Today's Objective

- List Comprahension
- File Data Processing
- Data Science Libraries
  - Numpy
  - Pandas
  - Matplot Lib..
- AWS Educate Account

List Comprahension

In [3]:
```
hi='Hunuman'
#H
#u
#n
#u
# .
# .
# .
```

In [12]:
```
li=[]
for i in hi:
    li.append(i)
li
```

Out[12]:
```
['H', 'u', 'n', 'u', 'm', 'a', 'n']
```

In [6]:
```
li=[i for i in hi ]


```

Out[6]:
```
['H', 'u', 'n', 'u', 'm', 'a', 'n']
```

In [7]:
```
#print the fist 10 natural numbers
for i in range(1,11):
    print(i)
```

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

In [10]:
```
naturl=[num for num in range(1,11) if num%2==0]
```

In [11]:
```
naturl
```

Out[11]:
```
[2, 4, 6, 8, 10]
```

File Data Processing

- Storing the data permanetly
- Dif. type
  - .py,.txt,.pdf,.ext,.doc,.excel,.jpg,.png,mp3.. exc
- file DATA OPARATION
  - read opartion -->mode(r)
  - write -->mode(w)
  - append(a)
  - create (x)
- open(filepath,mode)
  - with open()

In [13]:
```
fp='mydatafile.txt'
```

In [15]:
```
#File Data Reading
with open(fp,'r') as f:
    fileData=f.read()
    print(fileData)

```

```
pubg
winner winner mudapapu dinner
ludo
coinmaster
8ballpool
bubble Shooter
apssdc
cbit
vbit
ksrm
srit
gouthami
vagdevi
ysr
```

In [17]:
```
#Write Data to Files

with open(fp,'w') as f:
    fdata_write=f.write("muni")
    print(fdata_write)
```

```
4
```

In [16]:
```
type(fileData)
```

Out[16]:
```
str
```

In [19]:
```
# Append The Data to file
with open(fp,'a') as f:
    fd=f.write("\nMuneiah tellakula")
    print(fd)
```

```
18
```

In [21]:
```
with open(fp,'r') as f:
    fileData=f.read(21)
    print(fileData)
```

```
muniMuneiah tellakula
```

In [26]:
```
#Total Word Count present in the file
with open(fp,'r') as f:
    fileData=f.read()
    lines=fileData.split()
    print(len(lines))
```

```
22
```

In [28]:
```
#Total line Count present in the file
with open(fp,'r') as f:
    fileData=f.readlines()
    print(len(fileData))

```

```
16
```

In [39]:
```
#How many Char present in the file
with open(fp,'r') as f:
    x=f.read()
    s=''
    for i in x:
        s=s+i
        print(i,end=" ")
print("\nTotal Char count is :",len(s))

```

```
c b i t v b i t
Total Char count is : 8
```

In [40]:
```
#Data Science Libraries

#     Numpy

import numpy as np

np.__version__rsion__
#     Pandas
#     Matplot Lib..


```

Out[40]: '1.16.4'

In [ ]:
```
# Extenstion
# Numarical Python

```

In [43]:
```
oneD=np.array([1,2,3,4])
oneD
```

Out[43]: array([1, 2, 3, 4])

In [51]:
```
s=range(1001)
n=np.arange(1,1001)

```

In [41]:
```
print(dir(np))
```

```
_asbuffer', 'intc', 'integer', 'interp', 'intersect1d', 'intp', 'invert', 'ip
mt', 'irr', 'is_busday', 'isclose', 'iscomplex', 'iscomplexobj', 'isfinite',
'isfortran', 'isin', 'isinf', 'isnan', 'isnat', 'isneginf', 'isposinf', 'isre
al', 'isrealobj', 'isscalar', 'issctype', 'issubclass_', 'issubdtype', 'issub
sctype', 'iterable', 'ix_', 'kaiser', 'kron', 'lcm', 'ldexp', 'left_shift',
'less', 'less_equal', 'lexsort', 'lib', 'linalg', 'linspace', 'little_endia
n', 'load', 'loads', 'loadtxt', 'log', 'log10', 'log1p', 'log2', 'logaddexp',
'logaddexp2', 'logical_and', 'logical_not', 'logical_or', 'logical_xor', 'log
space', 'long', 'longcomplex', 'longdouble', 'longfloat', 'longlong', 'lookfo
r', 'ma', 'mafromtxt', 'mask_indices', 'mat', 'math', 'matmul', 'matrix', 'ma
trixlib', 'max', 'maximum', 'maximum_sctype', 'may_share_memory', 'mean', 'me
dian', 'memmap', 'meshgrid', 'mgrid', 'min', 'min_scalar_type', 'minimum', 'm
intypecode', 'mirr', 'mod', 'modf', 'moveaxis', 'msort', 'multiply', 'nan',
'nan_to_num', 'nanargmax', 'nanargmin', 'nancumprod', 'nancumsum', 'nanmax',
'nanmean', 'nanmedian', 'nanmin', 'nanpercentile', 'nanprod', 'nanquantile',
'nanstd', 'nansum', 'nanvar', 'nbytes', 'ndarray', 'ndenumerate', 'ndfromtx
t', 'ndim', 'ndindex', 'nditer', 'negative', 'nested_iters', 'newaxis', 'next
after', 'nonzero', 'not_equal', 'nper', 'npv', 'numarray', 'number', 'obj2sct
ype', 'object', 'object0', 'object_', 'ogrid', 'oldnumeric', 'ones', 'ones_li
ke', 'outer', 'packbits', 'pad', 'partition', 'percentile', 'pi', 'piecewis
```

In [57]:
```
# Normal list
import sys

s=range(1000)
print(sys.getsizeof(s)*len(s))

```

```
48000
```

In [58]:
```
#numpy list
s1=np.arange(1000)
print(s1.size*s1.itemsize)
```

```
4000
```

In [62]:
```
li=[1,2,3,4,7]
# multilpy of 2
# output:2,4,6,8,14
li+2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-62-82d69873129d> in <module>
      2 # multilpy of 2
      3 # output:2,4,6,8,14
----> 4 li+2

TypeError: can only concatenate list (not "int") to list
```

In [65]:
```
oneD
```

Out[65]:
```
array([1, 2, 3, 4])
```

In [68]:
```
oneD+2
```

Out[68]:
```
array([3, 4, 5, 6])
```

In [71]:
```
oneD
```

Out[71]:
```
array([1, 2, 3, 4])
```

In [72]:
```
oneD[3]
```

Out[72]:
```
4
```

In [73]:
```
oneD[2:]
```

Out[73]:
```
array([3, 4])
```

In [78]:
```
twoD=np.array([[10,20,30,1],[1,2,3,4]])
twoD
```

Out[78]:
```
array([[10, 20, 30,  1],
       [ 1,  2,  3,  4]])
```

In [83]:
```
oneDimen=np.random.randint(10,size=6)
oneDimen
```

Out[83]:
```
array([3, 8, 7, 6, 8, 8])
```

In [86]:
```
td=np.random.randint(10,size=(6,4))
td
```

Out[86]:
```
array([[2, 4, 9, 3],
       [4, 4, 3, 2],
       [0, 4, 3, 0],
       [3, 4, 3, 0],
       [2, 3, 9, 1],
       [5, 5, 0, 8]])
```

In [88]:
```
td[0][1]
```

Out[88]:
```
4
```

In [107]:
```
print(td[4][2:],td[5][2:])
```

```
[9 1] [0 8]
```

In [91]:
```
td[5][3]
```

Out[91]:
```
8
```

In [110]:
```
multi=np.random.randint(10,size=(3,4,4))
multi
```

Out[110]:
```
array([[[4, 5, 1, 6],
        [2, 1, 8, 1],
        [0, 6, 8, 3],
        [2, 1, 4, 9]],

       [[7, 8, 5, 9],
        [1, 9, 2, 3],
        [3, 2, 7, 7],
        [3, 4, 5, 9]],

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

In [111]:
```
1 multi[1]
```

Out[111]:
```
array([[7, 8, 5, 9],
       [1, 9, 2, 3],
       [3, 2, 7, 7],
       [3, 4, 5, 9]])
```

In [112]:
```
1 sum(multi)
```

Out[112]:
```
array([[17, 19, 10, 16],
       [ 9, 10, 13, 12],
       [ 5, 15, 17, 12],
       [13,  5, 12, 25]])
```

In [113]:
```
1 oneD
```

Out[113]:
```
array([1, 2, 3, 4])
```

In [117]:
```
1 oneD.reshape()
```

Out[117]:
```
<function ndarray.reshape>
```

In [118]:
```
1 oneD
```

Out[118]:
```
array([1, 2, 3, 4])
```

In [119]:
```
1 help(np.reshape)
```

```
Help on function reshape in module numpy:

reshape(a, newshape, order='C')
    Gives a new shape to an array without changing its data.

    Parameters
    ----------
    a : array_like
        Array to be reshaped.
    newshape : int or tuple of ints
        The new shape should be compatible with the original shape. If
        an integer, then the result will be a 1-D array of that length.
        One shape dimension can be -1. In this case, the value is
        inferred from the length of the array and remaining dimensions.
    order : {'C', 'F', 'A'}, optional
        Read the elements of `a` using this index order, and place the
        elements into the reshaped array using this index order.  'C'
        means to read / write the elements using C-like index order,
        with the last axis index changing fastest, back to the first
        axis index changing slowest. 'F' means to read / write the
        elements using Fortran-like index order, with the first index
        changing fastest, and the last index changing slowest. Note that
        the 'C' and 'F' options take no account of the memory layout of
        the underlying array, and only refer to the order of indexing.
        'A' means to read / write the elements in Fortran-like index
        order if `a` is Fortran *contiguous* in memory, C-like order
        otherwise.

    Returns
    -------
    reshaped_array : ndarray
        This will be a new view object if possible; otherwise, it will
        be a copy.  Note there is no guarantee of the *memory layout* (C- or
        Fortran- contiguous) of the returned array.

    See Also
    --------
    ndarray.reshape : Equivalent method.

    Notes
    -----
    It is not always possible to change the shape of an array without
    copying the data. If you want an error to be raised when the data is copie
d,
    you should assign the new shape to the shape attribute of the array::

     >>> a = np.zeros((10, 2))
     # A transpose makes the array non-contiguous
     >>> b = a.T
     # Taking a view makes it possible to modify the shape without modifying
     # the initial object.
     >>> c = b.view()
     >>> c.shape = (20)
     AttributeError: incompatible shape for a non-contiguous array
```

```
The `order` keyword gives the index ordering both for *fetching* the values
from `a`, and then *placing* the values into the output array.
For example, let's say you have an array:

>>> a = np.arange(6).reshape((3, 2))
>>> a
array([[0, 1],
       [2, 3],
       [4, 5]])

You can think of reshaping as first raveling the array (using the given
index order), then inserting the elements from the raveled array into the
new array using the same kind of index ordering as was used for the
raveling.

>>> np.reshape(a, (2, 3)) # C-like index ordering
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.reshape(np.ravel(a), (2, 3)) # equivalent to C ravel then C reshape
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.reshape(a, (2, 3), order='F') # Fortran-like index ordering
array([[0, 4, 3],
       [2, 1, 5]])
>>> np.reshape(np.ravel(a, order='F'), (2, 3), order='F')
array([[0, 4, 3],
       [2, 1, 5]])

Examples
--------
>>> a = np.array([[1,2,3], [4,5,6]])
>>> np.reshape(a, 6)
array([1, 2, 3, 4, 5, 6])
>>> np.reshape(a, 6, order='F')
array([1, 4, 2, 5, 3, 6])

>>> np.reshape(a, (3,-1))       # the unspecified value is inferred to be 2
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Pynthon Pandas

- Data tranformation
- Data cleaing
- Data Processing
- Data presents in two Froms
  - Series
  - Data Frams

In [120]:
```
1 import pandas as pd
2
3 fp=pd.read_csv('income.csv')
4 fp
5
```

```
C:\Users\APSSDC-23\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning: n
umpy.ufunc size changed, may indicate binary incompatibility. Expected 192 from
C header, got 216 from PyObject
  return f(*args, **kwds)
C:\Users\APSSDC-23\Anaconda3\lib\importlib\_bootstrap.py:219: RuntimeWarning: n
umpy.ufunc size changed, may indicate binary incompatibility. Expected 192 from
C header, got 216 from PyObject
  return f(*args, **kwds)
```

Out[120]:

	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 [190]:
```
1 fp.columns
```

Out[190]:
```
Index(['GEOID', 'State', '2005', '2006', '2007', '2008', '2009', '2010',
       '2011', '2012', '2013'],
      dtype='object')
```

In [191]:
```
1 years=fp.columns[2:]
2 years
```

Out[191]:
```
Index(['2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013'],
dtype='object')
```

In [193]:
```
1 fp.values[1]
```

Out[193]:
```
array(['04000US02', 'Alaska', 55891, 56418, 62993, 63989, 61604, 57848,
       57431, 63648, 61137], dtype=object)
```

In [195]:
```
1 alaska=list(fp.values[1,2:])
2 alaska
```

Out[195]:
```
[55891, 56418, 62993, 63989, 61604, 57848, 57431, 63648, 61137]
```

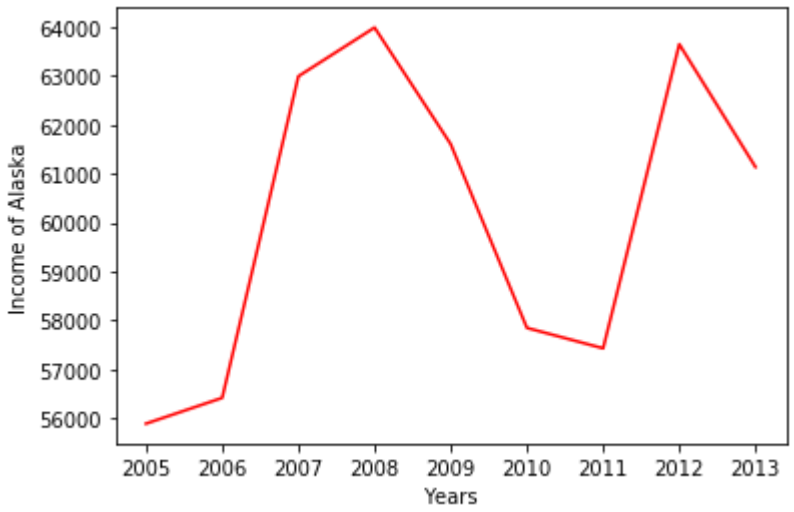
In [220]:

```
import matplotlib.pyplot as plt

plt.plot(years,alaska,'r')

plt.savefig('alaska.jpg')
plt.xlabel("Years")
plt.ylabel("Income of Alaska")
plt.show()

```



In [221]:
```
1 print(dir(pd))
```

```
['Categorical', 'CategoricalDtype', 'CategoricalIndex', 'DataFrame', 'DateOffse
t', 'DatetimeIndex', 'DatetimeTZDtype', 'ExcelFile', 'ExcelWriter', 'Float64Ind
ex', 'Grouper', 'HDFStore', 'Index', 'IndexSlice', 'Int16Dtype', 'Int32Dtype',
'Int64Dtype', 'Int64Index', 'Int8Dtype', 'Interval', 'IntervalDtype', 'Interval
Index', 'MultiIndex', 'NaT', 'Panel', 'Period', 'PeriodDtype', 'PeriodIndex',
'RangeIndex', 'Series', 'SparseArray', 'SparseDataFrame', 'SparseDtype', 'Spars
eSeries', 'TimeGrouper', 'Timedelta', 'TimedeltaIndex', 'Timestamp', 'UInt16Dty
pe', 'UInt32Dtype', 'UInt64Dtype', 'UInt64Index', 'UInt8Dtype', '__builtins__',
'__cached__', '__doc__', '__docformat__', '__file__', '__git_version__', '__loa
der__', '__name__', '__package__', '__path__', '__spec__', '__version__', '_has
htable', '_lib', '_libs', '_np_version_under1p13', '_np_version_under1p14', '_n
p_version_under1p15', '_np_version_under1p16', '_np_version_under1p17', '_tsli
b', '_version', 'api', 'array', 'arrays', 'bdate_range', 'compat', 'concat', 'c
ore', 'crosstab', 'cut', 'date_range', 'datetime', 'describe_option', 'errors',
'eval', 'factorize', 'get_dummies', 'get_option', 'infer_freq', 'interval_rang
e', 'io', 'isna', 'isnull', 'lreshape', 'melt', 'merge', 'merge_asof', 'merge_o
rdered', 'notna', 'notnull', 'np', 'offsets', 'option_context', 'options', 'pan
das', 'period_range', 'pivot', 'pivot_table', 'plotting', 'qcut', 'read_clipboa
rd', 'read_csv', 'read_excel', 'read_feather', 'read_fwf', 'read_gbq', 'read_hd
f', 'read_html', 'read_json', 'read_msgpack', 'read_parquet', 'read_pickle', 'r
ead_sas', 'read_sql', 'read_sql_query', 'read_sql_table', 'read_stata', 'read_t
able', 'reset_option', 'set_eng_float_format', 'set_option', 'show_versions',
'test', 'testing', 'timedelta_range', 'to_datetime', 'to_msgpack', 'to_numeri
c', 'to_pickle', 'to_timedelta', 'tseries', 'unique', 'util', 'value_counts',
'wide_to_long']
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [173]:
```
1 fp.columns
```

Out[173]:
```
Index(['GEOID', 'State', '2005', '2006', '2007', '2008', '2009', '2010',
       '2011', '2012', '2013'],
      dtype='object')
```

In [187]:
```
1 years=fp.columns[2:]
2 years
```

Out[187]:
```
Index(['2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013'],
dtype='object')
```

In [184]:
```
1 fp.values[1]
```

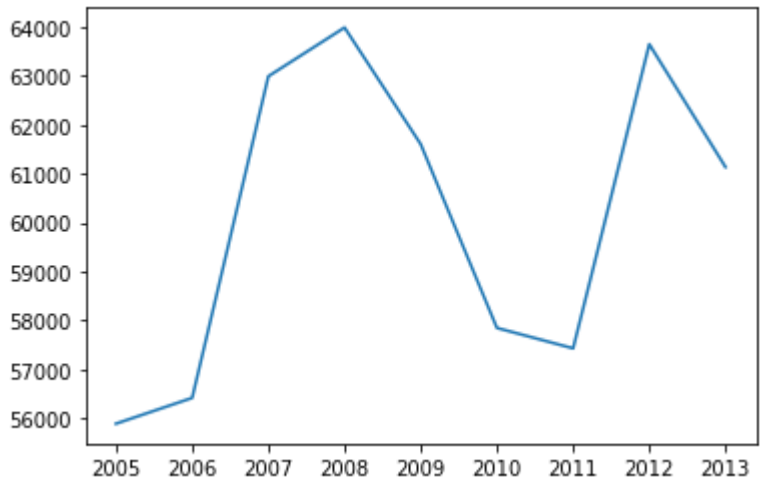
Out[184]:
```
array(['04000US02', 'Alaska', 55891, 56418, 62993, 63989, 61604, 57848,
       57431, 63648, 61137], dtype=object)
```

In [186]:
```
alaska=list(fp.values[1,2:])
alaska
```

Out[186]: [55891, 56418, 62993, 63989, 61604, 57848, 57431, 63648, 61137]

In [189]:
```
import matplotlib.pyplot as plt
plt.plot(years,alaska)
```

Out[189]: [<matplotlib.lines.Line2D at 0x199ee380eb8>]



In [ ]:
```

```

In [122]:
```
internalMark1={"Sub1":23,'Sub2':19,"Sub3":4}

internalMarkData=pd.Series(internalMark1)
internalMarkData
```

Out[122]:
```
Sub1    23
Sub2    19
Sub3     4
dtype: int64
```

In [125]:
```
i1={'s1':2,'s2':23}
i1=pd.Series(i1)
i1
```

Out[125]:
```
s1     2
s2    23
dtype: int64
```

In [126]:
```
i2={'s1':2,'s2':23}
i2=pd.Series(i2)
i2
```

Out[126]:
```
s1     2
s2    23
dtype: int64
```

In [145]:
```
1 final={'i1':i1,'i2':i2}
2 final=pd.DataFrame(final)
3 final
```

Out[145]:

	i1	i2
s1	2	2
s2	23	23

In [146]:
```
1 final.columns
```

Out[146]:
```
Index(['i1', 'i2'], dtype='object')
```

In [147]:
```
1 final.values
```

Out[147]:
```
array([[ 2,  2],
       [23, 23]], dtype=int64)
```

In [148]:
```
1 final
```

Out[148]:

	i1	i2
s1	2	2
s2	23	23

In [177]:
```
1 final.values[0]
2 avg={}
3 s1=(final.values[0][0]+final.values[0][1])//2
4 avg['s1']=s1
5 s2=(final.values[1][0]+final.values[1][1])//2
6 avg['s2']=s2
7 avg
```

Out[177]:
```
{'s1': 2, 's2': 23}
```

In [178]:
```
1 final['s3']=avg
```

In [179]:
```
1 final
```

Out[179]:

	i1	i2	Avarage	s3
s1	2	2	s1	s1
s2	23	23	s2	s2

In [ ]:
```
1
```