```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  2 12:55:40 2020

@author: jimmy
"""
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from ReadFile import ReadNASA
from astropy import units as u
from astropy import constants as const
from matplotlib.patches import Rectangle

# Class to plot grazing transits and compare to real data
class GrazingTransit:

    def __init__(self,star_type):

        # Set star type as global variable
        self.star_type = star_type

        # Define stellar host parameters
        if star_type == 'M':
            self.multiple = .559 # fractional size of Sun for M0V star
            self.star_temp = 3870.0
            self.host = 'M0V'
        elif star_type == 'K':
            self.multiple = 10.0 # fractional size of Sun for K0III star
            self.star_temp = 4810.0
            self.host = 'K0III'

        # Calculate radius of star in au
        self.rad_star = self.multiple*const.R_sun.to(u.au).value

        # Define min and max liquid water temperatures in K
        # Source (slide 3):https://www.astro.umd.edu/~miller/teaching/astr380f09/slides14.pdf
        self.TempWaterMin = 274.15
        self.TempWaterMax = 303.15

    # Function to calculate inclinations of minimally observable transits
    def Graze(self,rad_pl,a):
        # acos(i) = R_pl + R_star
        # Inputs:
        #   rad_pl: radius of planet (in au)
        #   rad_star: radius of host star (in au)
        #   a: arbitray semi-major axis
        # Returns:
        #   i: inclination

        # Maximum acos(i) value to observe transit
        threshold = rad_pl+self.rad_star

        # Calculate inclination if acos(i) = threshold (grazing condition)
        i = np.degrees(np.arccos(threshold/a))

        return(i)

    # Function to calculate habitable zone around star
    def HabitableZone(self):
        # Returns: min and max habitable zone distances (liquid water temps)

        # Calculate minimum and maximum habitable zone distances
        HZMax = self.rad_star/2*np.sqrt(1-0.3)*((self.star_temp/self.TempWaterMin)**2)
        HZMin = self.rad_star/2*np.sqrt(1-0.3)*((self.star_temp/self.TempWaterMax)**2)
```

```
        return(HZMin,HZMax)

    # Function to plot lines of minimally detectable transits
    def InclinationSemiMajor(self):

        # Define radii of Earth and Jupiter in au
        EarthRad = const.R_earth.to(u.au).value
        JupRad = const.R_jup.to(u.au).value
        increment = JupRad/EarthRad/5

        # Make array of planet radii from Earth's to Jupiter's
        PlRad = np.linspace(EarthRad,JupRad,5)

        # Make array of semi-major axes
        a = np.linspace(.01,10,2000) # in au

        # Generate array of inclinations
        inclinations = [self.Graze(PlRad[i],a) for i in range(len(PlRad))]
        inclinations = np.asarray(inclinations)

        # Create figure and axis object
        plt.figure(figsize=(10,6))
        ax = plt.subplot(111)

        # Plot grazing transit lines for each planet (with different linestyles)
        linestyles = ['-',':','--',':','-']
        for i in range(len(PlRad)):
            if i == 0:
                phrase = 'Earth-size'
            elif i == len(PlRad)-1:
                phrase = 'Jupiter-size'
            else:
                phrase = str(np.round(increment*i,2))+r'$R_{\oplus}$'
            ax.plot(a,inclinations[i],label=phrase,linestyle=linestyles[i])
        ax.set_xscale('log')

        # Read in data from NASA Exoplanet Archive
        ExoplanetData = ReadNASA('NASAExo.csv',skip=76)

        # Identifty locations of data points for each discovery method
        detections = np.where(ExoplanetData['pl_discmethod'] == 'Transit')

        # Define x (mass) and y (semi-major axis) datasets to plot
        x = ExoplanetData['pl_orbsmax'][detections]
        y = ExoplanetData['pl_orbincl'][detections]

        # Plot NASA Data
        ax.scatter(x,y,label='NASA Transits')
        ax.set_xlabel(r'log(a) (au)',fontsize=14)
        ax.set_ylabel('Inclination (deg)',fontsize=14)
        ax.set_title('Orbital Inclination vs. Semi-Major Axis \n (Host: {0})'.format(self.host),

        # Get limits of axes for adding plots
        ymin,ymax = ax.get_ylim()

        # Shade habitable zone
        minD, maxD = self.HabitableZone()
        ax.vlines(minD,ymin,ymax,label=r'$r_{hab,min} = %.2f au$'%minD,linestyle='--',color='bla
        ax.vlines(maxD,ymin,ymax,label=r'$r_{hab,max} = %.2f au$'%maxD,linestyle='--',color='bla
        ax.add_patch(Rectangle((minD,ymin),maxD-minD,ymax-ymin,fill=True,color='r',alpha=0.5))
        ax.legend(loc='center left',bbox_to_anchor=(1, 0.5))
        plt.tight_layout()
"""
# Lines to test function (comment out when not testing)
test = GrazingTransit('K')
test.InclinationSemiMajor()
"""
```