PART A: CS 5890/6890 - Grads and Undergrads

Welcome to Assignment 0 !

In this assignment, we will explore the Time series decomposition library in python and apply the knowledge we have seen in the class.

You are a brilliant scientist that works at NOAA (National Aceanic & atmospheric administration), and part of your job is to analyse the precipitation data in different regions in the country. Today your job is to analyze the precipitation time series in the "Salmon River".

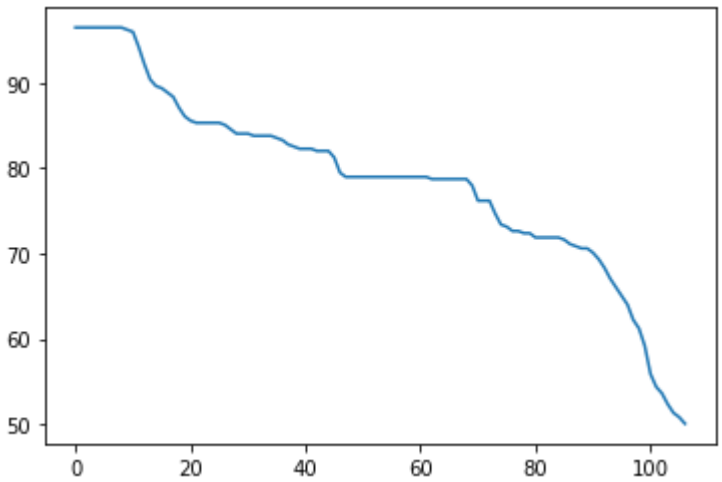
Question 0: Upload the 'Precipitation.csv' under the sample_data folder and open the files for reading. Then, print and plot the data.

In [1]:

```
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import acf
import statsmodels.api as sm
import pandas as pd
import pandas.util.testing as tm
import matplotlib.pyplot as plt

data = pd.read_csv('sample_data/Precipitation.csv')

# Print the data file using the following comand  print(data)
# INSERT CODE HERE
print(data)
# Visualize the uploaded data file using the following comand   plt.plot(data
# INSERT CODE HERE
plt.plot(data)
```

```
     thickness_of_precipitation_amount
0                                96.520
1                                96.520
2                                96.520
3                                96.520
4                                96.520
..                                  ...
102                              53.594
103                              52.324
104                              51.308
105                              50.800
106                              50.038

[107 rows x 1 columns]
<ipython-input-1-d5ffd244180d>:6: FutureWarning: pandas.util.testing is depre
cated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

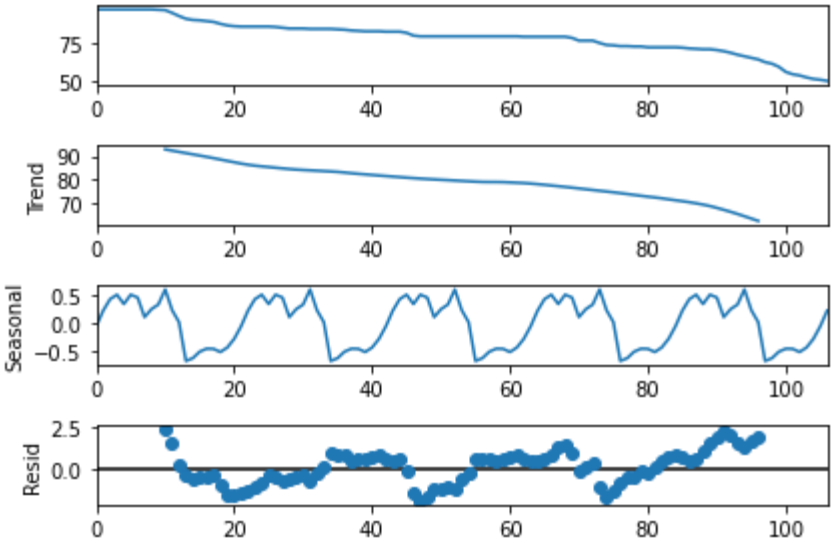
Out[1]: `[<matplotlib.lines.Line2D at 0x7f0f7fae2820>]`



Question 1: decompose the newly read time series data into its different components using the TSA python library

In [2]:

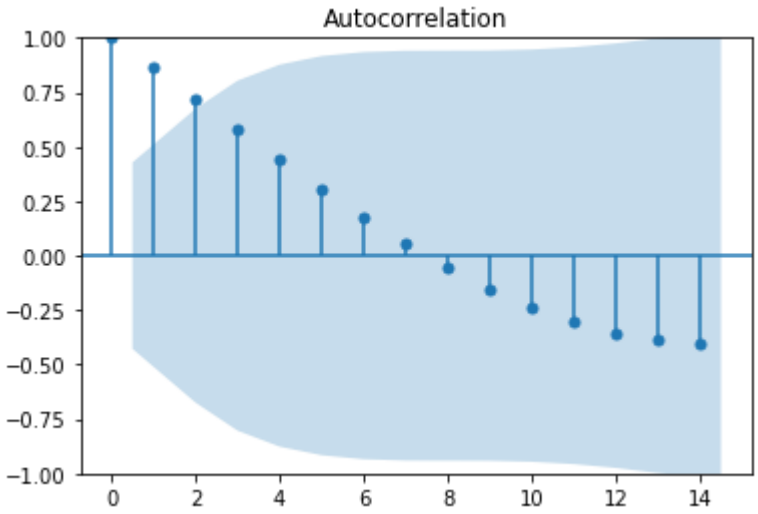
```
# Freq parameter refers to the number of points taken into account when compu
decompose_result = seasonal_decompose(data, period= 21, model="additive")

# Plot the TS extracted components
decompose_result.plot()

# To get different  components of the TS use the following:
# decompose_result.trend or decompose_result.seasonal or decompose_result.res

trend = decompose_result.trend
seasonal = decompose_result.seasonal # INSERT CODE HERE
residual = decompose_result.resid # INSERT CODE HERE
```



Question 2: Generate the autocorollation plot (ACF) to check whether there is any uncovered trend or seasonality and remove the K missing values from the trend line.

In [3]:
```
# You can compute the ACF plot using the following command:    acf(data)
ACF = acf(data) # INSERT CODE HERE
sm.graphics.tsa.plot_acf(ACF)
```

Out[3]:



Notice that there are "**K**" elements with missing values (NAN values) at the beginning and at the end of the trend series that corresponds to the formula we have seen in class:

$$m_t = \sum_{-k}^{k} \frac{1}{2k+1} x_{t+1}$$

In [4]:
```
# You can print the trend component using             print(trend.values)
print(trend.values) # INSERT CODE HERE
```

```
[        nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan 93.00028571 92.46809524
 91.93590476 91.40371429 90.87152381 90.33933333 89.79504762 89.22657143
 88.63390476 88.0412381  87.46066667 86.88009524 86.38419048 85.98504762
 85.67057143 85.38028571 85.09       84.79971429 84.52152381 84.29171429
 84.11028571 83.95304762 83.79580952 83.63857143 83.48133333 83.28780952
 83.00961905 82.71933333 82.4532381  82.21133333 81.96942857 81.72752381
 81.49771429 81.26790476 81.03809524 80.80828571 80.59057143 80.38495238
 80.20352381 80.03419048 79.87695238 79.71971429 79.55038095 79.39314286
 79.23590476 79.07866667 78.95771429 78.92142857 78.90933333 78.86095238
 78.72790476 78.59485714 78.46180952 78.25619048 77.99009524 77.71190476
 77.40952381 77.10714286 76.79266667 76.47819048 76.13952381 75.80085714
 75.46219048 75.13561905 74.80904762 74.47038095 74.10752381 73.73257143
 73.34552381 72.95847619 72.58352381 72.25695238 71.882      71.44657143
 71.03533333 70.63619048 70.2007619  69.70485714 69.16057143 68.53161905
 67.74542857 66.91085714 66.04       65.10866667 64.12895238 63.12504762
 62.09695238         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan]
```

Before detrending the original data, remove the k missing values from the trend and original data signals

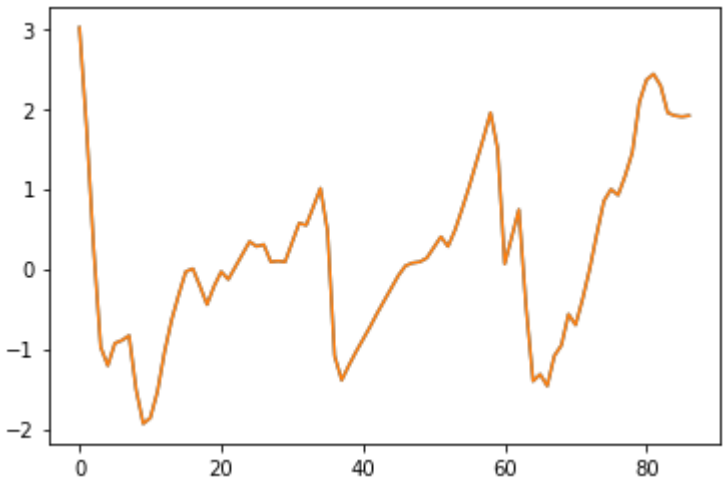
In [5]:
```
# First remove the K missing values from the trend components and their corre
K = 10 #INSET VALUE HERE#

shortData = data[K:len(data)-K]
trend = trend[K:len(trend)-K]
```

Question 3: Detrend the original data by substracting the extracted trend from the data and plot the detrended data. Check whether your computations are correct by computing the detrended data using by addding (seasonality plus residual)

In [6]:
```
detrend = [d-t for d,t in zip(shortData.values,trend.values)]

# Plot the detrended
plt.plot(detrend) # INSERT CODE HERE

detrend_data = residual.values + seasonal.values

# Print the new detrend_data
# INSERT CODE HERE
print(detrend_data[K:len(detrend_data)-K]) #I'm assuming the nan values are n
# Plot and show the new detrend_data plot
# INSERT CODE HERE
plt.plot(detrend_data[K:len(detrend_data)-K]) #Again assuming the nan values
```

```
[ 3.01171429  1.76590476  0.26609524 -0.97971429 -1.20952381 -0.93133333
 -0.89504762 -0.83457143 -1.51190476 -1.9352381  -1.86266667 -1.53609524
 -1.04019048 -0.64104762 -0.32657143 -0.03628571  0.         -0.21771429
 -0.44752381 -0.21771429 -0.03628571 -0.13304762  0.02419048  0.18142857
  0.33866667  0.27819048  0.30238095  0.08466667  0.0967619   0.08466667
  0.32657143  0.56847619  0.54428571  0.77409524  1.00390476  0.47171429
 -1.08857143 -1.39095238 -1.20952381 -1.04019048 -0.88295238 -0.72571429
 -0.55638095 -0.39914286 -0.24190476 -0.08466667  0.03628571  0.07257143
  0.08466667  0.13304762  0.26609524  0.39914286  0.27819048  0.48380952
  0.74990476  1.02809524  1.33047619  1.63285714  1.94733333  1.49980952
  0.06047619  0.39914286  0.73780952 -0.45961905 -1.40304762 -1.31838095
 -1.46352381 -1.08857143 -0.95552381 -0.56847619 -0.70152381 -0.37495238
  0.          0.43542857  0.84666667  0.99180952  0.9192381   1.16114286
  1.45142857  2.08038095  2.35857143  2.43114286  2.286       1.94733333
  1.91104762  1.89895238  1.91104762]
```

Out[6]: `[<matplotlib.lines.Line2D at 0x7f0f7f0c02b0>]`



In [7]:
```
#It just looks like one plot, since the lines overlap.
```

Question 4: What can be said about the following precipitation time series components:

a. Trend?

b. Seasonality?

c. Cyclicity?

d. Stationarity?

In [8]:
```
# (a) There is a downward trend.
# (b) There is an obvious component of periodicity found by the algorithm, bu
# by which the rainfall varies each period is comparitively small, even when
# residual amount varies. Thus, while it is appealing to think that there wou
# rainfall, the seasonal component of this data appears to be very small.
# (c) There does not seem to be any cyclicity.
# (d) The data do not appear to be stationary, due to the downward trend and
```

Question 5: Implement two functions called get_MovingAverage and get_MovingMedian that takes 2 arguments:

- data: the original data time series
- k : that helps you find how many elements the MA and MM should consider

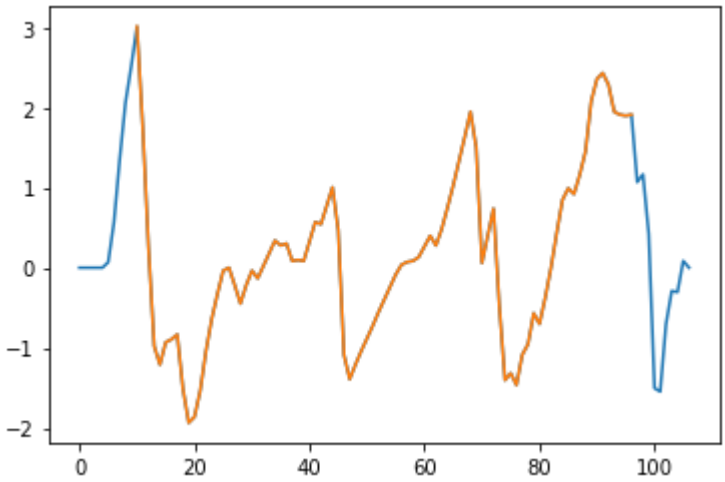
The functions should return the detrended data

Remember: the number of points considered in every window of the moving average is 2K+1

In [9]:
```
def get_MovingAverage(data, K):
    #detrended = []
    trended = []
    L = len(data)
  #detrended = []
  # INSERT CODE HERE.   To get the average of a list (for example [1,2,3]) us
    # we note that mt = sum(1/(2k+1) x_{t+i}, i=-k..k). But what if "i" is to
    for i in range(L):
        if (i >= K) and (i <= (L-K)-1): #This is the typical case.
            sublist = data[i-K:i+1+K]
            trended.extend([np.mean(sublist)])
        elif i < K: #What if the data point is closer than K to the beginning
            sublist = data[0:2*i+1]
            trended.extend([np.mean(sublist)])
        else: #Same issues here. Why get nan values when you can take a crack
            d = L-i
            sublist = data[L-2*d+1:L]
            trended.extend([np.mean(sublist)])

    detrended = [data[i:i+1].values[0][0]-trended[i] for i in range(L)] #detr
    return detrended
  #return detrended


# When done, uncomment the code below, you should get the same results of the
detrended = get_MovingAverage(data, 10)
plt.plot(detrended)
plt.plot(detrend_data) # I added this part to show that the main difference i
plt.show()
```

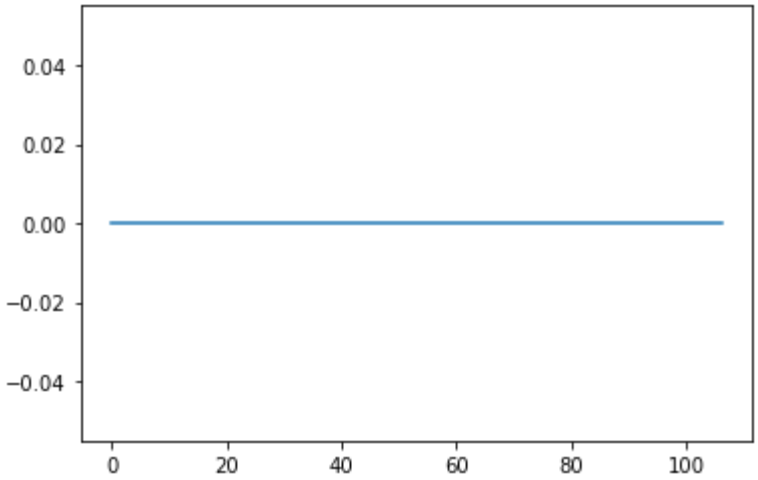
```
/home/ben/anaconda3/lib/python3.8/site-packages/numpy/core/fromnumeric.py:347
2: FutureWarning: In a future version, DataFrame.mean(axis=None) will return
a scalar mean over the entire DataFrame. To retain the old behavior, use 'fra
```

me.mean(axis=0)' or just 'frame.mean()'



In [10]:
```
#It only differs from the previous plot in that I defined the trend for value
```

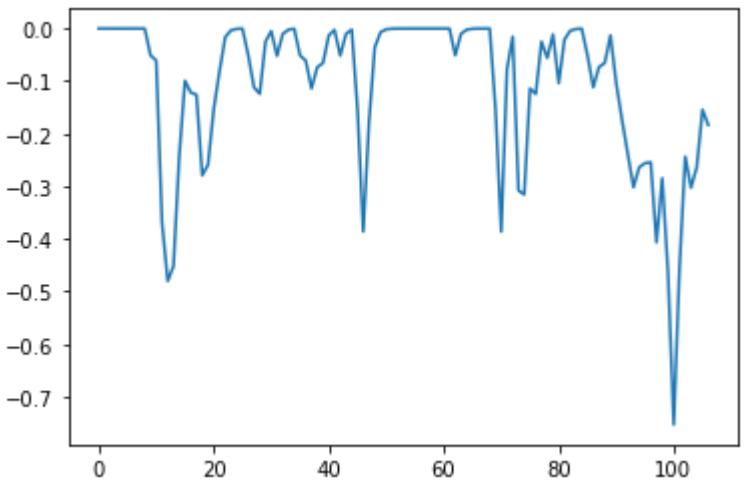
In [11]:
```
def get_MovingMedian(data, K):
  #detrended = []
  # INSERT CODE HERE.  To get the median of a list (for example [1,2,3])  use
    #detrended = []
    trended = []
    L = len(data)
    # we note that mt = sum(1/(2k+1) x_{t+i}, i=-k..k). But what if "i" is to
    for i in range(L):
        if (i >= K) and (i <= (L-K)-1):
            sublist = data[i-K:i+1+K]
            trended.extend([np.median(sublist)])
        elif i < K:
            sublist = data[0:2*i+1]
            trended.extend([np.median(sublist)])
        else:
            d = L-i
            sublist = data[L-2*d+1:L]
            trended.extend([np.median(sublist)])

    detrended = [data[i:i+1].values[0][0]-trended[i] for i in range(L)] #detr
    return detrended
  #return detrended

# When done, uncomment the code below
detrended = get_MovingMedian(data, 10)
plt.plot(detrended)
plt.show()
```



CS 6890 - ONLY FOR GRADUATE STUDENTS:

Question 6: Implement the exponential smoothing we have seen in class with alpha=0.8.

In [12]:
```
def get_ExponentialSmoothning(data, alpha=.8):
    #detrended = []
    trended = []
    L = len(data)
    for i in range(L):
        if i == 0:
            trended.extend([data.values[i][0]])
        else:
            term = alpha*data.values[i][0] + (1-alpha)*trended[i-1]
            trended.extend([term])

    detrended = [data[i:i+1].values[0][0]-trended[i] for i in range(L)] #detr
    return detrended
  #return detrended

detrended = get_ExponentialSmoothning(data)
plt.plot(detrended)
plt.show()
```



In [ ]: