Complete numpy Cheatsheet Notebook With Examples !

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

creating an array

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

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

In [3]:
```
# 2D array
arr=np.array([[2,3,4],[3,4,5]])
print(arr)
```

```
[[2 3 4]
 [3 4 5]]
```

In [4]:
```
# arange() function create matrix in given range
ar=np.arange(1,21)
print(ar)
```

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

In [5]:
```
# full() function is used to creat and fill an array with specfied range
np.full((10,10),10000)
```

Out[5]:
```
array([[10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000],
       [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
        10000]])
```

reshape function

In [6]:
```
a=ar.reshape(2,2,5)
print(a)
```

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

 [[11 12 13 14 15]
  [16 17 18 19 20]]]
```

In [7]:
```
# Dimension  Checking
arr.ndim
```

Out[7]: 2

Indexing of an array

In [8]:
```
# feting values from an array
arr[0,1]
```

Out[8]: np.int64(3)

In [9]:
```
arr[1,2]
```

Out[9]: np.int64(5)

In [10]:
```
### fetching multiple values at a time
arr[0:2,:]
```

Out[10]:
```
array([[2, 3, 4],
       [3, 4, 5]])
```

Shape function

In [11]:
```
arr.shape
```

Out[11]: (2, 3)

In [12]:
```
# split() : it splits the array in given parts
np.split(arr,2)
```

Out[12]: [array([[2, 3, 4]]), array([[3, 4, 5]])]

In [13]:
```
# ravel : it converts multidimensional array into 1D array by creating a  view
print('Actual array ',arr)
print('Ravel :',arr.ravel())
```

```
Actual array  [[2 3 4]
 [3 4 5]]
Ravel : [2 3 4 3 4 5]
```

In [14]:
```
# flatten() : it also converts multidimensional array into 1D but creates a copy
c=arr.flatten()
print('Actual: ',arr)
print('Flatten :',c)
```

```
Actual:  [[2 3 4]
 [3 4 5]]
Flatten : [2 3 4 3 4 5]
```

In [15]:
```
# unique() : it gives all unique values from an array
np.unique(arr,return_index=True)
```

Out[15]:
```
(array([2, 3, 4, 5]), array([0, 1, 2, 5]))
```

In [16]:
```
# Delelte () : it delets given value from array
d=np.delete(arr,[3,5])
print('Actual :\n ',arr)
print('deleted :\n ',d)
```

```
Actual :
  [[2 3 4]
 [3 4 5]]
deleted :
  [2 3 4 4]
```

In [17]:
```
# reshape() : reshape function changes the shape of array a/c to given critera
arr.reshape(3,2)
```

Out[17]:
```
array([[2, 3],
       [4, 3],
       [4, 5]])
```

In [18]:
```
# Transpose of a 2D array
transposed=np.transpose(arr)
print('Original :\n',arr)
print('Transposed :\n',transposed)
```

```
Original :
 [[2 3 4]
 [3 4 5]]
Transposed :
 [[2 3]
 [3 4]
 [4 5]]
```

Matric creation

In [19]:
```
# np.ones() : it create an array of given dimensions and fills it with one
D2=np.ones((4,4))
D3=np.ones((2,2,2))
print('2D array or matrix :\n',D2)
print('3D array or tensor :\n',D3)
```

```
2D array or matrix :
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
3D array or tensor :
 [[[1. 1.]
  [1. 1.]]

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

In [20]:
```
# np.zeros() : it creates same  an array in given range and fills it with zeros
np.zeros((2,2,2))
```

Out[20]:
```
array([[[0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.]]])
```

In [21]:
```
# np.identity() : it is used to create indentity matrix
np.identity(6)
```

Out[21]:
```
array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

In [22]:
```
# np.empty() : it is used to create empty arrayes
np.empty((3,3,3)) # but it does not create complete empty matrix instead it fill
```

Out[22]:
```
array([[[ 3.15116504e-312,  8.69555537e-322,  2.84809997e-306],
        [ 2.78134232e-308,  5.45353533e-312,  8.28904606e-317],
        [ 2.64227733e-308,  1.34497462e-284,  3.33761079e-308]],

       [[ 1.08858391e-311,  4.36832727e-314,  1.33504432e-307],
        [ 5.43443129e-311,  6.49032306e-314, -1.27020841e-234],
        [-1.04071560e+306,  2.98826672e+193, -3.11773210e+116]],

       [[ 2.71876654e-311,  4.46033568e-313, -4.21606021e-266],
        [ 2.12696454e-311,  1.34497462e-284, -1.09290407e+052],
        [ 1.51087288e+113,  1.68049892e+241,  7.79239294e-310]]])
```

Stack() function

In [23]:
```
arr1=np.array([3,3,3,5])
arr2=np.array([2,3,4,5])
print('first array :\n',arr1)
print('second array :\n',arr2)
```

```
first array :
 [3 3 3 5]
second array :
 [2 3 4 5]
```

In [24]:
```
# vstack() : it stack concatenates tow arrays on y axis or in simple  verticaly
np.vstack((arr1,arr2))
```

Out[24]:
```
array([[3, 3, 3, 5],
       [2, 3, 4, 5]])
```

In [25]:
```
# hstack() : it stack concatenates tow arrays on x axis or in simple  horizontly
np.hstack((arr1,arr2))
```

Out[25]:
```
array([3, 3, 3, 5, 2, 3, 4, 5])
```

In [26]:
```
# dstack() : it stack concatenates tow arrays on Z axis .
np.dstack((arr1,arr2))
```

Out[26]:
```
array([[[3, 2],
        [3, 3],
        [3, 4],
        [5, 5]]])
```

Concatenation of arrays

In [27]:
```
ar1=np.array([3,3,3,5]).reshape(2,2)
ar2=np.array([2,3,4,5]).reshape(2,2)
print('first array :\n',ar1)
print('second array :\n',ar2)
```

```
first array :
 [[3 3]
 [3 5]]
second array :
 [[2 3]
 [4 5]]
```

In [28]:
```
# verticaly concatenation
np.concatenate((ar1,ar2),axis=0)
```

Out[28]:
```
array([[3, 3],
       [3, 5],
       [2, 3],
       [4, 5]])
```

In [29]:
```
# verticaly concatenation
np.concatenate((arr1,arr2))
```

Out[29]:
```
array([3, 3, 3, 5, 2, 3, 4, 5])
```

Cummulative sum() function

In [30]:
```
cs=np.cumsum(ar1)
print('array :\n',ar1)
print('cummlative sum  :\n',cs)
```

```
array :
 [[3 3]
 [3 5]]
cummlative sum  :
 [ 3  6  9 14]
```

Linear Algebra functions

In [31]:
```
a=np.array([[2,3,4,5],[5,6,3,2],[6,7,8,0],[1,2,4,2]])
b=np.array([[1,21,2,4],[3,2,1,5],[6,8,9,7],[1,2,22,5]])
print(' A matrix :\n ',a)
print(' B matrix : \n',b)
```

```
A matrix :
 [[2 3 4 5]
[5 6 3 2]
[6 7 8 0]
[1 2 4 2]]
B matrix :
[[ 1 21  2  4]
[ 3  2  1  5]
[ 6  8  9  7]
[ 1  2 22  5]]
```

In [32]:
```
# Finding the sum of two Matrix
a+b
```

Out[32]:
```
array([[ 3, 24,  6,  9],
       [ 8,  8,  4,  7],
       [12, 15, 17,  7],
       [ 2,  4, 26,  7]])
```

In [33]:
```
# Multipication of two matrix : columns of first matrix must be equal to the row
np.dot(a,b)
```

Out[33]:
```
array([[ 40,  90, 153,  76],
       [ 43, 145,  87,  81],
       [ 75, 204,  91, 115],
       [ 33,  61,  84,  52]])
```

In [34]:
```
# Determinent of  a matrix
np.linalg.det(a)
```

Out[34]:
```
np.float64(65.00000000000004)
```

In [35]:
```
# inverse of a matrix
np.linalg.inv(a)
```

Out[35]:
```
array([[ 1.2       , -0.8       ,  0.8       , -2.2       ],
       [-1.16923077,  0.98461538, -0.75384615,  1.93846154],
       [ 0.12307692, -0.26153846,  0.18461538, -0.04615385],
       [ 0.32307692, -0.06153846, -0.01538462, -0.24615385]])
```

Boolean Indexing

In [36]:
```
# In Boolean Indexing , the indexing is performed on the basis of boolean values
array=np.array([2,3,5,6,8,9,10,11,12,22,32,4,54,36,78,89])
array
```

Out[36]:
```
array([ 2,  3,  5,  6,  8,  9, 10, 11, 12, 22, 32,  4, 54, 36, 78, 89])
```

In [37]:
```
Greater=array<8
print(Greater)
```

```
[ True  True  True  True False False False False False False False  True
 False False False False]
```

In [38]:
```
array[Greater]
```

Out[38]:
```
array([2, 3, 5, 6, 4])
```

Faancy Indexing

In [39]:
```
# In fancy indexing more then one vales are select randomly by specifing their i
print('value at index 3,2,7,1 :',array[[3,2,7,1]])
print('value at index 5,0,9,4 :',array[[5,0,9,4]])
```

```
value at index 3,2,7,1 : [ 6  5 11  3]
value at index 5,0,9,4 : [ 9  2 22  8]
```

Inserting data into an array

In [42]:
```
# insert() function is used to insert values into an array
np.insert(array,0,5000)
```

Out[42]:
```
array([5000,    2,    3,    5,    6,    8,    9,   10,   11,   12,   22,
         32,    4,   54,   36,   78,   89])
```

Delecting data from an array

In [46]:
```
# delete() function is used to delete data from an array
np.delete(array,len(array)-1)
```

Out[46]:
```
array([ 2,  3,  5,  6,  8,  9, 10, 11, 12, 22, 32,  4, 54, 36, 78])
```

Broadcasting

Arthmetic operations on an array is called Broadcasting which contains three rulles

Rule 1 : Make the tow arrays have the same number of dimensions

Rule 2 : if the No of dimensions of the tow arrays are different add new dimension with size 1 to head of the array with smaller dimension

Rule 3 : If still the size of each of two arrays do not match dimension with size 1 ate strached and makes it equal to the size of other array

Rule 4 : If None of above 3 Rule is applyed on both array ,then Broadcasting is not possible

Note : Python will will automaticaly followe the given rules internaly while performing arithmetic operations on arrays ,u do not manually do it.

In [53]:
```
z=np.array([500])
y=np.array([[2,3,4],[5,6,7]])
```

In [54]:
```
z+y
```

Out[54]:
```
array([[502, 503, 504],
       [505, 506, 507]])
```

In [57]:
```
a=np.array([[50],[40000]])
b=np.array([[2,4,5],[5,4,6]])
```

In [58]:
```
a+b
```

Out[58]:
```
array([[   52,    54,    55],
       [40005, 40004, 40006]])
```

In [59]:
```
a=np.array([[50,33],[40000,53]])
b=np.array([[2,4,5],[5,4,6]])
# these could not brodcast bacause there dimenson are diffenent and both do not
```

In [60]:
```
a+b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[60], line 1
----> 1 a+b

ValueError: operands could not be broadcast together with shapes (2,2) (2,3)
```

Most Important functions of arrays

1. np.isna

In [61]:
```
# np.nan function generates nan or null values in arrray
np.nan
```

Out[61]:
```
nan
```

2. np.isnan( )

In [72]:
```
# np.isnan() function is used to check the nan values in an array
g=np.array([[2,np.nan,5],[np.nan,3,np.nan]])
np.isnan(g)
```

Out[72]:
```
array([[False,  True, False],
       [ True, False,  True]])
```

3. np.linespace( )

In [77]:
```
# linespace function is used to generate values in an specfied range at equal in
np.linspace(-10,10,10)
```

Out[77]:
```
array([-10.        ,  -7.77777778,  -5.55555556,  -3.33333333,
        -1.11111111,   1.11111111,   3.33333333,   5.55555556,
         7.77777778,  10.        ])
```

4. np.exp( )

In [79]:
```
# exp() function function is used to find the exponent values of an array
x=np.array([2,3,4,5,6])
np.exp(x)
```

```
Out[79]: array([  7.3890561 ,  20.08553692,  54.59815003, 148.4131591 ,
                403.42879349])
```

5. np.sin( ) ,np.cos( )

```
In [88]: # np.sin() or other trignometric functions are used to find the trignometric val
         x=np.array([2,7,4,3,6,1])
         print('sinx :',np.sin(x))
         print('cosx : ',np.cos(x))
```

```
sinx : [ 0.90929743  0.6569866  -0.7568025   0.14112001 -0.2794155   0.84147098]
cosx :  [-0.41614684  0.75390225 -0.65364362 -0.9899925   0.96017029  0.54030231]
```

6. np.sort( )

```
In [92]: # sort function sorts the values of an array either ascendin or decending order
         x=np.array([2,7,4,3,6,1,0,10])
         np.sort(x)
```

```
Out[92]: array([ 0,  1,  2,  3,  4,  6,  7, 10])
```

7. np.append( )

```
In [93]: # append function is used to append any value in an array
         arr=np.array([2,7,4,3,6,1,0,10])
         np.append(array,606)
```

```
Out[93]: array([  2,   3,   5,   6,   8,   9,  10,  11,  12,  22,  32,   4,  54,
                 36,  78,  89, 606])
```

8.np.unique ()

```
In [96]: # unique( ) function gives all the unique values in an array
         arr=np.array([2,7,4,3,6,6,1,0,2,7,8,4,10])
         print('array :',arr)
         print('unique array :',np.unique(arr))
```

```
array : [ 2  7  4  3  6  6  1  0  2  7  8  4 10]
unique array : [ 0  1  2  3  4  6  7  8 10]
```

9. np.where( )

```
In [100… # where() function works exactly like if-else in python checkes for a condition
         ar=np.array([2,7,4,3,6,1,10])
         print('array :',ar)
         print('1 for Even and 0 for odd :',np.where(x%2==0,1,0))
```

```
array : [ 2  7  4  3  6  1 10]
1 for Even and 0 for odd : [1 0 1 0 1 0 1 1]
```

10. np.argmax( ) , np.argmin( )

```
In [102… # argmax() and argmin() functions are used to find the indices of max and min va
         y=np.array([2,7,4,3,6,1,10])
         print('Array :',y)
```

```
print('Max value index :',np.argmax(y))
print('Min value index :',np.argmin(y))
```

```
Array : [ 2  7  4  3  6  1 10]
Max value index : 6
Min value index : 5
```

11. np.cumsum( ) , np.cumprod( )

In [103…
```
# cumsum() function is used to find the cumelative sum of an array
t=np.array([2,7,4,3,6,1,10])
print('Array :',t)
print('Cumelative sum :',np.cumsum(t))
print('Cumelative product :',np.cumprod(t))
```

```
Array : [ 2  7  4  3  6  1 10]
Cumelative sum : [ 2  9 13 16 22 23 33]
Cumelative product : [    2    14    56   168  1008  1008 10080]
```

12. np.percentile( )

In [105…
```
# percentile() function is used to find the percentiles of any given array or li
c=np.array([2,7,4,3,6,1,10])
print('Array :',c)
print('Max value index :',np.percentile(c,25))
```

```
Array : [ 2  7  4  3  6  1 10]
Max value index : 2.5
```

13. np.isin( )

In [108…
```
# isin() checks wheather any given values is present in an array or not
a=np.array([2,7,4,3,6,1,10])
np.isin(a,5)
```

Out[108…
```
array([False, False, False, False, False, False, False])
```

14. np.flip( )

In [111…
```
# flip() function reverse the given array
a=np.array([2,7,4,3,6,1,10])
np.flip(a)
```

Out[111…
```
array([10,  1,  6,  3,  4,  7,  2])
```

15. np.clip( )

In [124…
```
# clip() function shrinks the array in a given max and min range for array
a=np.array([0,1,2,7,4,3,6,1,10])
cliped=np.clip(a,a_min=2,a_max=6)
print('Array :',a)
print('Cliped :',cliped)
```

```
Array : [ 0  1  2  7  4  3  6  1 10]
Cliped : [2 2 2 6 4 3 6 2 6]
```

Set functions

1. np.union1d( )

In [112…
```
A=np.array([1,2,3,5,6,7,8])
B=np.array([0,1,3,4,5,7,10])
```

In [113…
```
# union1d() function finds the union of two given arrays or sets
np.union1d(A,B)
```

Out[113…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8, 10])
```

2. np.intersect1d( )

In [114…
```
# intersect1d() fuction finds the intersection of two arrays or sets
np.intersect1d(A,B)
```

Out[114…
```
array([1, 3, 5, 7])
```

3. np.setdiff1d( )

In [116…
```
# setdiff1d() function finds those values of A which do not exsist in B ,like di
np.setdiff1d(A,B)
```

Out[116…
```
array([2, 6, 8])
```

4. np.setxor1d( )

In [118…
```
# setxor1d() function finds tow sided difference , those values of A which do no
np.setxor1d(A,B)
```

Out[118…
```
array([ 0,  2,  4,  6,  8, 10])
```

Random module of numpy

In [2]:
```
# 1.random.rand() : it generates random numbers in a given range
np.random.rand(2,5)
```

Out[2]:
```
array([[0.82700572, 0.44826155, 0.26111095, 0.72385351, 0.79173838],
       [0.63538867, 0.14875187, 0.99083436, 0.7673074 , 0.16745592]])
```

In [16]:
```
# 2. random.randn() : it generates random numbers from standard normal distribut
np.random.randn(2,5)
```

Out[16]:
```
array([[ 0.64622643,  2.22479829, -1.37560442,  1.88388503, -2.13967431],
       [ 1.2401821 ,  1.1764323 ,  0.56880395,  0.71157939, -0.84116461]])
```

In [7]:
```
# 3. random.normal() : Generate random numbers from a normal distribution
np.random.normal(2,6,10) # you can replace normal with binomial,uniform,bimodial
```

Out[7]:
```
array([11.52203845, -0.10514766, -7.82837118,  4.18325424, -0.20382237,
        4.24485929,  6.26164856,  6.58359785, -4.52241934, -1.75939302])
```

In [15]:
```
# 4. random.choice() : it select any number randomly by choice from given list
x=[1,2,3,4,5,6,7,8,9,10,11,12]
np.random.choice(x)
```

Out[15]:
```
np.int64(3)
```

In [18]:
```
# 5. random.randint() : it generates interger values in a given range
np.random.randint(2,6,10)
```

Out[18]:
```
array([5, 2, 5, 5, 5, 4, 3, 4, 3, 4], dtype=int32)
```

THE END . . . !