NOTE: This is the solutions document. All code has been filled in, all lines that were meant to be uncommented have been uncommented, and you can run every cell as is. Remember to still run every cell in order!

Section 1: Beginning Tasks - Diagnostics

Welcome to mission control! We're going to start off by performing a few diagnostic tasks to make sure all of our systems are functioning, and along the way we will be learning the basics of coding.

Much like spoken languages, different programming languages have different syntax and sematics. Syntax refers to the set of rules, principles, and processes that govern the structure of a given language. Semantics refers to the meaning of a word, phrase, sentence, or text.

An interface is any device or program that allows a user to communicate with a computer. The interface we are working in right now, Jupyter notebooks, is based on a programming language called Python. In a Jupyter notebook, code is written within cells, which can be executed separately. As per the setup instructions, you should have already activated line numbers, which show up on the left hand side of the cell below; these are handy and allow for easy reference to parts of your code. We will proceed cell by cell.

Run the cell below by clicking within the cell (you will see a box show up around the cell), and then pressing the run button located on the toolbar at the top. Note that when you run a cell, the program is going through each line of code and executing the lines in order.

In [1]:
```
## *** CELL 1.1 *** ##

import numpy as np
import astropy as ap
import math
from matplotlib import pyplot as plt
import HelperFunctions as h
```

When you ran the cell above, it might have seemed as though nothing happened. That is because the code within that cell is working behind the scenes importing external content that will allow you to do cool stuff later on! You don't need to know the specifics of that right now, so let's move on to some coding basics.

Read through the cell below and then run it. This time, you will see an output below the cell.

In [2]:
```
## *** CELL 1.2 *** ##

# You'll notice this text is a different color. This is a comment. Comments are started with the # sign.
# Comments do not execute within your code, and have no output.
# Comments are used to make notes throughout your code.
# You can also comment out lines of code that you don't want to run by adding the # symbol at the start of the line.

# The print() function (shown below on line 11) prints the input to the screen.
# Below our input is the sentence "Hello world!"

print("Hello world!")

# You'll notice that "Hello world!" is highlighted red above.
# This is because the double quotes describe a string.

# Strings are a data type that represent characters.
# Examples of strings include "Let's go to mars!", "abcde", and "12345". Just remember the double quotes.

# After you run this cell, uncomment the line below by deleting the # at the start of line 20, then run it again.
print("Coding is fun!")

# Feel free to change the string content in line 11 or line 20!
```

```
Hello world!
Coding is fun!
```

Now we are going to introduce the concept of a variable. Variables are a very important part of coding. A variable is a storage location that is represented by a symbolic name.

The variable name needs to start with a letter, can only contain alpha-numeric characters, and is case sensitive. Variable names can be 1 letter long, or whole words! Underscores are also useful for ease of reading the variable name. Read through the cell below.

In [3]:
```
## *** CELL 1.3  *** ##
## *** INITIALIZATION *** ##
############################

# Variables often contain numeric values (integers - another data type!).
# You assign values to variables using the equals sign. For example...

x = 2
y = 6

# We can add our two variables together using the plus sign.
our_sum = x + y

# You can also divide with the / symbol, and multiply using the * symbol, as shown below:
our_quotient = y/x
our_product = x*y

# If you would like to use an exponent, you would use '**' to represent the exponent, as shown below:
our_exponent = y**x

# Subtraction is done using the minus sign.
# Can you complete the line of code (line 25) that calculates the difference between our two variables?
# Once you have replaced the <<INSERT CODE HERE>> section with your code, run the cell.

our_difference = y - x

# If you run this cell as is, you will print the sum of the two variables we defined.
# You can change the variable being printed by editing line 30 (change the variable input), and running the cell again.

print(our_sum)
```

```
8
```

Alright, it seems our system is initialized. Next up is checking the temperature of the systems board.

In [4]:
```
## *** CELL 1.4  *** ##
## *** TEMPERATURE OF SYSTEMS BOARD *** ##
##########################################

# Here we are defining two variables that represent our systems board temperature, and the target temperature.
Temp_SystemsBoard = 40   # degrees Celsius
Temp_Target = 40         # degrees Celsius

# Let's compare the temperature of our systems board (Temp_SystemsBoard) to our target temperature (Temp_Target).
# Lines 15-31 are an if/else conditional statement, which is a very important concept in coding.
# We use if/else statements when we want to execute code only if a certain condition is satisfied.

# The 'if' command starts our if/else statement. We have to define a condition.
# The next line will check if the value of the variable Temp_SystemsBoard is greater than the value of Temp_Target.
if Temp_SystemsBoard > Temp_Target:

    # If the above statement is true, the following line of code will execute. If it's not true, nothing will happen,
    # and we will move on to our next line of code, line 23.
    print("The systems board is too hot.")

    # The command "elif" is short form for else if. For this command we have to define another condition.
    # The next line will check if the value of Temp_SystemsBoard is less than Temp_Target.
elif Temp_SystemsBoard < Temp_Target:

    # If this is true, the following line of code will execute. Otherwise, we will move on to our next line of code.
    print("The systems board is too cold.")

    # Finally, if the two previous conditions did not execute, meaning that Temp_SystemsBoard is neither greater,
    # nor less than Temp_Target, the following code will execute. For the else command, no condition is defined.
else:
    print("Systems board is ready to go!")

# Try changing the value of Temp_SystemsBoard (line 6) and see how that affects the output of this cell when you run it.
# What happens when the two temperatures have equal values?
```

```
Systems board is ready to go!
```

Did you get a message that the systems board is ready to go? If yes, great! So far you have learned what a string is, how to define variables, and how to print strings and variables to your screen! You have even gone through an if/else statement.

We'll continue checking our systems are working properly.

In [5]:
```
## *** CELL 1.5  *** ##
## *** SPEED OF SPACECRAFT *** ##
##################################

# We want to calculate the required speed of our spacecraft for a given distance and time.
# Assign values to the two variables below on line 9 and 12.

# How far are we travelling?
Distance_Travelled = 450000 # kilometres

# How long do you want it to take?
Time_Elapsed = 32 # hours

# Now we calculate our speed.
Speed_Spacecraft = Distance_Travelled/Time_Elapsed # kilometres/hour

# Let's convert our speed to metres/seconds, and round the final value to the nearest integer.
Speed_Spacecraft = round(Speed_Spacecraft*1000/(60*60))

# We would like to print our variables to make sure the entire mission crew is aware of the distance, time and speed.
# Recall the print function, and while we're at it, let's make it a full sentence!

# First we need to convert our numeric values to strings.
Distance_Travelled = str(Distance_Travelled)
Time_Elapsed = str(Time_Elapsed)
Speed_Spacecraft = str(Speed_Spacecraft)

print("We are travelling " + Distance_Travelled + " km, and it should take " + Time_Elapsed + " hours")

# What we have done in line 28 is concatenate (link together multiple strings).
# The plus sign is used for concatenation.
# Some of these strings we wrote outright in line 28, while other strings we stored in our variables.

# You can only concatenate the same data type,
# which is why we converted our three numeric variables to strings in lines 24-26.

# Can you write a line of code below that prints 'The speed of the spacecraft needs to be __ m/s.' to the screen?
# Hint 1: Remember to use double quotes to define a string.
# Hint 2: Our Speed_Spacecraft variable has already been converted to a string.

print("The speed of the spacecraft needs to be " + Speed_Spacecraft + " m/s")
```

```
We are travelling 450000 km, and it should take 32 hours
The speed of the spacecraft needs to be 3906 m/s
```

You have completed the diagnostics! If you find you have finished this section early, you can take a peek at the advanced part of Section 1 below. Otherwise we will be moving on to Section 2 as a group. After the workshop feel free to look through the advanced sections on your own!

ADVANCED: the following five cells introduce several advanced topics that are not explicitly needed for this workshop, but are very important for coding!

In [6]:
```
## *** Pilot Check *** ##
#########################

# We will now be checking our list of pilots.
# To do this, we will be introducing 'for' loops. These can be incredibly powerful but also dangerous!

# The following code shows how we can use 'for' loops:
for i in range(5):
    num = str(i)
    print("i = " + num)

# Run this cell and see what is output:
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
```

In [7]:
```
# As we can see, the 'print' was called 5 times, but each time the 'i' value changed.
# This allows us to do repeated computations with changing values!

# Next, let's look at arrays and lists. A list is exactly what it sounds like, a list of values!
Pilots = ["Akanksha", "Sonja", "Yilda", "Olivera", "Simran", "Jasmine"]

# We see there are currently 6 pilots, so we define a variable for this number
pilot_num = 6

# We can look into this list entry by entry. Let's look at the first entry:
print(Pilots[1])
```

```
Sonja
```

In [8]:
```
# Wait, why do we see Sonja but not Akanksha? As you may have noticed when we introduced the 'for' loop,
# referencing in Python starts at 0. So, the first entry is actually the zeroth entry! Weird, but let's test it:
print(Pilots[0])

# Run the cell!
```

```
Akanksha
```

In [9]:
```
# Okay, now that we know how that works, let's check in on the pilots:
for i in range(pilot_num):
    print(Pilots[i] + " is ready to train for launch!")

# Run the cell and see what they think!
```

```
Akanksha is ready to train for launch!
Sonja is ready to train for launch!
Yilda is ready to train for launch!
Olivera is ready to train for launch!
Simran is ready to train for launch!
Jasmine is ready to train for launch!
```

In [10]:
```
# Try adding yourself to the Pilots list and modify the for loop accordingly:

# Pilot info
pilot_num = 7
Pilots = ["Akanksha", "Sonja", "Yilda", "Olivera", "Simran", "Jasmine", "One More"]

for i in range(pilot_num):
    print(Pilots[i] + " is ready to train for launch!")

# Hint: Remember that adding yourself to the list increases the number of pilots!
```

```
Akanksha is ready to train for launch!
Sonja is ready to train for launch!
Yilda is ready to train for launch!
Olivera is ready to train for launch!
Simran is ready to train for launch!
Jasmine is ready to train for launch!
One More is ready to train for launch!
```

Section 2: Preparation

How does the Earth orbit the Sun?

Johannes Kepler discovered that the planets orbit the Sun in ellipses! (with the Sun at the "focus" of the ellipse). This is known as Kepler's first law.

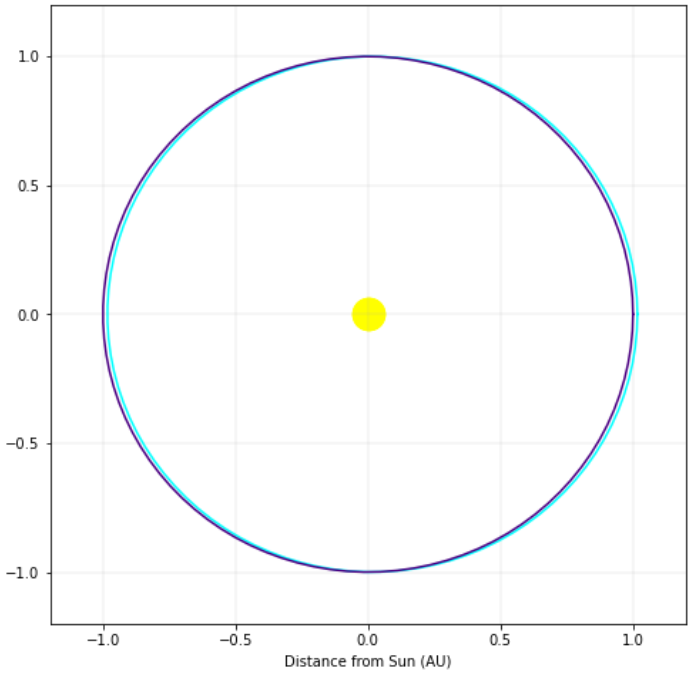
Ellipses are like squashed circles -> the amount they are "squashed" is given by their eccentricity, e. (Where e is a number between 0 and 1). The eccentricity of the Earth's orbit is $e = 0.0167$.

The Earth's distance from the sun is 1AU (Astronomical units - these are the units we use to measure distance in our solar system).

*Note we have ignored 'inclination' (the slant of the orbit relative to the horizontal) of the orbits for simplicity.

Now we will plot Earth's orbit:

In [11]:
```
## *** CELL 2.1  *** ##
#1) First Run the cell to see Earth's orbit!

# This parameter is intialitizing the plot axis limits (see 'HelperFunctions.py' file to see how it's used)
max_major=0
# Here we are calling to the python plotting library 'matplotlib' to intialize an empty figure of size 8x8
fig, ax = plt.subplots(figsize=[8,8])

# Now we will call on helper code from our HelperFunctions, 'h' library.

# 2) Try playing around with the 'e' parameter. REMEMBER: e has to be a number between 0 and 1
# For example: you can try e=0.2, 0.4, 0.6, 0.8, 0.999, 1
h.plot_orbit(major=1, e=0.0167, color='cyan', ax=ax, max_major=max_major)
# What value of e gives a perfect circle? What about a line?

# 3) Try playing around with the 'major' parameter
# 4) Uncomment the line below. For example: you can try major=0, 0.5, 1, 5, 10
h.plot_orbit(major=1, e=0, color='indigo', ax=ax, max_major=max_major)
# What does the parameter 'major' control? (Hint: make sure to look at the axis values when you change major)
```



True or False:

1) The Earth's orbit is almost a circle

2) During summer, the Earth is 3 AU away from the sun

What about the other planets?

The distance and eccentricity for all planets are shown below:

We have also included the time it takes each planet to orbit the sun (orbital period) in days. This will be useful in later sections.

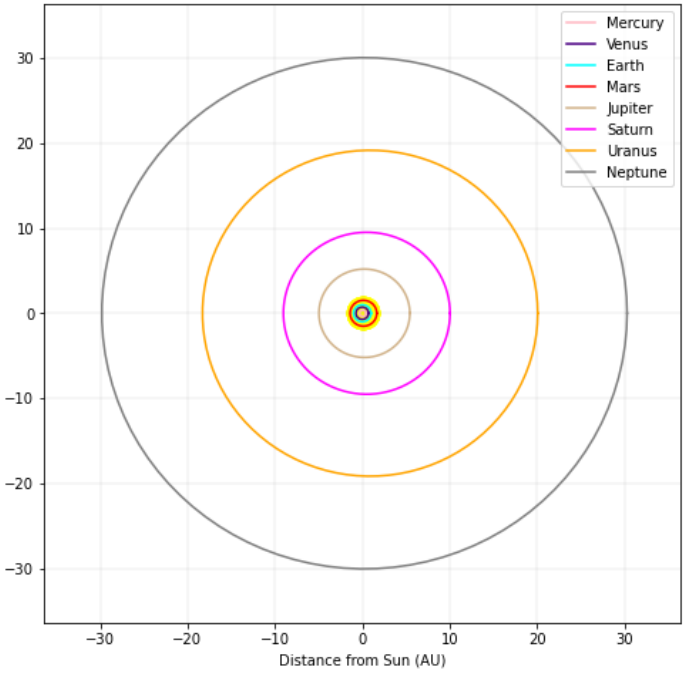
Planet	Distance (AU)	Eccentricity (e)	Orbital Period (days)
Mercury	0.387	0.2056	88
Venus	0.723	0.0068	225
Earth	1.000	0.0167	365.24
Mars	1.524	0.0934	686.68
Jupiter	5.203	0.0484	4380 (12 years)
Saturn	9.537	0.0542	10585 (29 years)
Uranus	19.191	0.0472	30660 (84 years)

Planet	Distance (AU)	Eccentricity (e)	Orbital Period (days)
Neptune	30.069	0.0086	60225 (165 years)

*NOTE - The further away the planet, the longer its period. This is known as Kepler's Third Law!

In [12]:
```
## *** CELL 2.2  *** ##
# Now we will succesively plot all the planet's orbits in order.
# This is done to scale so you can see how far Mercury and Neptune are!

# We want a new figure so we have to re-intialize our plotting parameters as before
max_major=0
fig, ax = plt.subplots(figsize=[8,8])

# Plot Mercury: Example given
h.plot_orbit(major=0.387, e=0.2056, color='lightpink', ax=ax, max_major=max_major, label='Mercury')
# 1) Run the cell to see Mercury's orbit!

# 2) Plot Venus: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=0.723, e=0.0068, color='indigo', ax=ax, max_major=max_major, label='Venus')
# 3) Run the cell to add Venus's orbit. How does it compare to Mercury's?

# 4) Plot Earth: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=1, e=0.0167, color='cyan', ax=ax, max_major=max_major, label='Earth')
# 5) Run the cell to add Earth's orbit. How does it compare to others?

# 6) Plot Mars: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=1.524, e=0.0934, color='red', ax=ax, max_major=max_major, label='Mars')
# 7) Run the cell to add Mars' orbit. How does it compare to others?

# 8) Plot Jupiter: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=5.203, e=0.0484, color='tan', ax=ax, max_major=max_major, label='Jupiter')
# 9) Run the cell to add Jupiter's orbit. How does it compare to others?

# 10) Plot Saturn: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=9.537, e=0.0542, color='fuchsia', ax=ax, max_major=max_major, label='Saturn')
# 11) Run the cell to add Saturn's orbit. How does it compare to others?

# 12) Plot Uranus: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=19.191, e=0.0472, color='orange', ax=ax, max_major=max_major, label='Uranus')
# 13) Run the cell to add Uranus' orbit. How does it compare to others?

# 14) Plot Neptune: Uncomment line below fill in the major and e values using table above
h.plot_orbit(major=30.069, e=0.0086, color='gray', ax=ax, max_major=max_major, label='Neptune')
# 15) Run the cell to add Neptune's orbit. How does it compare to others?

#adding the plot legend
plt.legend(loc='upper right')
```

Out[12]:
```
<matplotlib.legend.Legend at 0x7f11b07ba650>
```



Now let's move on to our final section.

Section 3: Final Mission - Going to Mars!

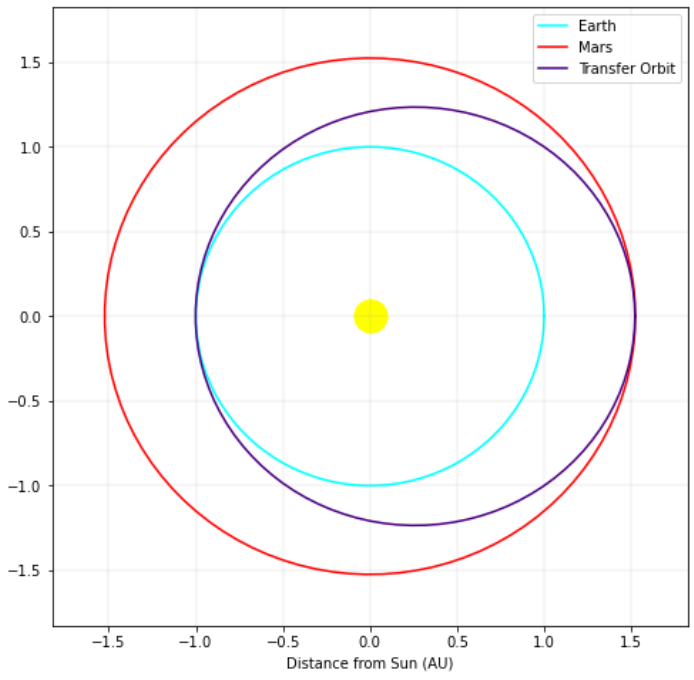
In order to get to Mars, we need to figure out how to leave Earth's orbit and arrive at Mars' orbit. Let's first do some trajectory planning.

Part 1: Our Path To Mars

Run the code below to plot our ship's trajectory to Mars' orbit.

In [13]:
```
## *** CELL 3.1  *** ##

# Here we are defining subplots for plotting the planets' orbits.
fig, ax = plt.subplots(figsize=[8,8])

# Plot the Earth's orbit
h.plot_orbit(major=1, e=0, color="cyan", ax=ax, label='Earth')
# Plot Mars' orbit
h.plot_orbit(major=1.524, e=0, color="red", ax=ax, label='Mars') #mars

# The function below allows us to plot our trajectory to Mars.
# It returns two values that allow us to plot an ellipse:
#   - a, the length of the semi-major axis (the distance from the Sun to the furthest
#     point on the ellipse)
#   - the eccentricity of the ellipse (which we learned about during our preparation).
#     as a reminder, it is the amount the ellipse is "squashed" compared to a circle

def plot_trajectory():
    R_Mars = 1.524 # the distance between Mars and the Sun, in AU
    R_Earth = 1 # the distance between Earth and the Sun, in AU

    a = (R_Mars + R_Earth)/2 # calculate the length of the semi-major axis
    focus = 0.26 # the location of the focus of the ellipse (the other focus is the Sun)
    eccentricity = focus/a # the eccentricity

    return a, eccentricity

a, eccentricity = plot_trajectory()

h.plot_orbit(major=a, e=eccentricity, color="indigo", ax=ax, label='Transfer Orbit')
plt.legend()
```

Out[13]: <matplotlib.legend.Legend at 0x7f11b0743050>



ADVANCED: consider the "plot_trajectory" function. This is a function that allows us to draw the ellipse of our trajectory by determining

the focal length and semi-major axis length. Can you figure out how we can find the focal length and semi-major axis length of the new ellipse, knowing what we already know about Earth's and Mars' orbits?

ADVANCED ANSWER The value of the focus point for our solar system is 0.26 AU - this is an accepted value that can be found online.

The length of the semi-major axis is half of the distance between Earth and Mars when they are most separated. This is because our elliptical orbit goes through both these locations, and they are the points on our ellipse that are the furthest from each other. Then we can calculate:

$2a = \text{distance between Earth and Mars} = \text{Earth-Sun distance} + \text{Mars-Sun distance}$

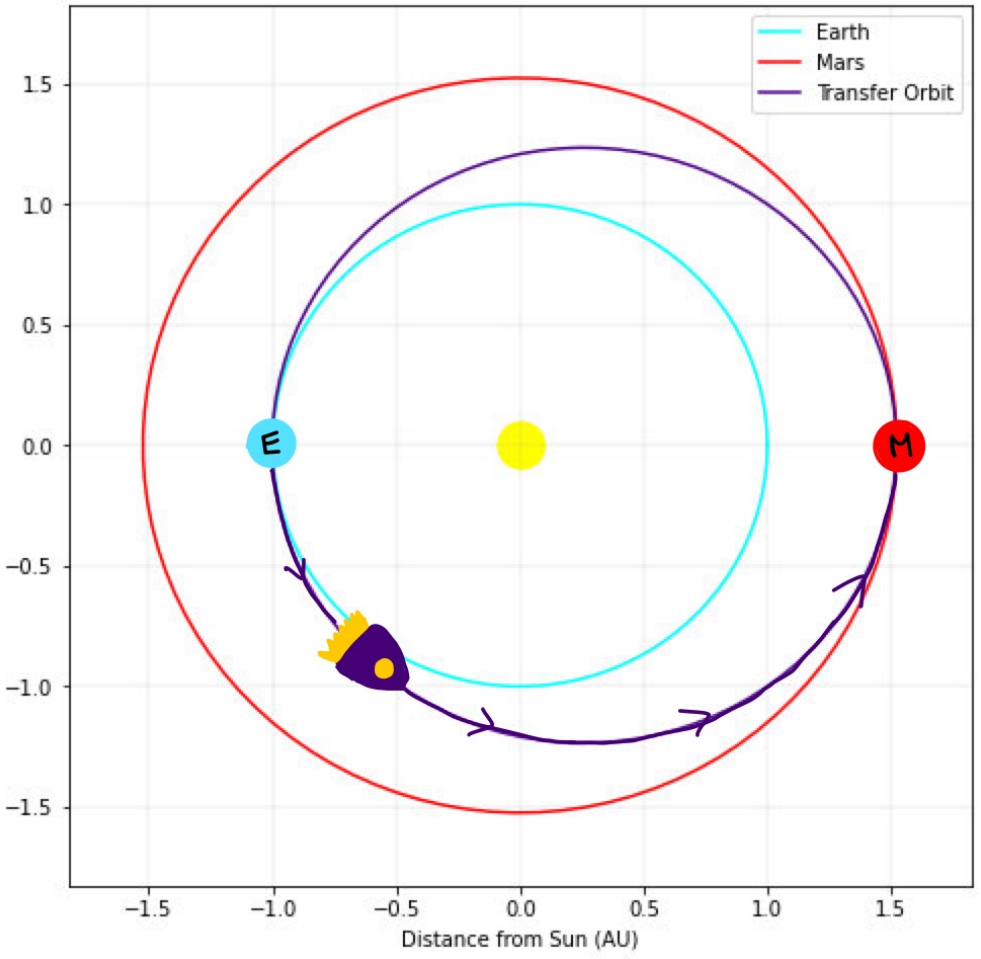
$2a = r_{earth} + r_{mars} = 1.524 + 1(AU) = 2.524AU$

$a = 1.262AU$

The animations in this link illustrate this quite well (Source: NASA JPL (https://www.jpl.nasa.gov/edu/teach/activity/lets-go-to-mars-calculating-launch-windows/))

From our trajectory plan above, we can see that our rocket will start in Earth's orbit, exit Earth's orbit, reach Mars' orbit and remain in that orbit. This path follows an elliptical shape - it is called a "Hohmann Transfer Orbit".

The drawing below illustrates this trajectory:



You might be asking yourself - why do we need to go in an ellipse? Why can't we go in a straight line to Mars? The reason for this is the gravitational pull from the Sun. Going in a straight line is less energy-efficient than more elliptical paths. (This is why planets orbit in ellipses).

It turns out that this ellipse is the most efficient path - it allows us to use the Sun's gravitational pull and not fight against it. Now, let's figure out how to launch our spaceship so that it follows this path!

Leaving Earth's atmosphere requires a certain speed, or we will not be able to escape Earth's gravity. In order to leave Earth's atmosphere and get to Mars, we need to go even faster!

We need to determine the speed we need to leave Earth's atmosphere and start our elliptical trajectory. We also need to determine the speed that we need to slow down to, once we arrive at Mars' orbit - otherwise, we will be launched too far out into space!

Fill in the information below from the table in Section 2, then run the next two cells to calculate these changes in velocity.

In [14]:
```
## *** CELL 3.2  *** ##

# Fill in the distance between the Earth and the Sun, in AU:
earth_sun_distance = 1

# Fill in the distance between Mars and the Sun, in AU:
mars_sun_distance = 1.524

# What is the period of the Earth's orbit around the Sun, in days?
P_earth = 365.24

# What is the period of Mars's orbit around the Sun, in days?
P_mars = 686.68
```

In [15]:
```
## *** CELL 3.3  *** ##

# This cell will calculate the speeds! There is a lot going on here, but it
# is all explained thoroughly in the second "Advanced Section" after this cell,
# so don't worry for now. You can run this cell as is.

# Define important values
G = 6.67*10**(-11)      # the gravitational constant
M_sun = 1.989*10**(30)  # the mass of the Sun

# We define the following function to calculate the necessary speeds:
def Hohmann_velocities(earth_sun_distance, mars_sun_distance, P_earth, P_mars, M_sun):
  # First we define some useful constants to use in our calculations:
    P_1 = P_earth*86400 # the period of the Earth, in seconds
    P_2 = P_mars*86400 # the period of Mars, in seconds
    r_1 = earth_sun_distance*1.496*10**(11) # the Earth-Sun distance, in m
    r_2 = mars_sun_distance *1.496*10**(11) # the Mars-Sun distance, in m

  # Next we calculate the semi-major axis of our transfer orbit ellipse:
    a = (r_1 + r_2)/2

  # We use this value as well as M_sun, G to calculate the period of our transfer orbit from Kepler's 3rd Law
    P_transfer = np.sqrt(4*(math.pi)**2*a**3/(G*M_sun))

  # Now we need to find the speed our spaceship would have if it were in Earth's orbit and in Mars' orbit.
    v_earth = 2*math.pi*r_1/P_1
    v_mars = 2*math.pi*r_2/P_2

  # We also find the speed of our spaceship at the points in our transfer orbit that intersect Earth and Mars.
    v_perihelion = 2*math.pi*a/P_transfer * np.sqrt((2*a/r_1)-1) #this intersects Earth's orbit

    v_aphelion = 2*math.pi*a/P_transfer * np.sqrt((2*a/r_2)-1) # this intersects Mars' orbit

  # Finally we find the difference between the speeds to determine by how much we need to slow down and speed up at each
    delta_v_perihelion = v_perihelion - v_earth

    delta_v_aphelion = v_mars - v_perihelion

    print("The amount we need to speed up by to leave Earth's orbit is", round(delta_v_perihelion, 1), "m/s.")

    print("The amount we need to slow down by to stay in Mars' orbit is", round(-1*delta_v_aphelion, 1), "m/s.")

# We can now call our defined function using our values from Cell 3.2
Hohmann_velocities(earth_sun_distance, mars_sun_distance, P_earth, P_mars, M_sun)
```

```
The amount we need to speed up by to leave Earth's orbit is 2938.3 m/s.
The amount we need to slow down by to stay in Mars' orbit is 8579.8 m/s.
```

ADVANCED: These equations will work for any planet further out than Earth! Change up the parameters to see where else you can travel. Do you think it will require a greater speed to reach Jupiter, or a slower speed?

It will require a greater speed! This is because the distance to be travelled is larger, so our ellipse must also be larger. The faster we go, the larger our elliptical path can be.

ADVANCED: You can write out your own code to calculate these velocities, or solve it by hand. Here are some hints if you would like to try it out:

1. First, you must figure out the speed of Earth in its orbit. This will be the initial speed of your spaceship, because it is in Earth's atmosphere. Remember circular motion, and the time it takes for Earth to go around the Sun!

Speed is distance over time, so the speed of Earth as it goes around its circular orbit is $v_{earth} = \frac{2\pi r_{earth}}{P_{earth}}$.

2. Then, you must figure out the speed the spaceship should have, if it were in this location but following the *indigo* transfer orbit. This location is called the "perihelion." It is the point on the ellipse that is closest to the Sun. To do this, you must use Kepler's Laws and conservation of energy. You may also need the fact that the total energy of an object orbiting in an elliptical path is is $E_{tot} = -\frac{GM_{sun}M_{earth}}{2a}$.

The period of our transfer orbit is, using Kepler's Third Law: $P = \sqrt{\frac{4\pi a^3}{GM}}$.

From conservation of energy, the total energy of the Earth is equal to its potential energy plus its kinetic energy. Thus, $E_{tot} = \frac{M_{earth}v^2}{2} - \frac{GM_{sun}M_{earth}}{r_{earth}}$.

From a theorem, the total energy of Earth is also equal to half of its potential energy at the semi-major axis distance a. So, $E_{tot} = -\frac{GM_{sun}M_{earth}}{2a}$.

We can set the two equations for E_{tot} equal to each other and solve for v. We will then get:

$$v_{perihelion} = \sqrt{GM}\sqrt{\frac{2}{r_{earth}} - \frac{1}{a}}$$

We can solve this right away, but we can also use our value of P to do so. From our equation for P, we can write $GM = \frac{4\pi^2 a^3}{P^2}$, and so we can finally get:

$$v_{perihelion} = \frac{2\pi a}{P_{transfer}}\sqrt{\frac{2a}{r_{earth}} - 1}$$

3. The difference between the two speeds will be the amount by which you need to speed up to enter the indigo orbit!

$$\Delta v = v_{perihelion} - v_{earth}$$

4. At the other side of the indigo transfer orbit, you will need to slow down in order to stay in Mars' orbit. This point is called the "aphelion" - the point furthest from the Sun. Once again, use Kepler's Laws to determine the speed your spaceship will have at this point.

This is the same procedure as in step 2, except we are using Mars' data. We get:

$$v_{aphelion} = \frac{2\pi a}{P_{transfer}}\sqrt{\frac{2a}{r_{mars}} - 1}$$

5. Like before, determine the speed your spaceship would have if it were in Mars' orbit - using circular motion again!

This is the same procedure as in step 1, except we are using Mars' data. We get:

$$v_{earth} = \frac{2\pi r_{earth}}{P_{earth}}$$

6. The difference between these two speeds is the amount by which you would need to slow down to get on Mars' orbit.

$$\Delta v = v_{mars} - v_{aphelion}$$

Part 2: Arriving On Time!

Great! We have figured out how to get to Mars' orbit with our spaceship! However, we still need to make sure that we will intersect Mars once we arrive - otherwise we will have an unpleasant surprise upon arriving...

To do this, we need to plan ahead and determine how long it will take us to complete our journey to Mars' orbit. We can use Kepler's Laws to do this. Kepler's Third Law is:

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

- **P** is the period of the transfer - the total amount of time it would take to go all the way around the green ellipse of our trajectory.
- **a** is the semi-major axis length of our elliptical trajectory
- **M** is the mass of the Sun
- **G** is the gravitational constant

Remember that we are not looking for the entire period P, because we are only following this path to get to Mars, and not to come back. Our travel time will be only a percentage of the period: time = (decimal percentage)*P.

Fill in the value of the decimal percentage below, and the number of seconds in a day, and then run the cell to calculate the time it will take to reach Mars, in seconds and in days.

Hint: look back at our trajectory diagram to help determine the percentage!

In [16]:
```
## *** CELL 3.4  *** ##

# Fill in the following two values:
decimal_percentage = 0.5
seconds_per_day = 86400

# This is the equation for the period of our transfer orbit, from Kepler's Third Law
P = np.sqrt(4*(math.pi)**2*(a*1.496*10**(11))**3/(G*M_sun))

# Now we can calculate the time it takes us to reach Mars by following this orbit.
time = decimal_percentage*P

# We will convert this time from seconds to days
time_in_days = time/seconds_per_day

print("It would take", round(time, 1), "seconds to reach Mars using this trajectory. \n This is equivalent to", round(ti
```

```
It would take 22374636.4 seconds to reach Mars using this trajectory.
 This is equivalent to 259.0 days.
```

We now know how long it will take us to reach Mars on this journey. How can we use this to plan our departure date from Earth?

Think about this, then enter your plan in the text box below for your fellow astronauts.

We know that it will take us 259 days to travel to Mars along our elliptical orbit. We must choose a departure time such that, after 259 days, Mars will be at the opposite end of the elliptical orbit from where we launched our ship. This sounds daunting, but with observations of Mars' orbits, and some calculations, our engineers will be able to pinpoint an exact date!

In [ ]: