Topics

- python Basics
- Numpy Introduction

In [2]:
```
# Sum of all Mid-values in the given list of integers
n = int(input("enter how many numbers: "))
res = []
for i in range(n):
    p = input("enter a number: ")
    l = len(p)
    mid = l//2
    ans = p[mid]
    print("mid value: ",ans)
    res.append(int(ans))
print("sum of mid values: ",sum(res))
```

```
enter how many numbers: 3
enter a number: 123
mid value:  2
enter a number: 456
mid value:  5
enter a number: 789
mid value:  8
sum of mid values:  15
```

Data Analysis:

- It is a meaningful information said to be Data.
- Types of data in realtime/devices
  - .txt,.doc,.ipnyb,.xls,.ppt,.mp3,.mp4 etc;
- Types of Data in realtime
  1. Quantitative
     - represnets the size/volume of data.
     - Discrete and Continous
       - This type of data is Fixed/Constant
         - Ex: numbers on Dies: varies from 1 to 6
           - Distance,Color,Nationality,Size etc...
       - Continous Data means the value/data is Continoues
         - Height,weight & hours
  2. Qualitative
     - Data is observed and placed in terms of categories
       - Ex: Data sunmission in Feedback form
- Data in Stastics:
  - Nominal,Ordinak,Interval and Ratio
    - Nominal means data is categorized on the basics of names
      - Ex: nationality,Color,Gender
    - Ordinal means data is based upon the order of values

    - Ex: feedback Form
  - Interval means a range of data values.
    - Time interval in clock : 2:15 is between 2&3
  - Ratio means data is meaningfully added,subracted,multiplicated and divided (ratios)..

Importing modules

- Numpy
- Pandas
- Matplotlib and Seaborn

In [2]:
```
import numpy as np
dir(np) # directory of numpy
```

Out[2]:
```
['ALLOW_THREADS',
 'AxisError',
 'BUFSIZE',
 'Bytes0',
 'CLIP',
 'ComplexWarning',
 'DataSource',
 'Datetime64',
 'ERR_CALL',
 'ERR_DEFAULT',
 'ERR_IGNORE',
 'ERR_LOG',
 'ERR_PRINT',
 'ERR_RAISE',
 'ERR_WARN',
 'FLOATING_POINT_SUPPORT',
 'FPE_DIVIDEBYZERO',
 'FPE_INVALID',
 'FPE_OVERFLOW',
```

In [7]:
```
np.__version__ # to get the version of numpy
```

Out[7]:
```
'1.20.1'
```

Numpy

- Numpy stands for Numerical Python
- which is used to deal with array type of data,Scitific computation

Array Creation()

- array is the sub-module in numpy module
- Syntax:
  - np.array(iterables)

In [12]:
```
a = np.array([[1,2,3],[4,5,6],[7,8,9]]) # array allows only homogenous data-struc
print("Matrix with 3x3 order: \n",a)
```

```
Matrix with 3x3 order:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [15]:
```
# tuple into array
t = np.array((1,2,3))
print("matrix: ",t)
```

```
matrix:  [1 2 3]
```

In [21]:
```
# declaration
ar = np.array([[1.2,3.4],[5.6,7.8],[4.5,6.9]],dtype = "int")
ar1 = np.array([[1.2,3.4],[5.6,7.8],[4.5,6.9]],dtype = "float")
ar2 = np.array([[1.2,3.4],[5.6,7.8],[4.5,6.9]],dtype = 'str')
print("matrix of integers: \n",ar)
print("matrix of float values: \n",ar1)
print("matrix of string values: \n",ar2)
```

```
matrix of integers:
 [[1 3]
 [5 7]
 [4 6]]
matrix of float values:
 [[1.2 3.4]
 [5.6 7.8]
 [4.5 6.9]]
matrix of string values:
 [['1.2' '3.4']
 ['5.6' '7.8']
 ['4.5' '6.9']]
```

In [25]:
```
print("result: ",np.array(res, dtype = 'str'))
```

```
result:  ['2' '5' '8']
```

In [29]:
```
# array
r = np.array(range(20))
r1 = np.array(range(0,20))
r2 = np.array(range(0,20,2))
print("range values: ",r)
print("range values with start and end: ",r1)
print("range values with start and end and step: ",r2)

# dimensions of array
print("dimension of r1: ",r1.ndim)
```

```
range values:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
range values with start and end:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
15 16 17 18 19]
range values with start and end and step:  [ 0  2  4  6  8 10 12 14 16 18]
dimension of r1:  1
```

In [35]:
```
# Multi dimension array
s = np.array([[1.2,3.4],[5.6,7.8],[4.5,6.9]],dtype = "int")
print("Matrix: \n",s)
print("dimension of array: ",s.ndim,"\n")

# size attribute represents the totaL NO.OF ELEMNETS
print("size of matrix: ",s.size)
print("shape of the mtrix: ",s.shape)
print("data_type of the matrix: ",s.dtype)
print("size of each item: ",s.itemsize) # size of  each item
```

```
Matrix:
 [[1 3]
 [5 7]
 [4 6]]
dimension of array:  2

size of matrix:  6
shape of the mtrix:  (3, 2)
data_type of the matrix:  int32
size of each item:  4
```

In [38]:
```
# 32 dimensional array using numpy
dd = np.array([1,2,3,4],ndmin = 32)
print("32 dimensional array: ",dd)

# dd = np.array([1,2,3,4],ndmin = 33)
```

```
32 dimensional array:  [[[[[[[[[[[[[[[[[[[[[[[[[[[[[1 2 3
4]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
```

In [45]:
```
# Creating another arrays

i = np.eye(5) # identity matrix
print("identity matrix: \n",i)

z = np.zeros((3,4))
print("\n zero matrix: \n",z)
```

```
identity matrix:
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]

 zero matrix:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [52]:
```
# matrix addition,subraction,multipication

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
b = np.array([[11,12,13],[14,15,16],[17,18,19]])
print("Matrix A: \n\n",a)
print("Matrix B: \n\n",b)
print("\nMatrix addition: \n\n",a + b)
print("\nMatrix subraction: \n\n",a - b)
print("\nMatrix multiplication: \n\n",a * b)
print("\nMatrix division: \n\n",a / b)
```

```
Matrix A:

 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix B:

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

Matrix addition:

 [[12 14 16]
 [18 20 22]
 [24 26 28]]

Matrix subraction:

 [[-10 -10 -10]
 [-10 -10 -10]
 [-10 -10 -10]]

Matrix multiplication:

 [[ 11  24  39]
 [ 56  75  96]
 [119 144 171]]

Matrix division:

 [[0.09090909 0.16666667 0.23076923]
 [0.28571429 0.33333333 0.375     ]
 [0.41176471 0.44444444 0.47368421]]
```

In [56]:
```
# Ones matrix
tr = np.ones((3,4))
print("One's matrix: \n",tr)
```

```
One's matrix:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [70]:
```
# matrix size, and element
f = np.full((3,3),'mounav') # 1st parameter is matrix order and 2nd parameter is
print("mtrix with all elemnts 2: \n",f)
```

```
mtrix with all elemnts 2:
 [['mounav' 'mounav' 'mounav']
 ['mounav' 'mounav' 'mounav']
 ['mounav' 'mounav' 'mounav']]
```

In [71]:
```
# fill function
f.fill(5) # modifing the existing matrix
print("modified matrix: \n",f)
```

```
modified matrix:
 [['5' '5' '5']
 ['5' '5' '5']
 ['5' '5' '5']]
```

In [73]:
```
# using linespace ===> linear space

ln = np.linspace(10,200,13)
print(ln)
len(ln)
```

```
[ 10.          25.83333333  41.66666667  57.5         73.33333333
  89.16666667 105.         120.83333333 136.66666667 152.5
 168.33333333 184.16666667 200.        ]
```

Out[73]:
```
13
```

Creating array with arange()

In [91]:
```
ar = np.arange(25)
print("array_1: \n",ar)
ar1 = np.arange(20,100,2)
print("array_2: \n",ar1) # even values
print("array_3: \n",np.arange(100,45,-5)) # reverse order
```

```
array_1:
 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24]
array_2:
 [20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66
 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98]
array_3:
 [100  95  90  85  80  75  70  65  60  55  50]
```

In [97]:
```
# coverting existing array int 2d array
# using reshape()
print("array length: ",len(ar1))
y = ar1.reshape(5,8)
print("matrix: \n",y)
```

```
array length:  40
matrix:
 [[20 22 24 26 28 30 32 34]
 [36 38 40 42 44 46 48 50]
 [52 54 56 58 60 62 64 66]
 [68 70 72 74 76 78 80 82]
 [84 86 88 90 92 94 96 98]]
```

In [98]:
```
# coverting existing array int 2d array
# using reshape()
print("array length: ",len(ar1)) # it has 40 elements
r = ar1.reshape(10,4) # 10 x 4 = 40
print("matrix: \n",r)
```

```
array length:  40
matrix:
 [[20 22 24 26]
 [28 30 32 34]
 [36 38 40 42]
 [44 46 48 50]
 [52 54 56 58]
 [60 62 64 66]
 [68 70 72 74]
 [76 78 80 82]
 [84 86 88 90]
 [92 94 96 98]]
```

In [99]:
```
r2 = np.arange(100,20,-4).reshape(4,5)
print("r2 matrix: ",r2)
```

```
r2 matrix:  [[100  96  92  88  84]
 [ 80  76  72  68  64]
 [ 60  56  52  48  44]
 [ 40  36  32  28  24]]
```

random()

- used to generate random integer in a range
- random.randint()

In [4]:
```
rn = np.random.randint(10) # it will generate only single value in between 0 & 10
print("random number: ",rn)
```

```
random number:  8
```

In [5]:
```
rd = np.random.randint(100,200) # it will genetates in the range
print("randon number: \n",rd)
```

```
randon number:
 191
```

In [12]:
```
# multiple random values

tu = np.random.randint(2,20,8) # 2 is lower value, 20 is upper value , 8 is gener
print("multiple values: ",tu)
```

```
multiple values:  [ 2 11 15  7 12  5 16  5]
```

In [17]:
```
d = np.random.random((3,3)) # random() will generates the numbers b/n 0 & 1 of si
d1 = np.random.rand(3,4) # not nesscessary to pass  order in tuple
print("random matrix: \n",d)
print("\nrandom matrix_1: \n",d1)
```

```
random matrix:
 [[0.8987006  0.05975729 0.14806794]
 [0.46722078 0.38139724 0.02626758]
 [0.07885834 0.85308729 0.52441256]]

random matrix_1:
 [[0.4491428  0.5705233  0.80710538 0.24098525]
 [0.79921823 0.27636072 0.4634359  0.59671848]
 [0.10241039 0.22735069 0.57890322 0.91659665]]
```

Accessing the Array elements

- using index
- index is of 3 types
  - positive indexing: starts from left to right
  - negative indexing: starts from left to right
  - Fancy indexing: Condition based index

In [18]:
```
e = np.random.randint(50,200,25).reshape(5,-1) # yu can give any -ve numbers
print("Matrix: \n",e)
```

```
Matrix:
 [[156 171 130  86 177]
 [196  92  64 132 180]
 [166 102 118 131 107]
 [165 122 198 124 136]
 [ 84 171 150  91 194]]
```

In [3]:
```
arn = np.arange(50,100,2).reshape(-7,5)
print("arange matrix: \n",arn)
```

```
arange matrix:
 [[50 52 54 56 58]
 [60 62 64 66 68]
 [70 72 74 76 78]
 [80 82 84 86 88]
 [90 92 94 96 98]]
```

In [10]:
```
# accessing array elements
print("row3: ",arn[3])
print("\nfull matrix: \n",arn[:])
print("\nreverse order of matrix: \n",arn[::-1])
print("\nalternative rows: \n",arn[::2])
print("\nonly even numbers in a row: \n",arn[:,1:4])
```

```
row3:  [80 82 84 86 88]

full matrix:
 [[50 52 54 56 58]
 [60 62 64 66 68]
 [70 72 74 76 78]
 [80 82 84 86 88]
 [90 92 94 96 98]]

reverse order of matrix:
 [[90 92 94 96 98]
 [80 82 84 86 88]
 [70 72 74 76 78]
 [60 62 64 66 68]
 [50 52 54 56 58]]

alternative rows:
 [[50 52 54 56 58]
 [70 72 74 76 78]
 [90 92 94 96 98]]

only even numbers in a row:
 [[52 54 56]
 [62 64 66]
 [72 74 76]
 [82 84 86]
 [92 94 96]]
```

In [62]:
```
sub = arn[:,1:4]

for line in sub:
    for val in line:
        if val%2==1:
            print("odd number: ",val)
```

In [29]:
```
arn[::2,::2] # alternating rows and columns
```

Out[29]:
```
array([[50, 54, 58],
       [70, 74, 78],
       [90, 94, 98]])
```

In [32]:
```
# transpose of a existing matrix
arn.transpose
print("transpose matrix: \n",arn)
```

```
transpose matrix:
 [[50 52 54 56 58]
 [60 62 64 66 68]
 [70 72 74 76 78]
 [80 82 84 86 88]
 [90 92 94 96 98]]
```

In [36]:
```
# fancy indexing

w = arn<100
w1 = arn[arn<100]
print("matrix_1: \n",w)
print("matrix_2: \n",w1)
```

```
matrix_1:
 [[ True  True  True  True  True]
 [ True  True  True  True  True]
 [ True  True  True  True  True]
 [ True  True  True  True  True]
 [ True  True  True  True  True]]
matrix_2:
 [50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96
 98]
```

In [45]:
```
# scientific computation on arrays

# matrix addition,subraction,multipication

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
b = np.array([[11,12,13],[14,15,16],[17,18,19]])

print("Matrix A: \n\n",a)
print("Matrix B: \n\n",b)
print("\nMatrix addition: \n\n",a + b)
print("\nMatrix subraction: \n\n",a - b)
print("\nMatrix multiplication: \n\n",a * b)
print("\nMatrix division: \n\n",a / b)
print("\nsquare of the matrix_A: \n",a**2)
print("\nsquare of matrix_B: \n",b**2)
print("\nmaximum number in Matrix_A: ",b.max())
print("minimum number in Matrix_B: ",b.min())
print("Sum of all elements in Matrix_A: ",a.sum())
```

```
Matrix A:

 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix B:

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

Matrix addition:

 [[12 14 16]
 [18 20 22]
 [24 26 28]]

Matrix subraction:

 [[-10 -10 -10]
 [-10 -10 -10]
 [-10 -10 -10]]

Matrix multiplication:

 [[ 11  24  39]
 [ 56  75  96]
 [119 144 171]]

Matrix division:

 [[0.09090909 0.16666667 0.23076923]
 [0.28571429 0.33333333 0.375     ]
 [0.41176471 0.44444444 0.47368421]]

square of the matrix_A:
 [[ 1  4  9]
 [16 25 36]
```

```
 [49 64 81]]

square of matrix_B:
 [[121 144 169]
 [196 225 256]
 [289 324 361]]

maximum number in Matrix_A:  19
minimum number in Matrix_B:  11
Sum of all elements in Matrix_A:  45
```

In [46]:
```
sum(arn)
```

Out[46]:
```
array([350, 360, 370, 380, 390])
```

In [51]:
```
# Scientific Computation
# Logarthims and Exponentials

print("log1: ",np.log(1))
print("log10: ",np.log(10))
print("log0: ",np.log(0))
print("log2: ",np.log(2))
```

```
log1:  0.0
log10:  2.302585092994046
log0:  -inf
log2:  0.6931471805599453

<ipython-input-51-39aa32be6a4b>:6: RuntimeWarning: divide by zero encountered i
n log
  print("log0: ",np.log(0))
```

In [55]:
```
print("exp1: ",np.exp(1))
print("exp0: ",np.exp(0))
print("exp2: ",np.exp(2))
print("log e to base e value: ",np.log(np.exp(1)))
```

```
exp1:  2.718281828459045
exp0:  1.0
exp2:  7.38905609893065
log value:  1.0
```

In [57]:
```
a,b = [9,3,1],[4,10,4]
new = []
for p,q in zip(a,b):
    if p>q:
        new.append(p)
    else:
        new.append(q)

print("maximum array: ",new)
```

```
maximum array:  [9, 10, 4]
```

In [58]:
```
def greater(x,y):
    mx = []
    for c,v in zip(x,y):
        if c > v:
            mx.append(c)
        else:
            mx.append(v)
    return mx
print("max array: ",greater([9,3,1],[4,10,4]))
```

```
max array:  [9, 10, 4]
```

In [59]:
```
def big(x,y):
    if x > y:
        return x
    else:
        return y
```

In [61]:
```
h = np.vectorize(big) # works on any iterables
h([9,3,1],[4,10,4])
```

Out[61]:
```
array([ 9, 10,  4])
```

In [ ]:
```

```