File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [82]:
```
#in this module I'll try to explain and show mostly every method needed while working with NumPy. Good Luck!
```

In [83]:
```
import numpy as np
```

numpy.ndarray

In [84]:
```
#method np.all() - The np.all() function takes an array as input, and returns True if all elements
#of the array are non-zero (or equivalently, evaluate to True), and False otherwise.

a = np.array([1,2,3])
b = np.array([0,2,3])
c = np.array([[1,2],[3,4]])
d = np.array([[1,0],[3,4]])

print(np.all(a))
print(np.all(b))
print(np.all(c))
print(np.all(d))
```

```
True
False
True
False
```

In [85]:
```
#method np.any() - The np.any() function takes an array as input, and returns True if at least one
print(np.any(b))
print(np.any(c))
print(np.any(d))
```

```
True
False
True
True
```

In [86]:
```
#method np.astype() - is a method that is used to cast an array to a different data type.
#It returns a new array with the same values as the original array, but with a different data type.

a = np.array([1,2,3,4])
b = a.astype(float)   #(int)(bool)(str)...

print(a.dtype)  #it was a integer data type
print(b.dtype)  #now it's a floating data type
```

```
int32
float64
```

In [87]:
```
#method np.choose() - is a NumPy function that constructs a new array from an index array and a sequence of arrays.

chose = np.array([0,1,2])

a = np.array([1,2,3])
a = np.array([1,2,3,5,6,7])
b = np.clip(a,2,4)
print(b)
```

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

In [89]:
```
#method np.copy() - is a NumPy function that creates a copy of a given NumPy array.

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

print(a)
print(b)

```

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

In [90]:
```
#method np.cumprod() -  is a NumPy function that returns the cumulative product of elements along
#a given axis of a NumPy array.

a = np.array([1,2,3,4])
b = np.cumprod(a)


print(b)
```

```
[ 1  3  6 10]
```

In [92]:
```
#method np.diagonal() -  is a NumPy function that returns the diagonal elements of a 2D array or a
#1D array representing the diagonal of a 2D array.

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
b = np.diagonal(a)

print(b)
```

```
[1 5 9]
```

In [93]:
```
#method np.fill() - can be used to fill an array with a scalar value.

a = np.full((3,4),[1,2,3,4])
print(a)
```

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

In [94]:
```
#method np.flatten - is a NumPy function that is used to return a 1-dimensional copy of an input array.
#This function is used to convert a multidimensional array into a flat, one-dimensional array.
#to extract a single item from an array.

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

print(b)
```

```
4
```

In [96]:
```
#method np.max() - is a NumPy function that returns the maximum value of an array or along a specified axis.

a = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
b = np.max(a)
c = np.max(a, axis=0)
d = np.max(a, axis=1)

print(b)
print(c)
print(d)
```

```
9
[7 8 9]
[3 6 9]
```

```
print(c)
print(d)
```

```
1
[1 2 3]
[1 4 7]
```

In [98]:
```
#method np.mean() - is a NumPy function that returns the arithmetic mean (average) of the elements in a NumPy array.
#It can be used to calculate the mean along a specified axis or across the entire array.


a = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
b = np.mean(a)
c = np.mean(a, axis=0)
d = np.mean(a, axis=1)

print(b)
print(c)
print(d)
```

```
5.0
[4. 5. 6.]
[2. 5. 8.]
```

In [100]:
```
#method np.prod() - is a NumPy function that calculates the product of all the elements in a NumPy array.

a = np.array([1,2,3,4])
b = np.prod(a)

print(b)
```

```
24
```

In [101]:
```
#method np.put() -  is a NumPy function that replaces specified elements of an array with given values.
```

```
a = np.array([1, 2, 3, 4, 5])
np.put(a,[1,3],[10,20])

print(a)
```

```
[ 1 10  3 20  5]
```

In [102]:
```
#method np.ravel(just a view of a original, not original like np.ravel) - is a NumPy function that returns a
#flattened 1-dimensional array of a given input array. The resulting array contains all the elements of the
#input array in a flattened, sequential order.

a = np.array([[1,4],[9,5]])
b = np.ravel(a)

print(b)
c = np.arange(12)
b = np.reshape(a, (4,3))
d = np.reshape(c,(3,4))

print(b)
print(d)
```

```
[[ 1  2  3]
 [ 4  4  5]
 [ 6  7  7]
 [ 8  9 10]]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [104]:
```
#method np.resize() -  is a NumPy function used to change the size of an array by repeating or truncating its
#elements as necessary. It returns a new array with the specified size, which may be larger or smaller than the
#original array.

a = np.arange(21)
b = np.resize(a,(3,7))

print(b)
```

```
[[ 0  1  2  3  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]]
```

```
print(b)
```

```
[[1.54 2.55 3.35 5.28]
 [6.7  8.5  9.   4.3 ]]
```

In [106]:
```
#method np.sort - is a NumPy function that returns a sorted copy of an input array. By default,
#it sorts the elements of the array along the last axis in ascending order.

a = np.array([1,5,2,8,3,9,1,6,3,8,2,9,200,7])
b = np.sort(a)

print(b)
```

```
[  1   1   2   2   3   3   5   6   7   8   8   9   9 200]
```

In [107]:
```
#method np.std() - is a NumPy function that calculates the standard deviation of an array.
#The standard deviation is a measure of how spread out the values in the array are from the mean.
#It is defined as the square root of the variance, which is the average of the squared differences from the mean.

a = np.array([[1,5,8,2,6,0],
              [4,5,6,8,3,1]])
b = np.std(a)
c = np.std(a,axis=0)

print(b)
print(c)
              [3,6,3]])
b = np.sum(a)
c = np.sum(a,axis=1)

print(b)
print(c)
```

```
26
[14 12]
```

In [109]:
```
#method np.transpose() - is a NumPy function that rearranges the dimensions of an array.
#It takes the input array and rearranges the axes according to the specified order.
#The function returns a new array with the rearranged axes.

a = np.array([[1,5,3,6,7],
              [5,6,3,2,1],
              [2,4,7,2,5]])

b = np.transpose(a)
```

```
b = np.transpose(a)

print(b)
print(c)
```

```
[[1 5 2]
 [5 6 4]
 [3 3 7]
 [6 2 2]
```

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

In [111]:

```
#method np.zeros() - is a function from the NumPy library in Python, which returns a new array of a specified shape,
#where all the elements of the array are set to zero.

a = np.zeros((4,5))

print(a)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [112]:

```
#method np.arange() -  is a function from the NumPy library in Python, which returns an array of evenly spaced
#values within a specified interval.

a = np.arange(22)
print(a)

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

c = np.concatenate((a,b),axis=1)
d = np.concatenate((a,b),axis=0)
print(c)
print(d)
```

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

In [114]:

```
#method np.hstack() - is a function from the NumPy library in Python, which allows you to horizontally
#stack two or more arrays (i.e., concatenate them along the second axis, also known as the column axis).

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

c = np.hstack((a,b))
print(c)
```

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

```
c = np.vstack((a,b))
print(c)
```

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

In [116]:

```
#method np.eye() -  is a function from the NumPy library in Python, which allows you to create a 2-dimensional
#array with ones on the diagonal and zeros elsewhere (i.e., an identity matrix).

a = np.eye(4,3)
print(a)

b = np.eye(5,5,k=2)
print(b)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
[[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [118]:
```
#method np.where() - is a function from the NumPy library in Python that returns the indices where a given
#condition is true in an array.

a = np.array([1,6,3,8,5,9])
b = np.where(a>5)

print(b)
```

```
(array([1, 3, 5], dtype=int64),)
```

In [119]:
```
#method np.flip() - is a function from the NumPy library in Python that is used to reverse the order of
#elements in an array along a specified axis.

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

b = np.flip(a, axis=1)
print(b)

```

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

```
[1 5 9]
```

Randoms

In [121]:
```
#method np.random.rand() - generates an array of random numbers uniformly distributed between 0 and 1.

a = np.random.rand(2,2,2)
print(a)
```

```
[[[0.6136045  0.4439248 ]
  [0.84630546 0.05162755]]

 [[0.66401088 0.57563289]
  [0.80250803 0.55720754]]]
```

In [122]:
```
#method np.random.randn() - is a function from the NumPy library in Python, which generates an array of random
#numbers that are normally distributed with mean 0 and standard deviation 1.
```

In [123]:
```
#method np.random.randint() - is a function from the NumPy library in Python, which generates an array of random
#integers between a specified minimum and maximum value.

a = np.random.randint(low=0,high=10, size=(2,3))
print(a)
```

```
[[1 6 2]
 [5 9 2]]
```

In [124]:
```
#method np.random.choice() - is a function from the NumPy library in Python that generates a random sample from a
#given 1-D array or sequence.

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

b = np.random.choice(a,3,replace=False) #false - no relpy, true - reply

print(b)
```

```
[5 3 8]
```

In [125]:
```
#method np.random.shuffle() - is a function from the NumPy library in Python that shuffles the elements of a
#given array in place.

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

np.random.shuffle(a)
print(a)
```