Import Necessary Package

In [405]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Object of One Swing

For data analysis on a large dataframe, Python is a good analytical programming language since programmers can take advantage of existing packages and develop an application very fast. Therefore, I chose Python to complete this code challenge on data of a accelerometer and a gyroscope. Besides, in order to gain better performance on data processing, I utilize the numpy package, which is generally implemented in C. In brief, Python can help us to develop a feature very fase, but if we still need even better performance, we can implement the feature in C/C++.

The class swing contains data of sensors of one swing. It can be initialized by **swing(filename)**. I use pandas dataframe to contain all the data from .csv file, which is stored in **self.df**. For each member function, the targeted column can be designated by the argument **dataCol**, and other arguments also have to be specified since we do not have default values for now.

Examples of usage of functions can be found in **test.py**.

In [406]:

```
class swing:
    # Initialize a swing by reading csv file
    def __init__(self, filename):
        self.df = pd.read_csv(filename, header=None)
        self.df.columns = ['timestamp', 'ax', 'ay', 'az', 'wx', 'wy', 'wz']

    # Print data in a column (or the whole data frame by default)
    def print(self, col="default"):
        if col is "default":
            print(self.df)
        else:
            assert col in self.df.columns, "invalid column, should be one of [timestam
p, ax, ay, az, wx, wy, wz]"
            print(self.df[col])

    def plot(self, col="default"):
        if col is "default":
            for col in self.df.columns:
                if col is not "timestamp":
                    data = self.df[col].values
                    x = np.arange(len(data))
                    plt.plot(x, data, linewidth=2)
            plt.title("Plt for all columns")
            plt.xlabel("Index")
            plt.ylabel("Value")
        else:
            data = self.df[col].values
            x = np.arange(len(data))
            plt.plot(x, data, linewidth=2)
            plt.title("Plt for " + col)
            plt.xlabel("Index")
            plt.ylabel("Value")


    '''
    Search values above threshold
    Return the first index where data have such values for at least winLength in a row
    Return -1 if cannot find the values
    '''
    def searchContinuityAboveValue(self, dataCol, indexBegin, indexEnd, threshold, winL
ength):
        assert dataCol in self.df.columns, "invalid column, should be one of [timestam
p, ax, ay, az, wx, wy, wz]"
        assert (indexBegin >= 0 and indexBegin < len(self.df.index)), "invalid begin in
dex"
        assert (indexEnd >= 0 and indexEnd < len(self.df.index)), "invalid end index"
        assert (indexBegin < indexEnd), "invalid pair of indexBegin and indexEnd"

        # Extract valid data by dataCol and index range
        data = self.df[dataCol].values
        data = data[indexBegin: indexEnd + 1]

        # Get bitmap that data are above threshold
        bitmap = (data > threshold).astype(int)
        bitmap = np.insert(bitmap, 0, 0) # for first entry

        # Find points going above threshold and going below threshold
        diff = np.diff(bitmap)
        higherPoints = np.where(diff == 1)[0]
```

```
        lowerPoints  = np.where(diff == -1)[0]

        # If not going below threshold at the end
        if len(lowerPoints) < len(higherPoints):
            lowerPoints = np.append(lowerPoints, len(data))

        # Check how many valid values in a row and return the first valid index meeting
the requirements
        length = np.subtract(lowerPoints, higherPoints)
        bitmapLength = (length >= winLength)
        indexList = np.where(bitmapLength == True)[0]
        if indexList.size == 0:
            return -1
        else:
            return higherPoints[indexList[0]] + indexBegin;

    '''
    Search values within range in backward direction (indexBegin > indexEnd)
    Return the first index where data have such values for at least winLength in a row
    Return -1 if cannot find the values
    '''
    def backSearchContinuityWithinRange(self, dataCol, indexBegin, indexEnd, thresholdL
o, thresholdHi, winLength):
        assert dataCol in self.df.columns, "invalid column, should be one of [timestam
p, ax, ay, az, wx, wy, wz]"
        assert (indexBegin >= 0 and indexBegin < len(self.df.index)), "invalid begin in
dex"
        assert (indexEnd >= 0 and indexEnd < len(self.df.index)), "invalid end index"
        assert (indexBegin > indexEnd), "invalid pair of indexBegin and indexEnd"
        assert (thresholdLo <= thresholdHi), "invalid threshold, should have thresholdL
o <= thresholdHi"

        # Extract valid data by dataCol and index range and reverse it
        data = self.df[dataCol]
        data = data[indexEnd: indexBegin + 1]
        data = np.array(data[::-1])

        # Get bitmap that data are within range
        bitmap = (np.logical_and(data > thresholdLo, data < thresholdHi)).astype(int)
        bitmap = np.insert(bitmap, 0, 0) # for the first entry

        # Find points going into range and out of range
        diff = np.diff(bitmap)
        InRangePoints = np.where(diff == 1)[0]
        OutRangePoints = np.where(diff == -1)[0]

        # If not going out of range at the end
        if len(OutRangePoints) < len(InRangePoints):
            OutRangePoints = np.append(OutRangePoints, len(data))

        # Check how many valid values in a row and return the first valid index meeting
the requirements
        length = np.subtract(OutRangePoints, InRangePoints)
        bitmapLength = (length >= winLength)
        indexList = np.where(bitmapLength == True)[0]
        if indexList.size == 0:
            return -1
        else:
            return indexBegin - InRangePoints[indexList[0]];

    '''
```

```
    Search values above threshold in two signals
    Return the first index where data have such values for at least winLength in a row
    Return -1 if cannot find the values
    '''
    def searchContinuityAboveValueTwoSignals(self, dataCol1, dataCol2, indexBegin, inde
xEnd, threshold1, threshold2, winLength):
        assert dataCol1 in self.df.columns, "invalid column1, should be one of [timesta
mp, ax, ay, az, wx, wy, wz]"
        assert dataCol2 in self.df.columns, "invalid column2, should be one of [timesta
mp, ax, ay, az, wx, wy, wz]"
        assert dataCol1 is not dataCol2, "invalid columns: two same columns"
        assert (indexBegin >= 0 and indexBegin < len(self.df.index)), "invalid begin in
dex"
        assert (indexEnd >= 0 and indexEnd < len(self.df.index)), "invalid end index"
        assert (indexEnd > indexBegin), "invalid pair of indexBegin and indexEnd"

        # Extract valid data by dataCol and index range
        data1 = self.df[dataCol1]
        data2 = self.df[dataCol2]
        data1 = self.df[dataCol1].values
        data2 = self.df[dataCol2].values
        data1 = data1[indexBegin: indexEnd + 1]
        data2 = data2[indexBegin: indexEnd + 1]

        # Get bitmap that data are above both threshold
        bitmap = (np.logical_and(data1 > threshold1, data2 > threshold2)).astype(int)
        bitmap = np.insert(bitmap, 0, len(data1)) # for the first entry

        # Find points going above thresold and below threshold
        diff = np.diff(bitmap)
        higherPoints = np.where(diff == 1)[0]
        lowerPoints = np.where(diff == -1)[0]

        # If not going below threshold at the end
        if len(lowerPoints) < len(higherPoints):
            lowerPoints = np.append(lowerPoints, 0)

        # Check how many valid values in a row and return the first valid index meeting
the requirements
        length = np.subtract(lowerPoints, higherPoints)
        bitmapLength = (length >= winLength)
        indexList = np.where(bitmapLength == True)[0]
        if indexList.size == 0:
            return -1
        else:
            return higherPoints[indexList[0]] + indexBegin;

    '''
    Search values within range
    Return list of tuples of (beginIndex, endIndex) where data have such values for at
 least winLength in a row
    Return [(-1, -1)] if cannot find the values
    '''
    def searchMultiContinuityWithinRange(self, dataCol, indexBegin, indexEnd, threshold
Lo, thresholdHi, winLength):
        assert dataCol in self.df.columns, "invalid column, should be one of [timestam
p, ax, ay, az, wx, wy, wz]"
        assert (indexBegin >= 0 and indexBegin < len(self.df.index)), "invalid begin in
dex"
        assert (indexEnd >= 0 and indexEnd < len(self.df.index)), "invalid end index"
        assert (thresholdLo <= thresholdHi), "invalid threshold, should have thresholdL
```

```
o <= thresholdHi"
        assert (indexEnd > indexBegin), "invalid pair of indexBegin and indexEnd"

        # Extract valid data by dataCol and index range
        data = self.df[dataCol].values
        data = data[indexBegin : indexEnd + 1]

        # Get bitmap that data are within range
        bitmap = (np.logical_and(data > thresholdLo, data < thresholdHi)).astype(int)
        bitmap = np.insert(bitmap, 0, 0)

        # Find points going into range and out of range
        diff = np.diff(bitmap)
        InRangePoints = np.where(diff == 1)[0]
        OutRangePoints = np.where(diff == -1)[0]

        # If not going out of range at the end
        if len(OutRangePoints) < len(InRangePoints):
            OutRangePoints = np.append(OutRangePoints, len(data))

        # Check how many valid values in a row and return the first valid index meeting
the requirements
        length = np.subtract(OutRangePoints, InRangePoints)
        bitmapLength = (length >= winLength)
        indexList = np.where(bitmapLength == True)[0]
        if indexList.size == 0:
            return [(-1, -1)]
        else:
            tuplesList = []
            for i in list(indexList):
                begin = InRangePoints[i] + indexBegin
                end = OutRangePoints[i] + indexBegin - 1
                tuplesList.append((begin, end))
            return tuplesList
```

In [408]:

```
latestSwing = swing('./latestSwing.csv')
latestSwing.plot()
```