Exercise 1: Define a function that summarises basic statistics data

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

In [2]:
```
def my_function(k):

 if k.ndim > 1:
  print("Atention! This array has more than one dimension and can't be computed in this function.")
 else:

   print("Length:",len(k))
   print("Maxim:",max(k))
   print("Minimum:",min(k))
   print("Average:",np.mean(k))
   print("Median:",np.median(k))
   print("Standard deviation:", np.std(k),"\n")
```

Testing the function with Case A and Case B

In [3]:
```
dim_1 = np.array([1,2,3,4,12,15])

print("Basic statistics summary Case A:\n")
print("Dimensions:", dim_1.ndim)
my_function(dim_1)
```

```
Basic statistics summary Case A:

Dimensions: 1
Length: 6
Maxim: 15
Minimum: 1
Average: 6.166666666666667
Median: 3.5
Standard deviation: 5.3359368645273735
```

In [4]:
```
dim_2 = np.array([(1.5,2,3),(4,5,6)])
print("Basic statistics summary Case B:")
print("Dimensions:", dim_2.ndim,"\n")
my_function(dim_2)
```

```
Basic statistics summary Case B:
Dimensions: 2

Atention! This array has more than one dimension and can't be computed in this function.
```

Exercice 2: Define a function that generates a square of radom numbers from 0 to 100. The user can define the number of rows and columns

In [5]:
```
def random_square(x):

 r=np.random.randint(0,101,size= (x,x))
 print(r)
```

Testing the function

In [6]:
```
random_square(5)
```

```
[[85 27 76 47 60]
 [47 26 33 83  8]
 [ 9 29 36 24 68]
 [70 66 48 38 98]
 [87 76 39 38 91]]
```

Exercice 3: Define a function that adds totals for raw and column in 2 dimensional array

In [8]:
```
def sumas(x):
 row_sums = x.sum(axis=1)
 column_sums= x.sum(axis=0)
 print("The sum of each row is:",row_sums)
 print("The sum of each column is:", column_sums)
```

Testing the function

In [9]:
```
a = np.array([(7,9,3),(4,5,6)])
print("This is the 2 dimensional array:\n",a,"\n")


sumas(a)
```

```
This is the 2 dimensional array:
 [[7 9 3]
 [4 5 6]]

The sum of each row is: [19 15]
The sum of each column is: [11 14  9]
```

Exercice 4: Create a function to calculate manually the correlation coefficient

In [11]:
```
def corr_coef(x,y):
    print(np.mean(np.multiply((a-np.mean(a)),(b-np.mean(b))))/(np.std(b)*np.std(a)),"\n")
```

Testing the function

In [13]:
```
a=[1,5,61,6,7]
b=[1,2,38,0,8]
```

In [14]:
```
print("The correlation coeficient generated by the defined function is:")
corr_coef(a,b)
```

```
The correlation coeficient generated by the defined function is:
0.986149780217546
```

In [15]:
```
print("Checking the result with np.coorcoef:")
print(np.corrcoef(a,b))
```

```
Checking the result with np.coorcoef:
[[1.         0.98614978]
 [0.98614978 1.        ]]
```

In [ ]:
```

```