Revision- Week1

```
from IPython.display import Image

import numpy as np
import pandas as pd
```

1. Write 2 `mutable` data-types of python:

- list
- dictinoary
- set

2. Write 2 `immutable` data-types of python:

- int
- float
- tuple

```
Image(filename='../images/py_datatypes.png')
```

Bullet Points

- every variable is Python is an **object instance**.
- When an object is created, it is assigned a unique **object id**.
- The type of that **object** is defined at **runtime**
- For an object, **object id can never change**
- The **state** of that object **can be changed** if it is mutable.
- A **mutable** object can be changed after it is created, and an **immutable** object can't be.

```
#case 1:
x = [1,2,3]; y = x; x.pop()
```

3. is id(x) == id(y) ?

```
id(x) == id(y)
```

```
#case 2:
x = 3; y = x; x = x +1
```

4. is id(x) == id(y) ?

```
id(x) == id(y)
```

5. Find all errors in the function definition below

func add elems:(k=3,r)
x = r + k
return X

add elems(2,3)

```
def add_elems(r,k=3):
    x = r + k
    return x
add_elems(2,3)
```

6. Define a class:

Upon initialisation (`__init__`), the class should take a string from console input

The class should have 2 methods:

`get_upper` : to return the string in upper case.

`get_lower` : to return the string in lower case.

```
s = InputOutString()
```

```
s.get_upper()
```

```
s.get_lower()
```

```
class InputOutString(object):
    def __init__(self):
        self.s = input("Enter a string: ")

    def get_upper(self):
        return self.s.upper()

    def get_lower(self):
        return self.s.lower()
```

Numpy Quick Recap

```
Image(filename='../images/np_arrays.png')
```

Bullent Points: NumPy Array

1. ***Attributes of arrays***: Determining the size, shape, memory consumption, and data types of arrays

   `ndarray.size` `ndarray.shape` `ndarray.dtype`
2. ***Indexing of arrays***: Getting and setting the value of individual array elements

   `array[r,c]`
3. ***Slicing of arrays***: Getting and setting smaller subarrays within a larger array

   `array[start_row:end_row, start_col:end_col]`

   `row_indices=[r1,r2,..]; col_indices=[c1,c2..]; array[row_indices, col_indices]`
4. ***Reshaping of arrays***: Changing the shape of a given array

   `array.reshape(r,c)`
5. ***Joining and splitting of arrays***: Combining multiple arrays into one, and splitting one array into many

   `np.vstack` `np.hstack` `np.tile`

```
Image(filename='../images/np_broadcasting.png', width=800)
```

7. Using `lst` generate `arrint` as shown:

```
lst = [1,2,3,4.0]
```

```
array
```

```
array = np.array(lst, dtype=np.int32)
```

8. Write a function to generate a list of all even numbers between 0 to 100

```
my_list = []
for number in range(0, 100):
    if number % 2 == 0:
        my_list.append(number)
my_list
```

9. Write the same function as a list comprehension

```
my_list = [number for number in range(0,100) if number % 2 == 0]
my_list
```

10. For the above function, return the output as a 2D array with 2 rows

```
my_arr = np.array([number for number in range(0,100) if number % 2 == 0]).reshape(2,25)
my_arr.shape
```

11. What are the different attributes of an array?

1. ndim (the number of dimensions)
2. shape (the size of each dimension)
3. size (the total number of elements in the array)
4. dtype (the type of data in the array)
5. nbytes (lists the total memory consumed by the array (in bytes))

12. Write 5 array generating functions: (like numpy.arange())

- numpy.linspace
- numpy.random
- numpy.zeros
- numpy.ones
- numpy.eye
- numpy.empty

13. From array `a` , create array `b` as a deep copy:

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
a
```

```
b
```

```
b = a[1:3, 2:3].copy()
```

14. Reshape arr into 2D array with 3 rows:

```
twoD_arr = np.arange(1, 46)
```

```
twoD_arr = np.arange(1, 46).reshape(3, -1)
twoD_arr.shape
```

- The criterion to satisfy for providing the new shape is that '*The new shape should be compatible with the original shape*'
- numpy allow us to give one of new shape parameter as -1 (eg: (2,-1) or (-1,3) but not (-1, -1)).
- It simply means that it is an unknown dimension and we want numpy to figure it out. And numpy will figure this by looking at the 'length of the array and remaining dimensions' and making sure it satisfies the above mentioned criteria

15. From the array `twenty` below, generate the `result` as shown

```
twenty = (np.arange(4 * 5)).reshape(4, 5)
twenty
```

```
result
```

```
row_indices = [1, 2, 3]
col_indices = [1, 2, -1]
result = twenty[row_indices, col_indices]
```

16. Using conditional statements, generate `result` from `range_arr` as shown:

```
range_arr = np.arange(0, 10, 0.5)
range_arr
```

```
result
```

```
idxs = (range_arr > 5) & (range_arr < 7.5)
result = range_arr[idxs]
```

17. Using array `a` and `b`, create `vs` and `hs` as shown:

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

```
b = np.array([[5, 6]])
b
```

```
vs
```

```
hs
```

```
vs = np.vstack((a,b))
```

```
hs = np.hstack((a,b.T))
```

18. From the given array, generate row wise and column wise average as shown:

```
arr = np.arange(5 * 6).reshape(5, 6)
arr
```

```
avg1 = arr.mean(axis=0, keepdims=True) #column-wise
```

```
avg2 = arr.mean(axis=1, keepdims=True) #row-wise
```

19. Show how to do this?

Suppose that X is a data matrix of shape `(n,p)`. That is, there are `n` data points and `p` features per point. Often, we have to remove the mean from each feature. That is, we want to compute the mean for each feature and then remove the mean from each column.

```
# say n =10, and p =30
n = 10
p = 30
X = np.random.rand(n,p)
X.shape
```

```
# Compute the mean per column using the axis command
Xm = np.mean(X,axis=0)  # This is a p-dim (here (30,) matrix

# Subtract the mean
X_demean = X - Xm.reshape(1,30)
```

Read data from csv into a Pandas Dataframe

```
file='../data/loans data.csv'
```

```
ld=pd.read_csv(file)
```

```
ld.head(5)
```

```
ld.columns
```

```

```

```
ld["Home.Ownership"].unique()
```

```
ld["Home.Ownership"].value_counts()
```

```
ld.groupby("Home.Ownership")["Monthly.Income"].mean()
```

```

```

Select rows meeting logical condition, and only the specific columns .

```
condition = (ld['Home.Ownership']=='MORTGAGE') & (ld['Monthly.Income']>5000)
```

```
desired_cols = ['Home.Ownership','Monthly.Income']
```

```
result = ld.loc[condition,desired_cols]
result.shape
```

```

```

```
c = ld.drop(['Home.Ownership','Monthly.Income'],axis=1)
```

```

```

```
#clean the data
intrate_clean=ld['Interest.Rate'].map(lambda x: round(float(x.rstrip('%')))/100, 4)
```

```
intrate_clean=loansData['Interest.Rate'].map(lambda x: round(float(x.rstrip('%')))/100, 4)
print (intrate_clean[0:5]) #working

fico_clean=loansData['FICO.Range'].map(lambda x: float(str(x[0:3])))
print (fico_clean[0:5]) #working

loanamt_clean = loansData['Amount.Requested'].map(lambda x: float(x))
print (loanamt_clean[0:5]) #working

#swap the columns with column with clean data
#Note the column head doesn't change but only the data is replace
loansData['Interest.Rate']=intrate_clean
loansData['FICO.Range']=fico_clean
loansData['Amount.Requested'] = loanamt_clean
```

```
```

```
```

Bonus Questions

1. Compute Eigen Value and Eigen Vectors

`A * eigen_vector - eigen_value * eigen_vector = 0`

```
arr = np.arange(0,9).reshape(3,3)
arr
```

```
eigen_values, eigen_vectors = np.linalg.eig(arr)
```

```
print(eigen_values.shape)
eigen_values
```

```
print(eigen_vectors.shape)
```

```
eigen_vectors
```

```
vector = eigen_vectors[:,0]
vector
```

```
value = eigen_values[0]
value
```

```
np.matmul(arr,vector) - value*vector
```

2. Write a function to convert a list to dictionary as shown:

`mylist = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine']`

`mydict = { 0 : 'Zero', 1 : 'One', 2 : 'Two', 3 : 'Three', 4 : 'Four', 5 : 'Five', 6 : 'Six', 7 : 'Seven', 8 : 'Eight', 9 : 'Nine', }`

```
x = ['Zero', 'One', 'Two', 'Three', 'Four']
```

```
d = list_to_dict(x)
d
```

```
def list_to_dict(x):
    """Create a dictionary where the key is the ordinal of the object in the list
    and the value is the object itself."""

    d = {} #initialise an empty dictionary

    for index, value in enumerate(x):

        d[index] = value

    return d
```

3. Binary Search

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do linear search.The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

In Binary search, we search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

```
# Returns index of x in arr if present, else -1
def binarySearch (arr, left, right, num):

    # Check base case
    if right >= left:

        mid = int((left + right)/2)

        # If element is present at the middle itself
        if arr[mid] == num:
            return mid

        # If element is smaller than mid, then it
        # can only be present in left subarray
        elif arr[mid] > num:
            return binarySearch(arr, left, mid-1, num)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid+1, right, num)

    else:
        # Element is not present in the array
        return -1
```

```
# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print("Element is present at index %d" % result)
else:
    print("Element is not present in array")
```

4. Given a list of numbers, write a function to return outliers?

How are outliers defined?

Here we can assume that outliers are defined as numbers outside of `[Q1 - 1.5 * IQR, Q3 + 1.5 * IQR]` where `Q1` is the first quartile, `Q3` is the third quartile, and `IQR` is the inner quartile range `(Q3 - Q1)`

The runtime of this algorithm is at least O(n), but potentially larger depending on how numpy computes quartiles.

```
L = [1,2,3,4,6,1,2,3,4,6,1,99,-100,1,2,3,4,5,2,2,22,34,1,2,3,4,-10]
findOutliers(L)
```

```
def findOutliers(L):
    Q1 = np.percentile(L, 25)
    Q3 = np.percentile(L, 75)
    IQR = Q3 - Q1
    outliers = []

    for num in L:
        if num < Q1 - 1.5 * IQR:
            outliers.append(num)
        elif num > Q3 + 1.5 * IQR:
            outliers.append(num)

    return outliers
```

5. Write a function to convert a list to dictionary as shown:

`mylist = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine']`

`mydict = { 0 : 'Zero', 1 : 'One', 2 : 'Two', 3 : 'Three', 4 : 'Four', 5 : 'Five', 6 : 'Six', 7 : 'Seven', 8 : 'Eight', 9 : 'Nine', }`

```
x = ['Zero', 'One', 'Two', 'Three', 'Four']
```

```
d = list_to_dict(x)
d
```

```
def list_to_dict(x):
    """Create a dictionary where the key is the ordinal of the object in the list
    and the value is the object itself."""

    d = {} #initialise an empty dictionary

    for index, value in enumerate(x):

        d[index] = value
```

```
    return d
```

```
```