```python
#*********************************************#
#
#   Review one Issue in Lecture 3
#
#*********************************************#

import numpy as np
import pandas as pd
from pandas import Series, DataFrame

alist = [1,2,3]
result = [(0 if c>2 else c) for c in alist]
print("alist:\n",result)

nlist = []
for c in alist:
    if c > 2 :
        nlist.append(0)
    else:
        nlist.append(c)

print(nlist)

arr = np.array([1,2,3])
result = [(0 if c>2 else c) for c in arr]
print("arr-to-list:\n",result)

arr = np.array([[1,2,3],[2,3,4]])
result =[(0 if c > 2 else c) for t in arr for c in t]
print(result)

result = np.reshape(result,(2,3))
print(result)

print(np.where(arr > 2, 0, arr))

#*********************************************#
#
#    Pandas -  DataFrame
#
#*********************************************#
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'], 'year': [2000,
2001, 2002, 2001, 2002],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
print(frame)

print(DataFrame(data, columns=['year','state','pop']))
print(DataFrame(data, columns=['year','state','pop','debt'])) # debt doesn't exist
```

```

# A column in a DataFrame can be retrieved as a Series either by dict-like notation or by attribute:
print(frame.columns)
print(frame['state'])
print(frame.state)

print(frame)

#Rows can also be retrieved by position or name by a couple of methods, such as the ix indexing field
print(frame.ix[3])

frame['debt'] = 16.5
print(frame)

# For example, the empty 'debt' column could be assigned a scalar value or an array of values
frame['debt'] = np.arange(5.)
print(frame)

# When assigning lists or arrays to a column, the value's length must match the length of the DataFrame.
# If you assign a Series, it will be instead conformed exactly to the DataFrame's index, inserting missing values in any holes:

val = Series([-1.2, -1.5, -1.7], index=[2, 4, 5])
frame['debt'] = val
print(frame)

#Assigning a column that doesn't exist will create a new column.

frame['eastern'] = 1
print(frame)


frame['marks'] = frame.state == 'Ohio' # if, select  target value
del frame['eastern']
print(frame)

# Index Objects
obj = Series(range(3), index=['a', 'b', 'c'])
print(obj)

# Index objects are immutable index[1] = 'd'

# Reindexing
# Calling reindex on this Series rearranges the data according to the new index,
# introducing missing values if any index values were not already present:
```

```

obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
print(obj2)

obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'], fill_value=0)
print(obj2)

# For ordered data like time series, it may be desirable to do some interpolation or filling of values when reindexing.
# The method option allows us to do this, using a method such as ffill which forward fills the values:

obj3 = Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
print (obj3)
obj3 = obj3.reindex(range(6), method='ffill')
print(obj3)

# ffill or pad : Fill (or carry) values forward, bfill or backfill : Fill (or carry) values backward

# With DataFrame, reindex can alter either the (row) index, columns, or both.

frame = DataFrame(np.arange(9).reshape((3, 3)), index=['a', 'c', 'd'], columns=['Ohio', 'Texas', 'California'])
print(frame)


# When passed just a sequence,  the rows are reindexed in the result:
frame2 = frame.reindex(['a', 'b', 'c', 'd'])
print(frame2)

# The columns can be reindexed using the columns keyword:
states = ['Texas', 'Utah', 'California']
frame = frame.reindex(columns=states)

print(frame)

# Both can be reindexed in one shot, though interpolation will only apply row-wise(axis 0)
frame = frame.reindex(index=['a', 'b', 'c', 'd'], method='ffill', columns=states)
print(frame)


# Dropping entries from an axis

obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
new_obj = obj.drop('c')
print(new_obj)

```

```
# With DataFrame, index values can be deleted from either axis:

data = DataFrame(np.arange(16).reshape((4, 4)), index=['Ohio', 'Colorado', 'Utah',
 'New York'], columns=['one', 'two', 'three', 'four'])

# print(data)
# for i in data.items():
#     print("items in data \n",i)


data.drop(['Colorado', 'Ohio'])

print(data)
data.drop('two', axis=1)
print(data)
# Summarizing and Computing Descriptive Statistics
print(data.describe())

print(data.sum())
print(data.sum(axis =1 ))

data.ix["ohio"] = None
print(data)
data1 = data.mean(axis=0, skipna=True)
print(data1)

#like idxmin and idxmax, return indirect statistics like the index value where the
 minimum or maximum values are attained:
print("idmax = \n",data.idxmax())





```