```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  5 12:56:37 2020

@author: jimmy
"""
# Import relevant modules/packages
import numpy as np
import matplotlib.pyplot as plt
from astropy.timeseries import LombScargle
from ReadFile import Read

class Periodicity:

    def __init__(self,filename,objectname,colNames,numPoints=None,contiguous='True',period=None
        # Inputs:
        #     filename: file path or file name (if file in same folder as notebook)
        #     objectname: name of object you're plotting curve of
        #     numPoints: number of data points you want to use from the file
        #     period: period of plot feature (only used if xaxis='Phase' to fold the data)
        self.file = filename
        self.object = objectname
        self.numPoints = numPoints
        self.period = period

        # Extract time, flux, and error data from text file
        self.times,self.fluxes,self.errors = Read(self.file,colNames)

        # Decide what times array to make
        if self.numPoints == None:
            self.times = [time - self.times[0] for time in self.times]
        else:
            if contiguous == 'True':
                # Select contiguous interval of points
                self.times = [time - self.times[0] for time in self.times[:self.numPoints]]
                self.fluxes = self.fluxes[:self.numPoints]
                self.errors = self.errors[:self.numPoints]
            elif contiguous == 'Bookend':
                # Select sparsed interval of data
                self.times_i = [time - self.times[0] for time in self.times[:self.numPoints]]
                self.times_f = [time - self.times[0] for time in self.times[-self.numPoints:]]
                self.times = self.times_i + self.times_f

                # Concatenate first and last n elements of flux list
                self.fluxes_i = self.fluxes[:self.numPoints]
                self.fluxes_f = self.fluxes[-self.numPoints:]
                self.fluxes = [*self.fluxes_i,*self.fluxes_f]

                # Concatenate first and last n elements of error list
                self.errors_i = self.errors[:self.numPoints]
                self.errors_f = self.errors[-self.numPoints:]
                self.errors = [*self.errors_i,*self.errors_f]
            elif contiguous == 'Random':
                # Generate random list of indices
                randoms = np.random.randint(0, high=len(self.times), size=numPoints)
                #print(randoms)
                # Select random sparsed interval of data
                self.times = self.times[randoms]

                # Concatenate first and last n elements of flux list
                self.fluxes = self.fluxes[randoms]

                # Concatenate first and last n elements of error list
                self.errors = self.errors[randoms]
```

```
# Function for plotting light curves from text file
def LightCurve(self,plot=True,xaxis='Time',curve='Flux'):
    # Inputs:
    #     plot: boolean to decide whether to plot the data (True) or not (False)
    #     xaxis: decide which x parameter to calculate/plot ('Time or Phase')
    #     curve: string that indicates y parameter being plotted (used in axis label)
    # Returns:
    #     xdata: array with data from x-axis
    #     fluxes: array with associated y-axis data
    #     errors: measurement errors read from text file

    # Define list of data to plot on x-axis (time or phase)
    xdata = []

    if plot == True:
        # Initialize axis figure and axis
        fig = plt.figure()
        ax = fig.add_subplot(111)

        # Decide what x-axis should be
        if xaxis == 'Time':
            xlabel = 'Time (days)'

            # Plot flux vs time
            ax.scatter(self.times,self.fluxes)

            xdata = self.times

        elif xaxis == 'Phase':
            xlabel = 'Phase'

            # Calculate phase from time data
            phases = [(time%self.period)/self.period for time in self.times]

            # Make a scatter plot of flux vs. phase
            ax.scatter(phases,self.fluxes,label='Period = {0:.3f} days'.format(self.period)

            xdata = phases
            ax.set_ylim(0.999,1.0006)
            ax.set_xlim(0.0,0.2)

        # Add plot features
        ax.set_xlabel(xlabel,fontsize=14)
        ax.set_ylabel('{0}'.format(curve),fontsize=14)
        ax.set_title('{0} vs. {1} for {2}'.format(curve,xaxis,self.object),fontsize=18)
        ax.legend()
    else:
        # Decide what x-axis should be
        if xaxis == 'Time':
            xdata = self.times
        elif xaxis == 'Phase':
            # Calculate phase from time data
            phases = [(time%self.period)/self.period for time in self.times]
            xdata = phases

    return(xdata,self.fluxes,self.errors)

# Function to generate power spectrum from a flux vs. time dataset
def LS(self,minP,maxP,numIntervals,i,flux,plot=False,trueP=None):

    if flux == []:
        flux = self.fluxes

    # Define range of frequencies to search over
    minfreq = 1./maxP
    maxfreq = 1./minP
```

```
        # Make list of frequencies within the range
        frequency = np.linspace(minfreq,maxfreq,numIntervals)

        # Use LombScargle method to calculate power as a function of those frequencies
        power = LombScargle(self.times,flux,self.errors,nterms=i).power(frequency)

        # Find maximum power and frequency/period of maximum power
        maxp = np.max(power)
        maxind = np.argmax(power)

        maxfreq = frequency[maxind]
        best_period = 1./maxfreq

        if plot == True:
            # Plot power spectrum using lists from above
            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.plot(frequency,power)

            # Set axes limits
            ax.set(xlim=(frequency[0],frequency[-1]), ylim=(0,np.max(power)))
            ax.set_xlabel('Frequency (1/days)',fontsize=14)
            ax.set_ylabel('Power',fontsize=14)
            ax.set_title('Power vs. Freq. for {0}'.format(self.object),fontsize=18)

            # Plot line indicating period of system from SIMBAD
            if trueP != None:
                ax.vlines(1./trueP,0,1,linestyle='dashed',label='Published Period ({0:.3f} days

            # Plot vertical line of best period
            ax.vlines(1./best_period,0,1,linestyle='dashed',label='Dominant Period ({0:.3f} day
            ax.legend(loc='center left',bbox_to_anchor=(1, 0.5))

        return(maxp)

    # Function to calculate the false alarm probability of a radial velocity detection
    def FAP(self,numIterations):
        # Calculate stats of errors on RV measurements
        meanErr = np.mean(self.errors)
        stddevErr = np.std(self.errors)
        length = len(self.errors)

        # Empty list of maximum powers
        maxPowers = []
        numExceed = 0

        # Set number of iterations for loop
        numIterations = 10000

        # Calculate max power from non-noisy data
        original_maxPower = self.LS(35,45,1000,1,flux=self.fluxes)

        # Monte Carlo simulation of 10000 noise profiles
        # Used to calculate False Alarm Probability (FAP)
        for i in range(numIterations):

            # Generate Gaussian noise to add to RV measurements
            noise = np.random.normal(meanErr,stddevErr,length)

            # Add noise to RV measurements
            #newRVs = np.add(self.fluxes,noise)

            # Calculate maximum power in the Lomb-Scargle periodogram
            maxPower = self.LS(35,45,1000,1,flux=noise)
            maxPowers.append(maxPower)
```

```
    # Check if maximum power exceeds that of period in non-noisy data
    if maxPower > original_maxPower:
        numExceed += 1

# Calculate FAP
FAP = numExceed/numIterations
print('FAP = {0:.3e}'.format(FAP))
return(FAP)
```