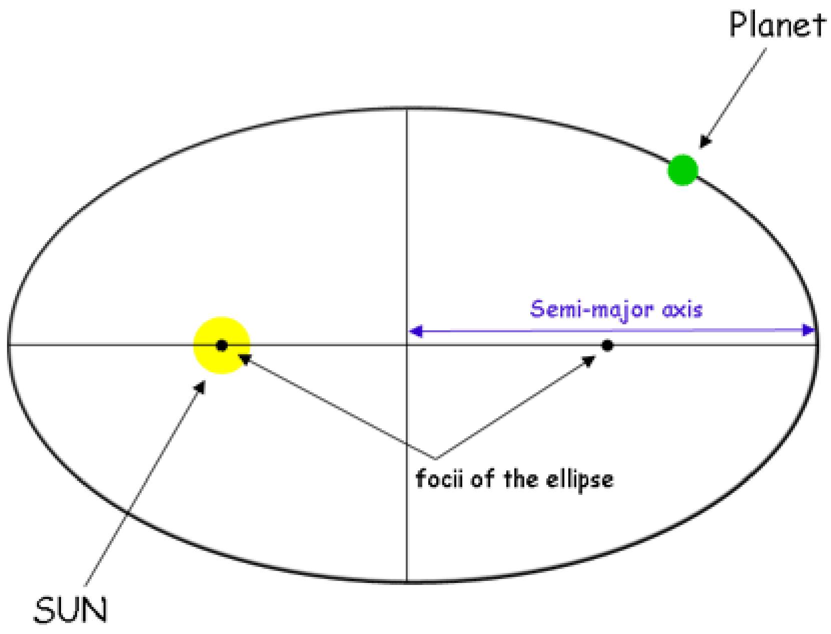
This is a jupyter notebook. These types of notebooks are useful for doing data analysis and making plots. We will do both here!

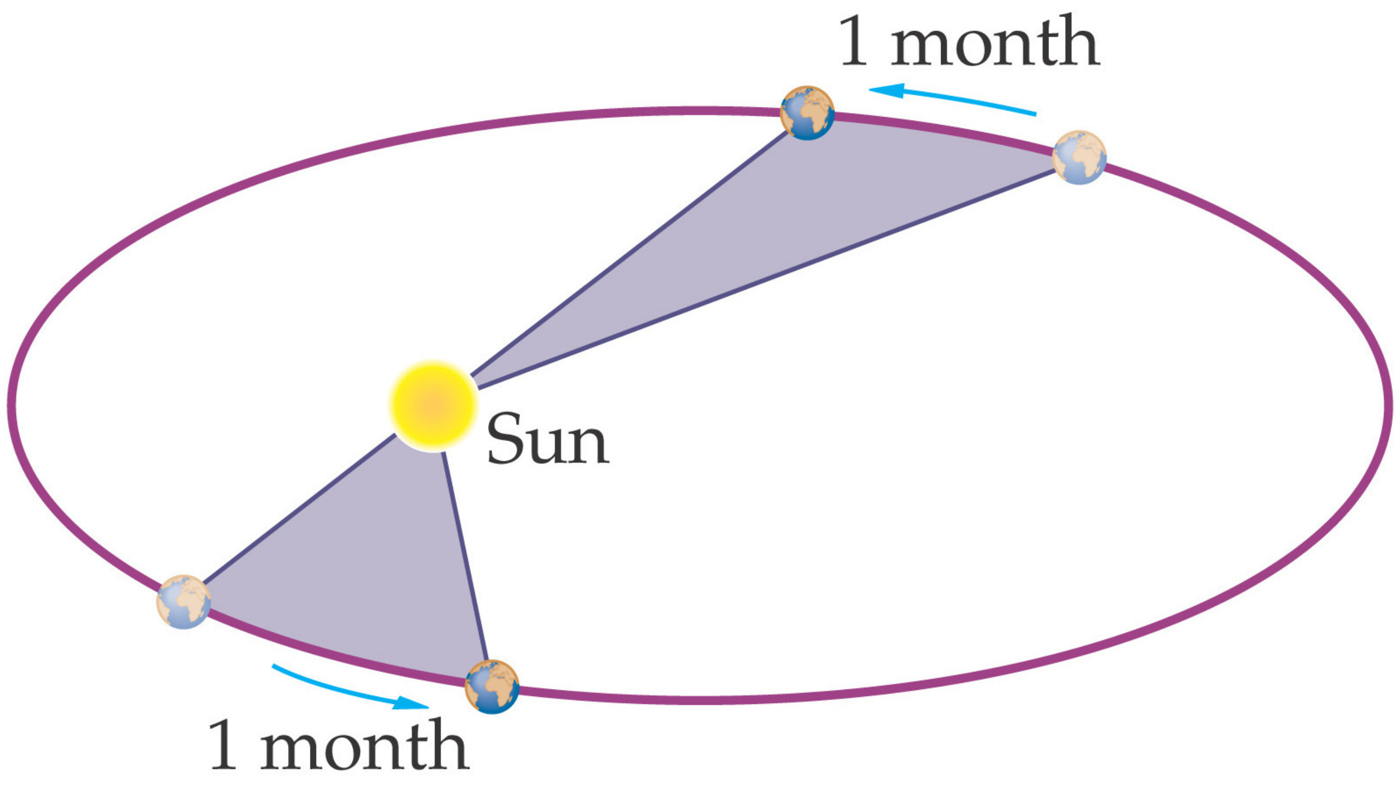
'Jupyter' is a mashup of Julia and python, two coding languages. We will be using python to simulate orbits and investigate how changing the mass, separation, and eccentricity changes the shape of the orbit.

Before we get going, let's review some stuff about orbits! Johannes Kepler taught the world how orbits work with three laws:

Kepler's Law #1: All planets move in elliptical orbits, with the sun at one focus. The place a planet resides at any time on it's orbit can be specified by the angle the planet makes from the semimajor axis. This angle is called the 'true anomaly'.



Kepler's Law #2: A line that connects a planet to the sun sweeps out equal areas in equal times.



Kepler's Law #3: The square of the period of any planet is proportional to the cube of the semimajor axis of its orbit.

$$T^2 = \frac{4\pi^2}{GM} a^3$$

can be expressed as simply

$$T^2 = a^3$$

If expressed in the following units:

T	Earth years
a	Astronomical units AU (a = 1 AU for Earth)
M	Solar masses $M_\odot$

Then $\frac{4\pi^2}{G} = 1$

Alright let's do some coding!!

Anytime you use a coding language, you need to let the computer know what packages you'll be using. In this notebook, we'd like to make our plots show up as we code them, so we tell the python plotting package, matplotlib, to plot 'inline'. We will also use numpy, which is short for 'numerical python'. Numpy is useful when using trigonometric functions like sin and cos.

In [1]:
```
% matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

It is always a good idea to define the physical constants you'll use in your code. In this case, we'll define the gravitational constant, G, and some mass, time, and distance units that will be useful.

In [2]:
```
# CONSTANTS
##########################################################################
######
G = 6.67384*10**(-11.0)
Msun = 1.9891*10**30.0
seconds_in_day = 86400.0
seconds_in_hour = 3600.0
seconds_in_year = 3.15569*10**7.0
day_in_year = 365.242
hours_in_day = 24.0
meters_in_AU = 1.496*10**11.0
pi = np.pi
```

We also need some functions that we'll use to generate orbit data. We'll write our own functions in the furture. For now, notice a few things about each function:

(1) the function is named something useful, so you know what it will do

(2) the function needs variables (the things inside the parentheses) to begin and returns variables at the end

(3) the function should tell you what the physical units are for each variable

We'll use these functions to generate orbits for several different scenarios. The first two functions use Kepler's third law to convert between separation and orbital period. The third equation solves for every point on an orbit; this can be used to graph the orbit!

In [3]:
```
def keplerIII_period_to_separation(m1,m2,p_orb):
    ##########################################################
    # Units: mass [kg], orbital period [s], separation [m] #
    ##########################################################
    sep_cubed = G*(m1+m2)/(4*pi**2)*p_orb**2
    sep = sep_cubed**(1./3.)

    return sep

def keplerIII_separation_to_period(m1,m2,sep):
    ##########################################################
    # Units: mass [kg], orbital period [s], separation [m] #
    ##########################################################
    p_orb_squared = (4*pi**2)/(G*(m1+m2))*sep**3
    p_orb = p_orb_squared**(1./2.)

    return p_orb


def make_kepler_orbit(m1,m2,e,p_orb,tmin,tmax):
    ####################################################################
###############
    # Units: mass [solar mass], orbital period [years], time [years], se
paration [au] #
    ####################################################################
###############
    m1 = m1
    m2 = m2
    tStep = p_orb/100
    nStep = int((tmax-tmin)/tStep)
    tRange = np.linspace(tmin,tmax,nStep)

    theta = []
    for time in tRange:
        nHalfPorb = int(2*(time-1)/p_orb)
        PsiDiff = 1
        M = 2*np.pi*time/p_orb
        PsiOld = M
        theta0old = 180.0
        while PsiDiff > 1e-10:
        #print PsiDiff, PsiOld, e*math.sin(PsiOld)
            PsiNew = M + e*np.sin(PsiOld)
            PsiDiff = PsiNew-PsiOld
            PsiOld = PsiNew
        theta0 = 2*np.arctan(((1+e)/(1-e))**(0.5)*np.tan(PsiOld/2.))
        theta.append(theta0)
    return theta
```

First, let's simulate the Earth's orbit! To do this, we need to specify all of the variables used in the 'make_kepler_orbit' function. Taking note of the units, let's fill in the mass, orbital period and eccentricity.

```
In [4]: mass_Sun = 1.0
        mass_Earth = 3*10**(-2.0)
        orbital_period = 1.0
        eccentricity = 0.02
```

Now we are ready to make our Kepler orbit. To do this, we need to send the right variables to the function and be ready for the variable the function sends back. make_kepler_orbit sends back a list of angles; these angles are different true anomaly angles for each time in the orbit.

We need to be sure to name the list that gets sent back from make_kepler_orbit makes sense. Choose a name that makes sense to you.

```
In [5]: list1 = make_kepler_orbit(mass_Sun,mass_Earth,eccentricity,orbital_perio
        d,0,1*orbital_period)
```

To make plots, we need to be able to specify the x and y position of each true anomaly value. This is done using the kepler shape equation:

$$r = \frac{a(1-e^2)}{1+e\cos\theta}$$

In the shape equation, theta is the true_anomaly.

The function below, orbit(sep,e,true_anomaly), computes the x and y values from the shape equation and returns them as xorbit and yorbit. We can use these x and y values to make a scatter plot of the orbit!

In [6]:
```
def orbit(sep,e,true_anomaly):
    ###############################################
    # Units: separation [same as units supplied] #
    ###############################################

    # define the shape equation
    rorbit = sep*(1 - e**2)/(1 + e*np.cos(true_anomaly))
    xorbit = rorbit*np.cos(true_anomaly)
    yorbit = rorbit*np.sin(true_anomaly)


    return xorbit,yorbit
```

Notice that the orbit function needs the separation. This means we'll need to use one of the Kepler's third law functions to convert the orbital period to separation.

Both function calls are listed below, choose which function we should used based on the conversion we need to make and fill in the variables. Be sure to rename the variable name to something that is easy to understand!

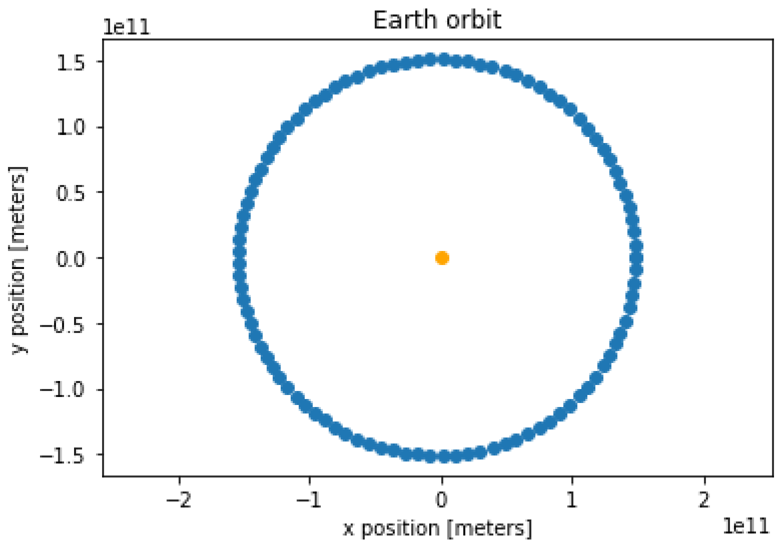
In [7]:
```
#variable = keplerIII_period_to_separation(variable_1, variable_2, varia
ble_3)
#variable = keplerIII_separation_to_period(variable_1, variable_2, varia
ble_3)

sep = keplerIII_period_to_separation(mass_Sun*Msun,mass_Earth*Msun,orbit
al_period*seconds_in_year)
```

Now that we have the separation, we can call the orbit function. The orbit function needs the separation, eccentricity and true anomaly to run. Remember that the true anomaly is returned by the make_kepler_orbit function.

In [8]:
```
#xOrbit, yOrbit = orbit(variable,e0,thetaTest)
xOrbit, yOrbit = orbit(sep,eccentricity,list1)
```

Thanks to the orbit function, we can now plot our xorbit and yorbit values. Notice that we set: plt.axis('equal'); this makes the plot use the same size for the x and y axes. Try commenting out the line that makes the axes equal by typing a '#' sign on the front part of the line. What happens to the orbit of the Earth? Why is this misleading? Fill in your answer in the box below the graph.

In [14]:
```
plt.title('Earth orbit')
# The Sun is located at the origin.
plt.scatter(0,0, color='orange')
plt.scatter(xOrbit,yOrbit)
plt.axis('equal')
plt.xlabel('x position [meters]')
plt.ylabel('y position [meters]')
plt.show()
```



The orbit looks eccentric. This is misrepresenting our data!

Now that we know how to make Earth's orbit, we can make orbits for any of the planets in the Solar System on a single plot! We'll need to know the mass of the planet in units of solar mass, its orbital period in years, and its eccentricity. This data is below for every planet other than Earth:

Planet = Mercury, Mass = 1.7 × 10^(-7) solar mass, orbital period = 0.48 years, eccentricity = 0.21

Planet = Venus, Mass = 2.6 × 10^(-6) solar mass, orbital period = 0.62 years, eccentricity = 0.01

Planet = Mars, Mass = 3.2 × 10^(-7) solar mass, orbital period = 1.88 years, eccentricity = 0.09

Planet = Jupiter, Mass = 9.5 × 10^(-4) solar mass, orbital period = 11.86 years, eccentricity = 0.05

Planet = Saturn, Mass = 2.9 × 10^(-4) solar mass, orbital period = 29.46 years, eccentricity = 0.05

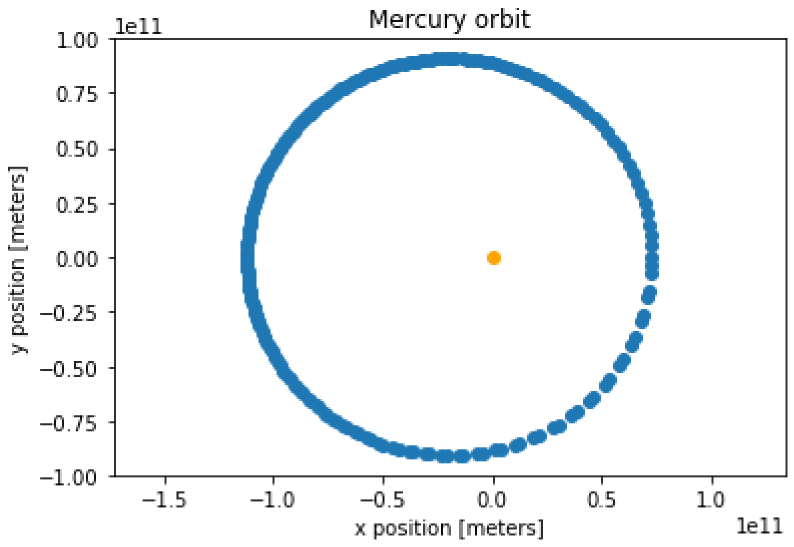
Planet = Uranus, Mass = 4.5 × 10^(-5) solar mass, orbital period =84.02 years, eccentricity = 0.05

Planet = Neptune, Mass = 5.2 × 10^(-5) solar mass, orbital period = 164.8 years, eccentricity = 0.01

Planet = Pluto, Mass = 6.6 × 10^(-9) solar mass, orbital period = 248.0 years, eccentricity = 0.25

Following the method in the next cell, pick three planets to plot and plot them in the empty cells below.

In [15]:
```
# Define Mercury's parameters;
#note we don't need to redefine the mass of the Sun since we already def
ined it above
mass_Mercury = 1.7*10**(-7.0)
orbital_period_Mercury = 0.48
eccentricity_Mercury = 0.21

# compute Mercury's true anomaly
true_anomaly_Mercury = make_kepler_orbit(mass_Mercury, mass_Sun,
                                         orbital_period_Mercury, eccentr
icity_Mercury,
                                         0,1*orbital_period_Mercury)

# compute Mercury's separation
separation_Mercury = keplerIII_period_to_separation(mass_Earth*Msun,mass
_Sun*Msun,
                                                    orbital_period_Mercu
ry*seconds_in_year)

# compute Mercury's x and y orbital coordinates
xOrbit_Mercury, yOrbit_Mercury = orbit(separation_Mercury,eccentricity_M
ercury,
                                       true_anomaly_Mercury)

plt.title('Mercury orbit')
# The Sun is located at the origin.
plt.scatter(0,0, color='orange')
plt.scatter(xOrbit_Mercury,yOrbit_Mercury)
plt.axis('equal')
plt.xlabel('x position [meters]')
plt.ylabel('y position [meters]')
plt.show()
```



In [ ]:

In [ ]:

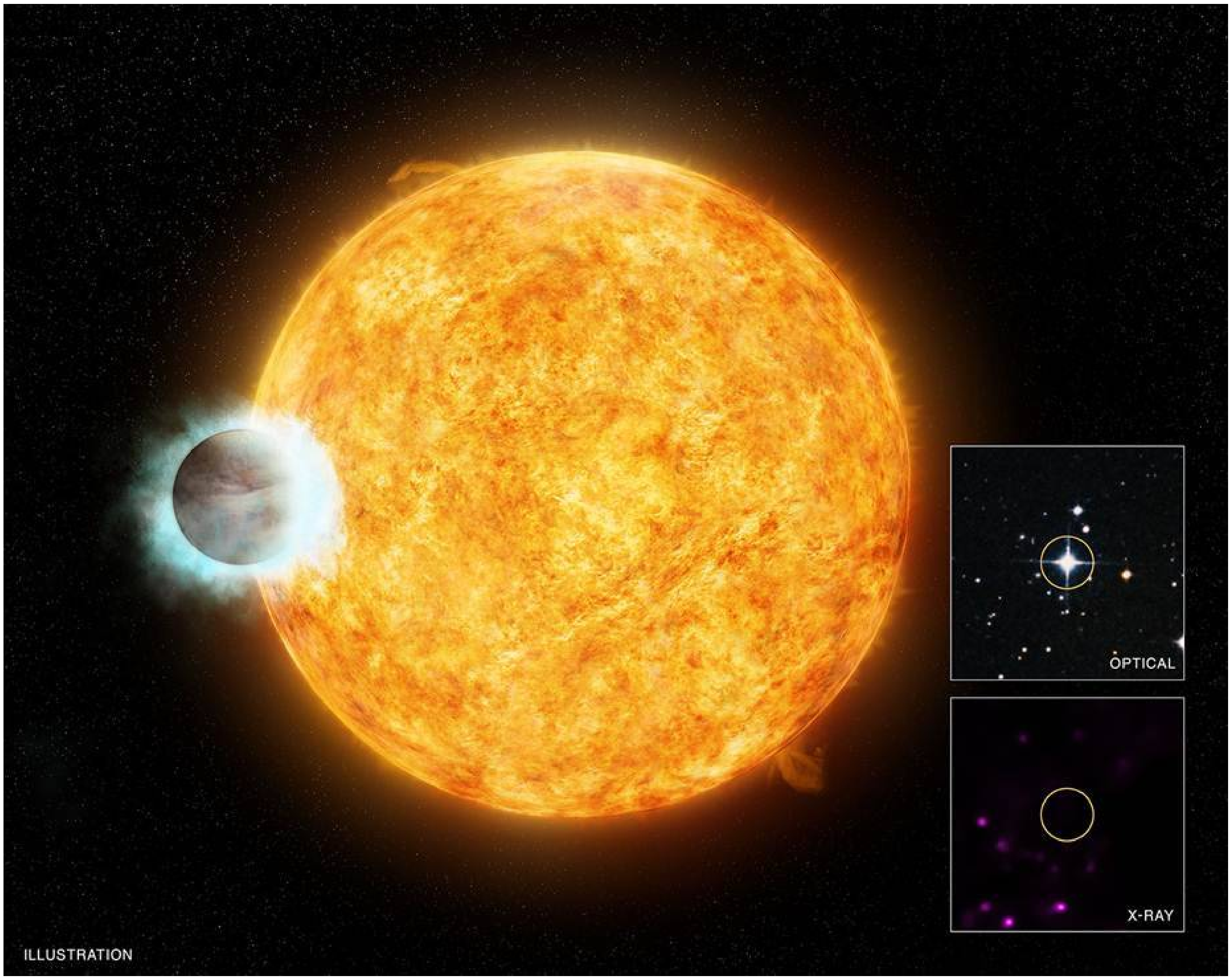
```
In [ ]:
```

Now try plotting every planet on the same plot in the cell below. Be sure to include axis labels and a plot title.

```
In [ ]:
```

In the cell below, type out which planet corresponds to which color on the plot

Now let's try comparing the solar system planets to planets orbiting other stars, called exoplanets. Since Kepler's laws work for ANY orbit, we can use the same code to make plots for any kind of exoplanet. One kind of exoplanet is called a hot Jupiter. Below is an artist picture of Wasp 18b:



Here is Wasp 18b's data:

Planet = Wasp-18b, Star Mass = 1.0 solar mass, Planet Mass = 9.5 × 10^(-3) solar mass, orbital period = 0.24 years, eccentricity = 0.01

Based on the data, how would you compare Wasp 18b to Jupiter? Fill your answer in the cell below