Numpy Library

- Numerical python Exitesion
- Numpy array is a powerful N-dimensional array object which is in the form of rows and columns.

In [44]:

```
# Single Dimensional Array

import numpy as np
a=np.array([1,2,3])
li = list(a)
print(len(li))
print(len(a))
```

```
3
3
```

In [2]:

```
# Multi-Dimensional Array
# 2-D array

a=np.array([(1,2,3),(4,5,6)])
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

Python NumPy Array v/s List

- We use python numpy array instead of a list because of the below three reasons:
    - Less Memory
    - Fast
    - Convenient

In [43]:

```
import numpy as np

import time
import sys
S= range(1000)
#print(list(S))
print(sys.getsizeof(S)*len(S))

D= np.arange(1000)
#print(list(D))
print(D.size*D.itemsize)
```

```
48000
4000
```

In [58]:

```
# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2,2,3)

print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)
```

```

Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]
```

In [20]:

```
# Python program to demonstrate
# basic operations on single array
import numpy as np

a = np.array([1, 2, 5, 3])

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)

# square each element
print ("Squaring each element:", a**2)

# modify existing array
a *= 2
print ("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

In [35]:

```
x2 = np.random.randint(10, size=(3, 4))
x2
```

Out[35]:

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

In [42]:

```
x1 = np.random.randint(100,size=(4,5))
print(x1)

print(x2)
```

```
[[62 91 18 61 42]
 [69 26 24 46 63]
 [88 53 12 47 57]
 [60 80 44 67  8]]
[[6 8 5 4]
 [7 1 3 0]
 [0 1 3 2]]
```

In [83]:

```
five = np.zeros((2,4))
one = np.ones((3,3))
one
```

Out[83]:

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

In [86]:

```
li = range(10)
list(li)
```

Out[86]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [91]:

```
np_li = np.arange(1,5,0.1)
np_li
```

Out[91]:

```
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2,
       2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5,
       3.6, 3.7, 3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8,
       4.9])
```

In [136]:

```
m3 = np.random.randint(0,10,size=(3,3,3,3))
m3.shape # prints shape of the matrix ex:(3,3,3,3)
m3.dtype # prints data type
m3.size  # Elements size
m3.ndim  # n Dimensional
m3.itemsize # Item size
m3.nbytes   # array size in bytes
```

Out[136]:

```
324
```

In [130]:

```
va = m3[0][0][0][1]
print(va)
```

```
1
```

Pandas

Use Cases

- Data processing
- Data Transformation
- Data Analysis

Notations

- Series
- DataFrames

In [140]:

```
import pandas as pd
internal_1 = {'s1':22,'s2':45,'s3':78}
internal_2 = {'s1':35,'s2':33,'s3':67}
print(pd.Series(internal_2))
```

```
s1    35
s2    33
s3    67
dtype: int64
```

In [145]:

```
heading = {'Internal_1':internal_1,'Internal_2':internal_2}
print(pd.Series(heading))
df = pd.DataFrame(heading)
df
```

```
Internal_1    {'s1': 22, 's2': 45, 's3': 78}
Internal_2    {'s1': 35, 's2': 33, 's3': 67}
dtype: object
```

Out[145]:

	Internal_1	Internal_2
s1	22	35
s2	45	33
s3	78	67

In [153]:

```
df.columns
```

Out[153]:

```
Index(['Internal_1', 'Internal_2'], dtype='object')
```

In [172]:

```
df.values[0]
avg = {}
# avg = {'s1':27.5,'s2':39,'s3':70}
s1=(df.values[0][0]+df.values[0][1])/2
s1
avg['s1']=s1
avg
s2=(df.values[1][0]+df.values[1][1])/2
avg['s2']=s2
avg
heading['Average']=avg
heading
```

Out[172]:

```
{'Internal_1': {'s1': 22, 's2': 45, 's3': 78},
 'Internal_2': {'s1': 35, 's2': 33, 's3': 67},
 'Average': {'s1': 28.5, 's2': 39.0}}
```

In [175]:

```
df=pd.DataFrame(heading)
df
```

Out[175]:

	Internal_1	Internal_2	Average
s1	22	35	28.5
s2	45	33	39.0
s3	78	67	NaN

In [176]:

```
import pandas as pd
df = pd.read_csv('income.csv')
df
```

Out[176]:

	GEOID	State	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	04000US01	Alabama	37150	37952	42212	44476	39980	40933	42590	43464	41381
1	04000US02	Alaska	55891	56418	62993	63989	61604	57848	57431	63648	61137
2	04000US04	Arizona	45245	46657	47215	46914	45739	46896	48621	47044	50602
3	04000US05	Arkansas	36658	37057	40795	39586	36538	38587	41302	39018	39919
4	04000US06	California	51755	55319	55734	57014	56134	54283	53367	57020	57528

In [178]:

```
df.values[1]
```

Out[178]:

```
array(['04000US02', 'Alaska', 55891, 56418, 62993, 63989, 61604, 57848,
       57431, 63648, 61137], dtype=object)
```

In [190]:

```
alaska = list(df.values[1,2:])
years = df.columns[2:]
years
```

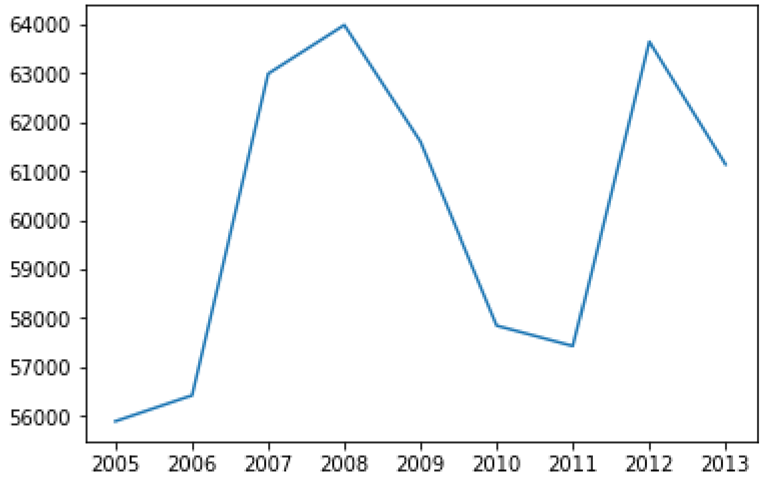
Out[190]:

```
Index(['2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '201
3'], dtype='object')
```

In [192]:

```
import matplotlib.pyplot as plt

plt.plot(years,alaska)
plt.show()
```



In [ ]:

```

```