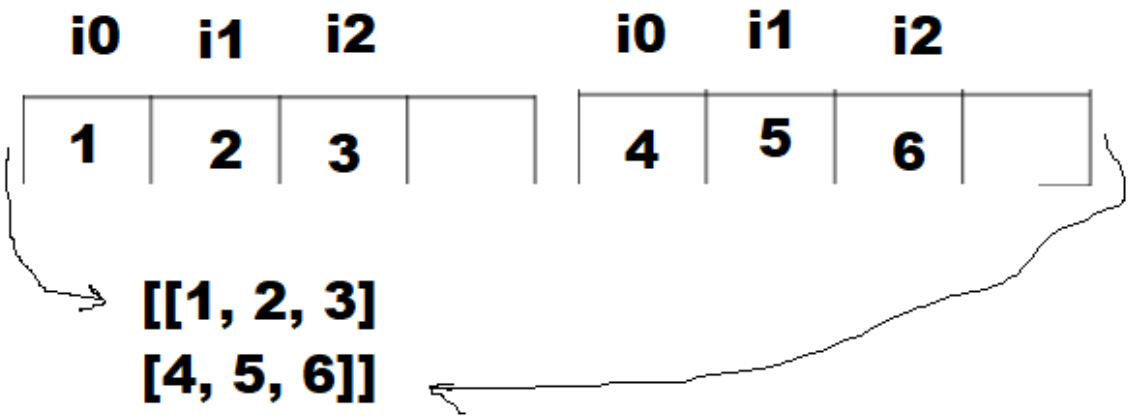
Esto es lo que está pasando en stack() :



Esto es lo que está pasando en hstack():



Esto es lo que está pasando en vstack():



array_split()

Lo contrario al joining: split divide un array en múltiples arrays.

In [5]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

#el resultado es un array que contiene cada uno de los arrays divididos
print(newarr)
```

```
print("")

#acceder a los arrays divididos
print(newarr[0])
print(newarr[1])
print(newarr[2])
print("")

#If the array has less elements than required, it will adjust from the en
newarr2 = np.array_split(arr, 4)
print(newarr2)
```

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

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

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

In [24]:

```
#splitting 2-D arrays

import numpy as np

arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]]) #6 fi

newarr = np.array_split(arr, 3) #devuelve 3 arrays de DOS DIMENSIONES
print("Array dividido por el axis 0: \n", newarr)

#también se puede dividir un array bidimensional por el axis 1!
newarr2 = np.array_split(arr, 2, axis=1)
print("\nArray dividido axis 1: \n", newarr2)

#que es lo mismo que hsplit() (opuesto a hstack)
otroMas = np.hsplit(arr, 2)
print("\n Es lo mismo: \n", otroMas)
```

```
Array dividido por el axis 0:
 [array([[1, 2],
        [3, 4]]), array([[5, 6],
        [7, 8]]), array([[ 9, 10],
        [11, 12]])]

Array dividido axis 1:
 [array([[ 1],
        [ 3],
        [ 5],
        [ 7],
```

NOTA: También tenemos vsplit() y dsplit().

BUSCAR EN ARRAYS

where()

- Puedes buscar en un array un valor y te devuelve una tupla con los índices donde haya hecho match.

searchsorted()

- Funciona solamente con arrays ordenados.
- Tu le das un valor, realiza una búsqueda binaria en el array y te devuelve el índice en el que el valor aportado debería ser insertado para mantener el orden del array.
- The method starts the search from the left and returns the first index where the number 7 is no longer larger than the next value.
- Puedes utilizar **side='right'** para hacer la búsqueda de derecha a izquierda y que devuelva la posición por la derecha, ES DECIR: en el caso del 7, como ya hay otro 7 en el array, lo puedes situar a su izquierda o derecha sin que el orden se vea perturbado, para situarlo a la derecha (indice 2) -en lugar de la izquierda (indice 1)- utilizamos esto.
- Para hacer searchsorted de más de un valor, pasa un array con los valores que quieres buscar.

In [30]:
```
import numpy as np

#WHERE()
arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x, "\n \nEsta tupla significa que el valor 4 está presente en los í
```

```
(array([3, 5, 6], dtype=int64),)

Esta tupla significa que el valor 4 está presente en los índices 3, 5
y 6 del array.
```

In [35]:
```
#SEARCHSORTED()
```

```
arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print("(Buscando desde la izquierda) El valor 7 debería ser insertado en

y = np.searchsorted(arr, 7, side='right')

print("(Buscando desde la derecha) El valor 7 debería ser insertado en el

z = np.searchsorted(arr, [2, 4, 8])
print("Índices para los valores 2, 4, 8: ", z)
```

```
(Buscando desde la izquierda) El valor 7 debería ser insertado en el í
ndice:  1
(Buscando desde la derecha) El valor 7 debería ser insertado en el índ
ice:  2
Índices para los valores 2, 4, 8:  [0 0 2]
```

ORDENAR ARRAYS: sort()

- La ordenación puede operar numéricamente o alfabéticamente, en orden ascendente o descendente.
- El método *sort()* devuelve una COPIA.

In [37]:
```
import numpy as np

#números
arr1 = np.array([3, 2, 0, 1])

print(np.sort(arr1))

#strings
arr2 = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr2))

#booleans
arr3 = np.array([True, False, True])

print(np.sort(arr3))

#arrays bidimensionales
arr4 = np.array([[3, 2, 4], [5, 0, 1]])
```

```
[0 1 2 3]
['apple' 'banana' 'cherry']
[False  True  True]
[[2 3 4]
 [0 1 5]]
```

FILTRAR ARRAYS

- Crea un nuevo array a partir del filtrage de otro array.
- Se realiza a partir de una lista de booleans, que filtrará a partir de los índices: los valores que correspondan en su índice a un valor True en el mismo índice serán incluidos, los valores que correspondan a False serán excluidos.

In [38]:
```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]
```

```
[41 43]
```

Normalmente se utiliza creando un filtro basado en condiciones:

In [39]:
```
arr = np.array([41, 42, 43, 44])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise Fa
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
```

```
[False, False, True, True]
[43 44]
```

Hay una forma más sencilla de hacer esto:

In [40]:
```
#Se asigna una condición a una variable y se pasa dicha variable como fil
filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)
```

```
[False False  True  True]
[43 44]
```

RANDOM

(pseudo)

- random.randint()
- random.rand()

In [41]:
```
from numpy import random

#Generate a random integer from 0 to 100
x = random.randint(100)

#Generate a random float from 0 to 1
x = random.rand()
```

GENERAR ARRAYS RANDOM

- The *randint()* method takes a **size parameter** where you can specify the shape of an array.

In [1]:
```
from numpy import random

#genera un array unidimensional de 5 numeros del 0 al 100
x=random.randint(100, size=(5))

print(x)
```

```
[62 21  9  8 77]
```

In [2]:
```
#genera un array BIDIMENSIONAL de 3 filas, cada una con 5 números del 0 a
x = random.randint(100, size=(3, 5))

print(x)
```

```
[[79 16 64  1 49]
 [31  7 92 10 53]
 [75 13 25 24 82]]
```

- The *rand()* method also allows you to specify the **shape of the array**.

In [6]:
```
from numpy import random

#genera un array UNIDIMENSIONAL con 5 floats
x = random.rand(5)

print("Un array unidimensional: \n", x, "\n")

#genera un array BIDIMENSIONAL de 3 filas, cada una con 5 floats
z = random.rand(3, 5)

print("Un array bidimensional: \n", z, "\n")
```

```
Un array unidimensional:
 [0.49120419 0.9236732  0.88412867 0.33451445 0.40789125]

Un array bidimensional:
 [[0.00643196 0.24319795 0.19486636 0.25428688 0.87720035]
 [0.18699581 0.06136688 0.90205468 0.28471377 0.65140111]
 [0.57800026 0.51224985 0.95251358 0.59404643 0.89950663]]
```

ELEGIR UN VALOR ALEATORIO ENTRE UN ARRAY DE VALORES

- The *choice()* method takes an array as a parameter and randomly returns one of the values.

In [7]:
```
from numpy import random

x = random.choice([3, 5, 7, 9])

print(x)
```

```
3
```

- The *choice()* method also allows you to return an array of values.
- Add a **size parameter** to specify the **shape** of the array.

In [8]:
```
#crea un array BIDIMENSIONAL de 3 filas de 5 valores aleatorios escogidos
x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

```
[[9 9 5 7 3]
 [3 7 7 5 9]
 [5 9 7 3 9]]
```

RANDOM DATA DISTRIBUTION

- Podemos generar números random definiendo las probabilidades de cada valor.
- La probabilidad de cada valor estará entre 0 y 1 (0 significa que nunca ocurrirá y 1 que siempre ocurrirá)

In [1]:
```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))

print(x)
```

```
[5 3 3 7 7 7 5 7 5 5 3 7 7 7 5 5 7 5 5 7 5 5 7 5 5 7 7 7 7 7 7 7 5 7 5
7 5
 7 7 7 7 5 7 7 5 7 7 5 7 5 5 7 5 7 7 7 3 7 7 7 7 5 7 7 3 7 5 7 7 7 7 7
5 7
 7 7 7 3 7 7 7 3 7 5 5 7 3 7 7 7 5 7 7 3 7 7 7 7 7 7]
```

- Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.

- The probability for the value to be 3 is set to be 0.1
- The probability for the value to be 5 is set to be 0.3
- The probability for the value to be 7 is set to be 0.6
- The probability for the value to be 9 is set to be 0

Nota: la probabilidad total de los valores debería ser 1.

In [2]:
```
#en 2 DIMENSIONES

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(3, 5))

print(x)
```

```
[[7 5 7 5 7]
 [7 5 7 3 7]
 [5 5 3 7 5]]
```

RANDOM PERMUTATIONS

- *shuffle()* cambia la posición de los valores de un array // **Nota: modifica el array original**
- *permutation()* realiza la misma acción pero **devuelve un re-arranged array, dejando el original intacto**

In [6]:
```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print("Permutación: ", random.permutation(arr))
print("Array original: ", arr) #se imprime el array original porque la pe

random.shuffle(arr)
print("Suffle: ", arr) #se ha suffleado el array original
```

```
Permutación:  [5 4 2 3 1]
Array original:  [1 2 3 4 5]
Suffle:  [3 4 5 2 1]
```

SEABORN MODULE

https://www.w3schools.com/python/numpy_random_seaborn.asp (https://www.w3schools.com/python/numpy_random_seaborn.asp) (para ver cómo descargarlo y toda la info)

- Es una librería que utiliza matplotlib por debajo
- Se utiliza para representar un gráfico de una distribución aleatoria

DISTPLOTS

Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to

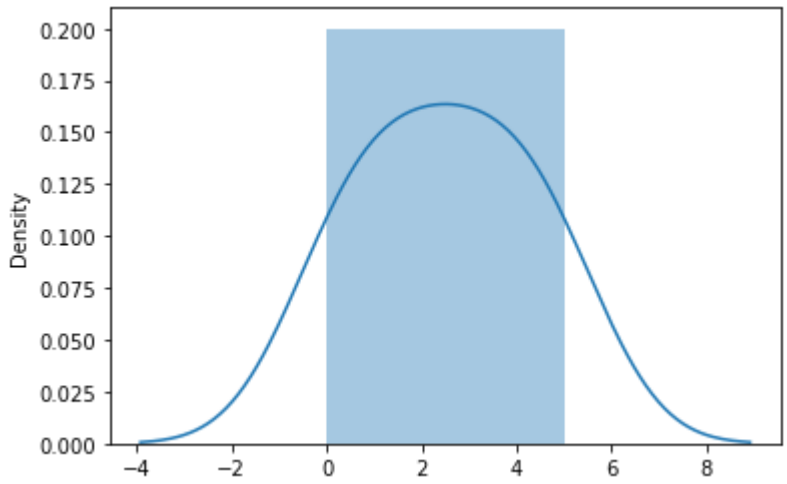
the distribution of points in the array

In [7]:
```
#qué importar para poder utilizar seaborn:

import matplotlib.pyplot as plt
```

```
Matplotlib is building the font cache; this may take a moment.
```

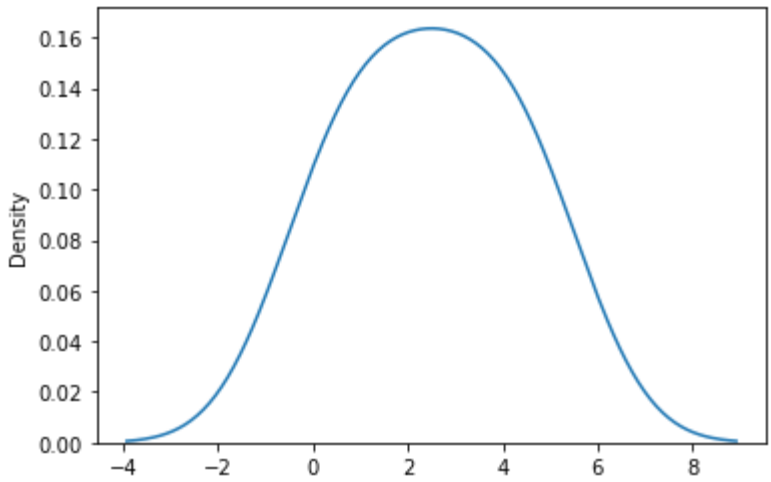
In [12]:
```
sns.distplot([0, 1, 2, 3, 4, 5])
```

```
C:\Users\elgab\anaconda3\lib\site-packages\seaborn\distributions.py:25
51: FutureWarning: `distplot` is a deprecated function and will be rem
oved in a future version. Please adapt your code to use either `displo
t` (a figure-level function with similar flexibility) or `histplot` (a
n axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5], hist=False) #sin historiograma

plt.show()
```

```
C:\Users\elgab\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



RANDOM.NORMAL()

Sirve para obtener una distribución normal random. Tiene 3 parámetros:

- loc: la media, donde se encuentra el pico.
- scale: la desviación
- size: la forma del array devuelto

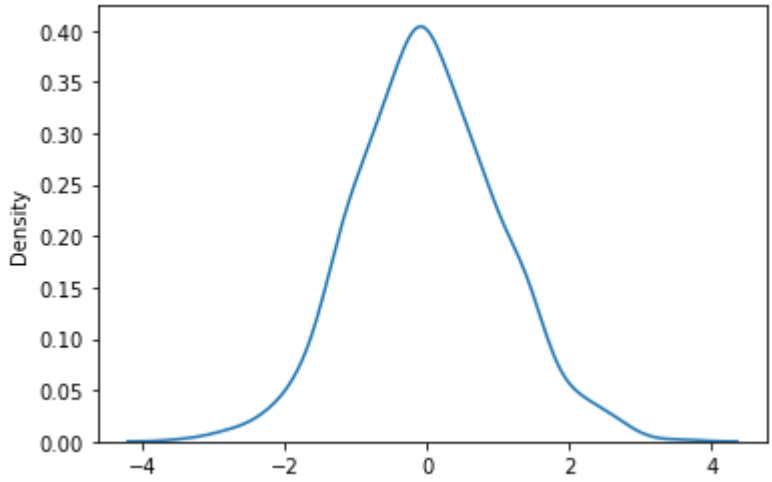
In [4]:
```
from numpy import random

x = random.normal(loc=1, scale=2, size=(2, 3))

print(x)
```

```
[[ 0.98055804 -0.2696866  -3.3947527 ]
 [ 2.25262312  1.08449944  2.41488347]]
```

Visualización:

In [5]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(size=1000), hist=False)

plt.show()
```

```
C:\Users\elgab\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



DISTRIBUCIÓN BINOMIAL

Muy buena web: https://www.statology.org/binomial-distribution-python/ (https://www.statology.org/binomial-distribution-python/)

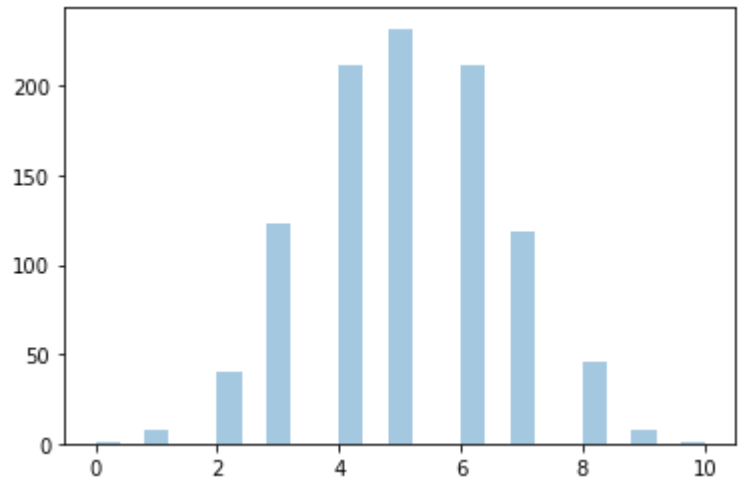
It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails. Tres parámetros:

- n: nº de intentos
- p: probabilidad de cada caso
- size: cantidad de experimentos

Nota:

- Distribución discreta (binomial): la distribución está definida por eventos separados (caras o cruces).
- Distribución continua (normal): e. g. la altura de una persona; 170, 170.1, 170.11, etc etc.

In [3]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = random.binomial(n=10, p=0.5, size=1000) #así se crea una distribución

sns.distplot(x, hist=True, kde=False) #representación

plt.show()
```



Explicación parámetros n, p, size:

Diferencia entre intento y experimento:

- Intento = (P), es decir, el espectro de posibles (p. ej. cuántas veces saldrá cara o cruz si tiro la moneda 10 veces)
- Experimento = número de veces que se realizará una prueba (p. ej. hacer 1000 veces el experimento de tirar la moneda 10 veces para ver cuántos éxitos y fracasos hay)

  El eje de las X equivale al número de éxitos que se han conseguido en 10 intentos. El eje de las Y equivale al número de veces que ha salido X número de éxitos de 10 intentos (éxitos/intentos) durante 1000 experimentos.

Por ende, n = intentos, size = experimentos y p es la probabilidad de éxito (en cara o cruz es 0.5)

POISSON DISTRIBUTION

Explicación de la distribución de poisson: https://www.statology.org/poisson-distribution/ (https://www.statology.org/poisson-distribution/)

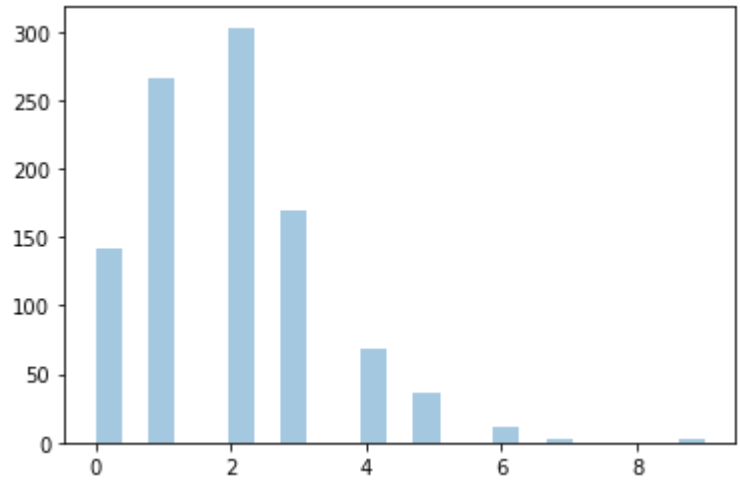
Poisson Distribution is a Discrete Distribution.

It estimates how many times an event can happen in a specified time. For example, suppose a particular hospital experiences an average of 10 births per hour. What is the probability that more than 12 births occur in a given hour? What is the probability that less than 5 births occur in a given hour? Etc.

Sigamos el siguiente ejemplo:

suppose a particular hospital experiences an average of 2 births per hour. We can use the formula above to determine the probability of experiencing 0, 1, 2, 3 births, etc. in a given hour:

In [4]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = random.poisson(lam=2, size=1000)

sns.distplot(x, kde=False)
```



La probabilidad de cada evento (0, 1, 2, 3... nacimientos) es la siguiente:

- 0 nacimientos = 140 / 1000 = 0.14
- 1 nacimiento = 270 / 1000 = 0.27
- 2 nacimientos = 300 / 1000 = 0.3
- Etc.

Nota: CREO que esta distribución también sirve para saber las probabilidades dada una serie de sucesos conocidos (sin saber la media???), p. ej. acaban de nacer 2 bebés en este hospital, ¿nacerá otro?

Los parámetros utilizados para la función son los siguientes:

- lam = rate or known number of occurences e.g. 2 for above problem.
- size = The shape of the returned array. A.K.A. número de experimentos.

UNIFORM DISTRIBUTION

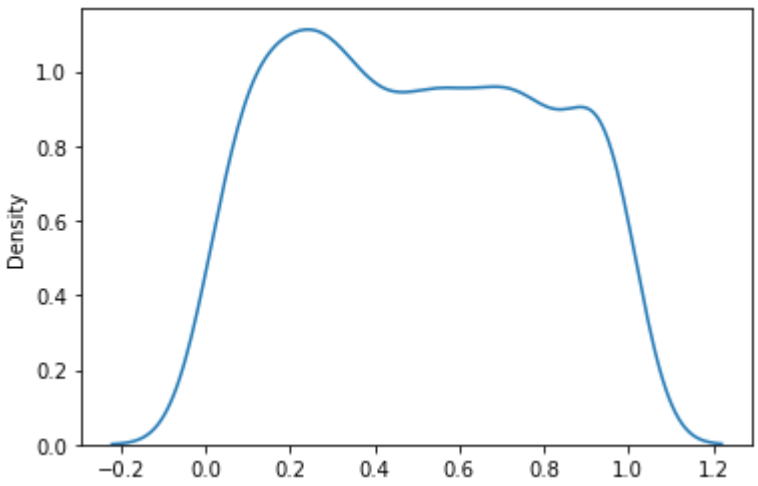
Used to describe probability where every event has equal chances of occuring. (E.g. Generation of random numbers)

It has three parameters:

- a - lower bound - default 0 .0.
- b - upper bound - default 1.0.
- size - The shape of the returned array.

In [2]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = random.uniform(size=1000)
sns.distplot(x, hist=False)
```

```
C:\Users\elgab\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



LOGISTIC DISTRIBUTION

UFUNCS

Aritmética simple:

https://www.w3schools.com/python/numpy/numpy_ufunc_simple_arithmetic.asp (https://www.w3schools.com/python/numpy/numpy_ufunc_simple_arithmetic.asp)

- **np.add()**: sums the content of two arrays, and return the results in a new array.
- **np.subtract()**: resta.
- **np.multiply()**
- **np.divide()**

- **np.power()**: rises the values from the first array to the power of the values of the second array, and return the results in a new array.
- **np.mod() // np.remainder()**: retorna el módulo.
- **np.divmod()**: retorna el cociente (resultado de una división 10/3=3) y el módulo(1)
- **np.absolute()**: retorna el valor absoluto

Rounding Decimals:

- **Truncation**: elimina los decimales y retorna un int: **np.trunc() // np.fix()**
- **rounding**: redondea los decimales y le puedes pasar el número de decimales como parámetro. **np.around()**
- **floor**: redondea abajo y retorna un float **np.floor()**
- **ceil**: redondea arriba y retorna float **np.ceil()**

EXTRA

- transpose() // .transpose() // .T (https://realpython.com/numpy-scipy-pandas-correlation-python/ (https://realpython.com/numpy-scipy-pandas-correlation-python/))