In [3]:

```
#Crea una funció que donat un Array d’una dimensió, et faci un resum estadístic bàsic de les dades. Si detecta que
#l’array té més d’una dimensió, ha de mostrar un missatge d’error.

import numpy as np
import pandas as pd

def listaAleatorios(n):
    import random
    lista = [0]  * n
    for i in range(n):
          lista[i] = random.randint(0, 10)
    return lista

print("Ingrese cuantos numeros aleatorios desea obtener")
n=int(input())

aleatorios=listaAleatorios(n)
print(aleatorios)
```

```
Ingrese cuantos numeros aleatorios desea obtener
10
[0, 3, 10, 1, 9, 8, 1, 6, 8, 7]
```

In [7]:

```
def func():
    arr = np.array(aleatorios)
    print(arr)

func()

arr = np.array(aleatorios)
print(arr)
```

```
[ 0  3 10  1  9  8  1  6  8  7]
[ 0  3 10  1  9  8  1  6  8  7]
```

In [9]:

```
if arr.ndim>1:
    print("No pot tenir més d'una dimensió")
```

In [10]:

```
dimArray= (arr.ndim)

print("El teu array té "+ str(dimArray)+ " dimensió")
```

```
El teu array té 1 dimensió
```

In [11]:

```
#Crides de diferents maneres de l'array analitzant la informació que ens podria interessar.
print(arr[2] + arr[3])
print(arr[1:5])
print(arr[4:])
print(arr[-3:-1])
print(arr[1:5:2])

print(arr.dtype)
```

```
11
[ 3 10  1  9]
[9 8 1 6 8 7]
[6 8]
[3 1]
int32
```

In [12]:

```
#Copy, no canvia l'original
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
[42  3 10  1  9  8  1  6  8  7]
[ 0  3 10  1  9  8  1  6  8  7]
```

In [13]:

```
#view, canvia l'original
y = arr.view()
arr[1] = 33

print(arr)
print(x)


print(x.base)
print(y.base)
```

```
[42 33 10  1  9  8  1  6  8  7]
[ 0  3 10  1  9  8  1  6  8  7]
None
[42 33 10  1  9  8  1  6  8  7]
```

In [15]:

```
aa = np.mean(arr)
print(aa)

bb=np.median(arr)
print(bb)

cc = np.std(arr)
print(cc)

dd= np.min(arr)
print(dd)

ee = np.max(arr)
print(ee)

ff= np.count_nonzero(arr)
print(ff)

gg= np.abs(arr)
print(gg)

hh= np.sin(arr)
print(hh)

ii= np.cos(arr)
print(ii)

jj= np.tan(arr)
print(jj)

kk = np.sin(np.pi/2)
print(kk)

ll= np.deg2rad(arr)
print(ll)

nn = np.hypot(arr[0], arr[1])
print(nn)

oo = np.cosh(arr)
print(oo)

pp  = np.unique(arr)
print(pp)

qq= np.lcm(arr[4], arr[3])
print(qq)

rr = np.gcd(arr[0], arr[6])
print(rr)

newarr = np.diff(arr)

print(newarr)
```

```
12.5
8.0
12.986531484580476
1
42
10
[42 33 10  1  9  8  1  6  8  7]
[-0.91652155  0.99991186 -0.54402111  0.84147098  0.41211849  0.98935825
  0.84147098 -0.2794155   0.98935825  0.6569866 ]
[-0.39998531 -0.01327675 -0.83907153  0.54030231 -0.91113026 -0.14550003
  0.54030231  0.96017029 -0.14550003  0.75390225]
[  2.29138799 -75.3130148    0.64836083   1.55740772  -0.45231566
  -6.79971146   1.55740772  -0.29100619  -6.79971146   0.87144798]
1.0
[0.73303829 0.57595865 0.17453293 0.01745329 0.15707963 0.13962634
 0.01745329 0.10471976 0.13962634 0.12217305]
53.41348144429457
[8.69637471e+17 1.07321790e+14 1.10132329e+04 1.54308063e+00
 4.05154203e+03 1.49047916e+03 1.54308063e+00 2.01715636e+02
 1.49047916e+03 5.48317035e+02]
[ 1  6  7  8  9 10 33 42]
9
1
[ -9 -23  -9   8  -1  -7   5   2  -1]
```

In [16]:

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

```
b'42'
b'33'
b'10'
b'1'
b'9'
b'8'
b'1'
b'6'
b'8'
b'7'
```

In [17]:

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

```
(0,) 42
(1,) 33
(2,) 10
(3,) 1
(4,) 9
(5,) 8
(6,) 1
(7,) 6
(8,) 8
(9,) 7
```

In [18]:

```
for x in np.nditer(arr):
    print(x)
```

```
42
33
10
1
9
8
1
6
8
7
```

In [19]:

```
print(np.sort(arr))

a = np.where(arr == 4)
b = np.where(arr%2 == 0)
c = np.where(arr%2 == 1)
d = np.searchsorted(arr, 7)
e = np.searchsorted(arr, 1, side='right')
f= x = np.searchsorted(arr, [3, 5, 8])
g= np.searchsorted(arr, 10)


print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
[ 1  1  6  7  8  8  9 10 33 42]
(array([], dtype=int64),)
(array([0, 2, 5, 7, 8], dtype=int64),)
(array([1, 3, 4, 6, 9], dtype=int64),)
0
0
[0 0 0]
10
```

In [20]:

```
np.sort(arr)
h = [True, False, True, False, False, True, True, False, True,False]

newArr = arr[h]

print(newArr)
```

```
[42 10  8  1  8]
```

In [21]:

```
newList= []

for i in arr:
    if i > 5:
        newList.append(True)
    else:
        newList.append(False)

newArray = arr[newList]
print(newArray)
print(newList)
```

```
[42 33 10  9  8  6  8  7]
[True, True, True, False, True, True, False, True, True, True]
```

In [22]:

```
filtering = []

for x in arr:
    if x % 2 == 0:
        filtering.append(True)
    else:
        filtering.append(False)

newFilter = arr[filtering]
print(newFilter)
print(filtering)
```

```
[42 10  8  6  8]
[True, False, True, False, False, True, False, True, True, False]
```

In [23]:

```
#Crea un quadrat de XperX amb nums aleatoris de 0 a 100

from numpy import random
import pandas as pd

quadrat = random.randint(100, size=(5, 5))

print(quadrat)
df = pd.DataFrame(quadrat)
print(df)
```

```
[[54 79 13 54  2]
 [67 28 11 58 18]
 [87 13 92  7  7]
 [77  0 32 21 36]
 [66 80 32 69 27]]
    0   1   2   3   4
0  54  79  13  54   2
1  67  28  11  58  18
2  87  13  92   7   7
3  77   0  32  21  36
4  66  80  32  69  27
```

In [24]:

```
#Crea una funció que donada una taula de dues dimensions, et calculi els totals per fila i els totals per columna.

from numpy import random

nouArr = random.randint(100, size=(2, 5))

print(nouArr)
```

```
[[50 30 96 29 30]
 [68 41  7 53 39]]
```

In [25]:

```
dff= pd.DataFrame(nouArr)
print(dff)

#Columnes
Total = dff[0].sum()
print ("Column 1 sum:",Total)

total2 = dff[1].sum()
print("Colum 2 sum:", total2)

total3 = dff[2].sum()
print("Colum 3 sum:", total3)

total4 = dff[3].sum()
print("Colum 4 sum:", total4)

total5= dff[4].sum()
print("Colum 5 sum:", total5)

#Files

totalFila = np.sum(nouArr, axis=1)
print(totalFila)
```

```
    0   1   2   3   4
0  50  30  96  29  30
1  68  41   7  53  39
Column 1 sum: 118
Colum 2 sum: 71
Colum 3 sum: 103
Colum 4 sum: 82
Colum 5 sum: 69
[235 208]
```

In [26]:

```
#Implementa manualment una funció que calculi el coeficient de correlació.
#Informa’t-en sobre els seus usos i interpretació.

#El coeficiente de correlación de Pearson mide la asociación lineal entre variables. Su valor se puede interpretar así:

#+1 – Correlación positiva completa
#+0,8 – Fuerte correlación positiva
#+0,6 – Correlación positiva moderada
#0 – sin correlación alguna
#-0,6 – Correlación negativa moderada
#-0,8 – Fuerte correlación negativa
#-1 – Correlación negativa completa

import matplotlib as plt

def myfunc():


    my_rho = np.corrcoef(nouArr)

    print(my_rho)

    seed = 13
    rand = np.random.RandomState(seed)

    x = rand.uniform(0,1,100)
    x = np.vstack((x,x*2+1))
    x = np.vstack((x,-x[0,]*2+1))
    x = np.vstack((x,rand.normal(1,3,100)))

    rho = np.corrcoef(x)
    print(rho)

#L'entrada per aquesta funció sol ser una matriu d'un tamany mxn on:
#Cada columna representa els valors d'una variable aleatoria
#Cada fila representa una sola mostra de n variables aleatories
#n representa el número total de diferents variables aleatories
#m representa el número total de mostres per cada variable

#Totes les entrades diagonals són 1 perquè perque el coeficient de correlació d'una variable amb ella mateixa sempre
#donarà aquest valor. Les diagonals solen ser iguals entre elles.

#Primer tenim una correlació completament positiva entre dues variables (+1)
#Després l'altre coeficient de correlació hauria de ser proper a 0.

#El primer rand.uniform() La trucada genera una distribucio uniforme aleatoria.
# I vstack() apila verticalment altres matrius en ell. Així podrem accedir seqüencialment.

#el segon te una relació positiva completa amb el primer, el tercer té una correlació negativa
#completa amb el primer i el quart és completament aleatori. Hauria de tenir una correlació de ~ 0.
```

```
[[ 1.         -0.68126337]
 [-0.68126337  1.        ]]
[[ 1.          1.         -1.         -0.01529094]
 [ 1.          1.         -1.         -0.01529094]
 [-1.         -1.          1.          0.01529094]
 [-0.01529094 -0.01529094  0.01529094  1.        ]]
```