In [1]:

```
## Why Numpy
# Size
# Simple API
# homogeneous n-dimensional arrays
# Faster computation than lists
# size in memory
```

In [47]:

```
# Numerical Python ... NumPy
# Matrix

x = [2, 4, '5']
```

In [40]:

```
type(x[1])
```

Out[40]:

```
str
```

In [ ]:

```

```

In [50]:

```
# homogeneous vs heteroeneous n-dimensional arrays
s = [3 ,5.0 , 'df']
print(type(s))
print(type(s[0]))
print(type(s[1]))
print(type(s[2]))
```

```
<class 'list'>
<class 'int'>
<class 'float'>
<class 'str'>
```

In [52]:

```
import numpy
import sys
```

In [53]:

```
s_numpy = numpy.array([1,2,3])
print(type(s_numpy))
```

```
<class 'numpy.ndarray'>
```

In [54]:

```
numpy_arr = numpy.array([1,2,3])
py_arr = [1,2,3]
print(type(py_arr))
print(type(numpy_arr))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

In [67]:

```
import time
import numpy as np


numpy_arr = numpy.array(range(10000000))
py_arr = range(1000)

tic = time.perf_counter()
numpy_arr = numpy_arr + numpy_arr
toc = time.perf_counter()
print(f"Downloaded the tutorial in {toc - tic:0.4f} seconds")
```

```
Downloaded the tutorial in 0.0557 seconds
```

In [63]:

```
time.perf_counter()
```

Out[63]:

```
1138.800717836
```

In [72]:

```
x = [1, 2 , 3]
y = [3, 4, 5]
#Z = [x[i] + y[i] for i in range(len(x))]
Z = [x[0] + y[0] , x[1] + y[1], x[2] + y[2]]
print(Z)
print(type(Z))
# numpy
x = numpy.array([1,2,3])
y = numpy.array([3, 4, 5])
Z = x + y
print(Z)
print(type(Z))
```

```
[4, 6, 8]
<class 'list'>
[4 6 8]
<class 'numpy.ndarray'>
```

In [ ]:

```
!vim test.py
python test.py
```

In [ ]:

```
# 8 bits = 255
# 16 bit = 65000
# 32 bit = 4294967296
```

In [89]:

```
import time as t
import numpy as np

def pure_python_sum(num):
    tic = t.perf_counter()
    X = range(num)
    #Y = range(num)
    Z = sum(X)/len(X) #[X[i] + Y[i] for i in range(len(X)) ]
    toc = t.perf_counter()
    print(f"List: Downloaded the tutorial in {toc - tic:0.4f} seconds")
    print(Z)

def numpy_sum(num):
    tic = t.perf_counter()
    X = np.arange(num,dtype=np.int32)
    #Y = np.arange(num,dtype=np.int32)
    Z = np.mean(X)#X + Y
    toc = t.perf_counter()
    print(f"NumPy: Downloaded the tutorial in {toc - tic:0.4f} seconds")
    print(Z)

n = 100000000
pure_python_sum(n)
numpy_sum(n)
```

```
List: Downloaded the tutorial in 7.2950 seconds
49999999.5
NumPy: Downloaded the tutorial in 0.9895 seconds
49999999.5
```

In [106]:

```
numpy_arr = numpy.array([1,2,700000],dtype=np.int8)
py_arr = [1,2,3]
```

In [107]:

```
print(numpy_arr)
```

```
[     1      2 -20896]
```

In [96]:

```
sizeof_numpy_arr = numpy_arr.itemsize * numpy_arr.size
sizeof_py_arr = sys.getsizeof(py_arr[0]) * len(py_arr)
```

In [97]:

```
print(sizeof_numpy_arr) ### size in bytes ####
print(sizeof_py_arr)
```

```
3
84
```

In [16]:

```
print(type(py_arr))
print(type(numpy_arr))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [110]:

```
## Size example
import numpy as np
import sys
## 8 bits = 1 byte
## x = 1 8 bit ===== 00000001
## x = 1 32 bits ===== 000000000000000000000000001
py_arr = [1,2,3,4,5,6]
numpy_arr8 = np.array([1,2,3,4,5,6]) # 255
numpy_arr16 = np.array([1,2,3,4,5,6], dtype=np.int16) # 65000
numpy_arr32 = np.array([1,2,3,4,5,6], dtype=np.int32) #
numpy_arr64 = np.array([1,2,3,4,5,6], dtype=np.int64)

sizeof_py_arr = sys.getsizeof(py_arr[0]) * len(py_arr)   # Size = 168
sizeof_numpy_arr8 = numpy_arr8.itemsize * numpy_arr8.size   # Size = 6
sizeof_numpy_arr16 = numpy_arr16.itemsize * numpy_arr16.size   # Size = 12
sizeof_numpy_arr32 = numpy_arr32.itemsize * numpy_arr32.size   # Size = 24
sizeof_numpy_arr64 = numpy_arr64.itemsize * numpy_arr64.size   # Size = 48
```

In [112]:

```
print(sizeof_py_arr)  # Size = 144
print(sizeof_numpy_arr8)   # Size = 6
print(sizeof_numpy_arr16)   # Size = 12
print(sizeof_numpy_arr32)   # Size = 24
print(sizeof_numpy_arr64)
```

```
  File "<ipython-input-112-ca4d4fc86dc2>", line 1
    print sizeof_py_arr # Size = 144
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(size
of_py_arr # Size = 144)?
```

In [127]:

```
py_arr = [1,2,3,4,5,6]
numpy_arr = np.array([1,5, 4,3], dtype=np.int8)
#[1, 3 ,4 ,5]
```

In [128]:

```
print(np.mean(numpy_arr)) ## 1 + 2 + 3 + 4 + 5 +6 / 6 average
print(np.median(numpy_arr)) ##  3 + 4 / 2 ...
```

```
3.25
3.5
```

In [126]:

```
np.max(numpy_arr)
```

Out[126]:

```
30
```

In [169]:

```
np.random.randint(10,20)
```

Out[169]:

```
16
```

In [ ]:

```
### random  === 0 ___ 1  ,,,, 0.5 *  500 = 250
```

In [223]:

```
X = np.random.random_sample(2) * 500
X
```

Out[223]:

```
array([224.19219226, 252.83794424])
```

In [221]:

```
np.mean(X)
```

Out[221]:

```
249.41554838145075
```

In [112]:

```
np.random.rand(6,2) * 10
```

Out[112]:

```
array([[5.54912015, 4.34697779],
       [7.45014642, 5.11993859],
       [0.17286041, 0.38599458],
       [0.49992011, 4.72657735],
       [7.68493213, 6.78701189],
       [8.64081273, 4.0601094 ]])
```

In [227]:

```
# Simple API
numpy_arr32 = np.array([1,2,3,5,6,4,5,66,77,500])
print(np.mean(numpy_arr32))
print(np.median(numpy_arr32))
print(np.std(numpy_arr32))
print(np.var(numpy_arr32))
print(np.percentile(numpy_arr32,75))
```

```
66.9
5.0
146.84852740153713
21564.49
51.0
```

In [228]:

```
x = numpy_arr32
```

In [229]:

```
print(np.mean(x))
print(np.median(x))
print(np.std(x))
print(np.var(x))
print(np.percentile(x,25)) # what is the value that 25% of data are below.
```

```
66.9
5.0
146.84852740153713
21564.49
3.25
```

In [230]:

```
print(np.sum(numpy_arr32))
print(np.min(numpy_arr32))
print(np.max(numpy_arr32))
```

```
669
1
500
```

In [231]:

```
print(numpy_arr32)
print(np.sqrt(numpy_arr32))
```

```
[  1   2   3   5   6   4   5  66  77 500]
[ 1.          1.41421356  1.73205081  2.23606798  2.44948974  2.
  2.23606798  8.1240384   8.77496439 22.36067977]
```

In [ ]:

```

```

In [244]:

```
np_arr = np.array ( [ [1, 2, 3, 4, 5],
                         [6, 7, 8, 9, 10], [2,3,4,5,6]] )
```

In [ ]:

```
# age , weight , height
# [20, 50, 150]
# [15, 60, 170]
# [30, 90, 180]
# grades
# [70, 75, 60]
# [40, 60, 70]
# [33, 55, 90]
```

In [263]:

```
students = np.array( [[70, 75, 60, 80], [40, 60, 70, 90], [33, 90, 55, 90]])
```

In [266]:

```
students.shape
```

Out[266]:

```
(3, 4)
```

In [272]:

```
students.reshape(4,5)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-272-748943de9659> in <module>
----> 1 students.reshape(4,5)

ValueError: cannot reshape array of size 12 into shape (4,5)
```

In [ ]:

```

```

In [264]:

```
np.median(students,0)
```

Out[264]:

```
array([40., 75., 60., 90.])
```

In [265]:

```
students
```

Out[265]:

```
array([[70, 75, 60, 80],
       [40, 60, 70, 90],
       [33, 90, 55, 90]])
```

In [258]:

```
np.transpose(students)
```

Out[258]:

```
array([[70, 40, 33],
       [75, 60, 55],
       [60, 70, 90]])
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [249]:

```
np.mean(np_arr,0)
```

Out[249]:

```
array([3., 4., 5., 6., 7.])
```

In [ ]:

```

```

In [273]:

```
# matrix manupulation
# A 2 * 5 matrix
np_md_arr = np.array ( [ [1, 2, 3, 4, 5],
                         [6, 7, 8, 9, 10]] )

# Creates a 5 * 2 Matrix
# [ [1,2], [3,4], [5,6], [7,8], [9,10]]
np_modmd_arr = np_md_arr.reshape(1,2)
```

In [274]:

```
print(np_md_arr)
print(np_modmd_arr)
```

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

In [82]:

```


# A 1 * 4 Matrix
np_md_arr2 = np.array ( [1, 2, 3, 4] )

# Creates a 2 * 2 Matrix
#[ [1,2], [3,4] ]
np_modmd_arr2 = np_md_arr2.reshape(2,2)
# n_arr.shape for finding the shape of an array
print(np_md_arr2)
print(np_modmd_arr2)
```

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

In [83]:

```
n_arr = np.array([[1,2],[3,0]])
print(n_arr.shape)
```

```
(2, 2)
```

In [84]:

```

np_nd_arr = np.arange(1,5,0.5)
print(np_nd_arr)
print(np_nd_arr.reshape(2,4))
# Generates => [0 1 2 3 4 5 6 7 8 9]
# np_nd_arr = np.arange(0,10,2) ...  or float instead
# np_nd_arr = np.arange(0,10).reshape(5,2)
# reshape back to vector or array
# np_arr.ravel()
```

```
[1.  1.5 2.  2.5 3.  3.5 4.  4.5]
[[1.  1.5 2.  2.5]
 [3.  3.5 4.  4.5]]
```

In [85]:

```
np_nd_arr
```

Out[85]:

```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

In [ ]:

```
##### slicing and indexing
```

In [276]:

```
s = np.array ( [ [1, 2, 3, 4, 5],
                         [6, 7, 8, 9, 10]] )
```

In [279]:

```
s
```

Out[279]:

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [293]:

```
s_row1 = s[:,2:5]
print(s_row1)
```

```
[[ 3  4  5]
 [ 8  9 10]]
```

In [295]:

```
np.ones(10)
```

Out[295]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [86]:

```
# more in generating data
# all zeros or ones
np_array_zeros = np.zeros(10)
np_array_ones = np.ones(10)
```

In [87]:

```
np_array_zeros = np.ones(10)
print(np_array_zeros)
```

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

In [88]:

```
# multi-dimensision math
# for example in python Lists
py_arr = [1,2,3] * 2 #### pure python list
print(py_arr)
# numpy array
numpy_arr = np.array([1,2,3], dtype=np.int32) + 2
print(numpy_arr)
## also for more dimensions can be used
```

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

In [314]:

```
type([1,2,3])
```

Out[314]:

```
list
```

In [318]:

```
py_arr = [1,2,3] + [1,2,3] #### pure python list
print(py_arr)
```

```
[1, 2, 3, 1, 2, 3]
```

In [323]:

```
numpy_arr = np.array([1,2,3]) / np.array([1,2,3])
```

In [324]:

```
numpy_arr
```

Out[324]:

```
array([1., 1., 1.])
```

In [ ]:

```

```

In [ ]:

```

```

In [325]:

```
x = np.array([2 , 4 , 10]) * np.array([5 , 12 , 33])
print(x)
```

```
[ 10  48 330]
```

In [348]:

```
students = np.array( [[70, 75, 60, 80], [40, 60, 70, 90], [33, 90, 55, 90]])
```

In [349]:

```
students
```

Out[349]:

```
array([[70, 75, 60, 80],
       [40, 60, 70, 90],
       [33, 90, 55, 90]])
```

In [380]:

```
np.mean(students,1)
```

Out[380]:

```
array([71.25, 65.  , 67.  ])
```

In [382]:

```
index = np.where(np.mean(students,1)<66)
```

In [390]:

```
index
```

Out[390]:

```
(array([1]),)
```

In [371]:

```
students[index]
```

Out[371]:

```
array([[70, 75, 60, 80]])
```

In [ ]:

```

```

In [ ]:

```
students = np.array( [[70, 75, 60, 80], [40, 60, 70, 90], [33, 90, 55, 90]])
```

In [91]:

```
np_arr = np.array([5,2,1,4,0,5,6,7,9,0,12,234,45,56,6])
ind = np.where(np_arr > 10)
print(ind)
## use index to get elements
#print np_arr[[0,4]]
print(np_arr[ind])
```

```
(array([10, 11, 12, 13]),)
[ 12 234  45  56]
```

In [376]:

```
# count_nonzero
np_arr = np.array([5,0,1,4,0,0,0,0,0,0,0,45,0,0,0,0,56,0])
# sparse array
print(np.count_nonzero(np_arr))
# nonzero
index = np.nonzero(np_arr)
print(index)
print(np_arr[index])
```

```
5
(array([ 0,  2,  3, 11, 16]),)
[ 5  1  4 45 56]
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [93]:

```
# homogeneous n-dimensional arrays
py_arr = [1,3,4,7,16]
numpy_arr = np.array([1,2,3,4,5,6])
```

In [94]:

```
py_arr = [1,3,4,7,'hello']
```

In [418]:

```
filename = './test.txt'
dataset = np.loadtxt(filename)
```

In [421]:

```
dataset
```

Out[421]:

```
array([[70., 75., 60., 80.],
       [40., 60., 70., 90.],
       [33., 90., 55., 90.]])
```

In [414]:

```
dataset.reshape(4,3)
```

Out[414]:

```
array([[70., 75., 60.],
       [80., 40., 60.],
       [70., 90., 33.],
       [90., 55., 90.]])
```

In [ ]:

```

```

In [ ]:

```

```

In [95]:

```
filename = './test.txt'
dataset = np.loadtxt(filename)
print(dataset.shape)
print(dataset)
```

```
(2, 3)
[[10.  4.  5.]
 [ 2.  3.  4.]]
```

In [103]:

```
filename = './test_comma.txt'
dataset = np.loadtxt(filename, delimiter = ",")
print(dataset.shape)
print(dataset)
```

```
(2, 4)
[[1. 4. 6. 6.]
 [7. 7. 7. 9.]]
```

In [104]:

```
np.transpose(dataset)
```

Out[104]:

```
array([[1., 7.],
       [4., 7.],
       [6., 7.],
       [6., 9.]])
```

In [115]:

```
np.savetxt('tmp.txt', dataset, delimiter = "," )##, fmt='%1.4e') ## fmt='%1.2f'
```

In [106]:

```
type(dataset)
```

Out[106]:

```
numpy.ndarray
```

In [ ]:

```

```