

Projects

1) A 2D Model of the Cosmic Voyage estimator.

● Introduction:

This project explores the fundamental principles of planetary motion, specifically Kepler's Laws and circular motion, by creating a simplified 2D model of the Solar System using Python. The program calculates the positions of planets based on their orbital periods and distances from the Sun, visualizing their orbits and relative positions.

● Objectives:

• Develop a program in Python .

• Simulate the positions of planets in the Solar System.

• Visualize the orbits and positions of the planets in a 2D plot.

• Calculate hypothetical travel distances and times between Earth and other planets.

• Analyze and interpret the results of the simulation.

• Kepler's Laws of Planetary Motion (focus on the first and second laws).

• Circular motion concepts, including angular velocity and centripetal force.Tasks

This program utilizes concepts from Physics and Astronomy to simulate the Solar System. Here's a breakdown of the relevant concepts and how they're applied in the program.

**Concepts.**

■ Kepler's Laws.

• Kepler's Laws of Planetary Motion:

• First Law (Law of Ellipses): Planets move in elliptical orbits with the Sun at one focus. (This program assumes circular orbits for simplicity).

• Second Law (Law of Areas): A line connecting the Sun and a planet sweeps equal areas in equal time intervals. (The program calculates the angular position based on the orbital period).

• Third Law (Law of Periods): The square of a planet's orbital period is proportional to the cube of its average distance from the Sun. (The program uses the orbital period to calculate the angular velocity for each planet).

■ **Circular Motion.**

• Angular Velocity (ω): The rate of change of angular position (θ) over time (t). It's calculated as ω = 2π / T, where T is the orbital period.

• Centripetal Force: The force acting on an object in circular motion that keeps it moving in a circular path. Gravity from the Sun acts as the centripetal force for planets in our Solar System.

Program Implementation.

■ **Constants:**

• AU\_TO\_KM: Converts Astronomical Units (AU) to kilometers (km).

• SPACECRAFT\_SPEED: Speed of the spacecraft used for hypothetical travel time calculations (not relevant to planetary motion).

• SECONDS\_IN\_DAY: Number of seconds in a day.

• EARTH\_YEAR\_IN\_SECONDS: Number of seconds in an Earth year.

□ Planetary Data:

• A dictionary planets stores information about each planet, including its distance from the Sun (AU), orbital period (Earth years), and color for visualization.

□ **calculate\_position Function:**

• This function calculates the x and y coordinates of a planet's position in a 2D plane based on its distance, orbital period, and the number of days

elapsed since a reference date.

□ **It uses the formula:**

• angular\_velocity = (2 \* np.pi) / (period \* EARTH\_YEAR\_IN\_SECONDS)

• theta = angular\_velocity \* days\_elapsed \* SECONDS\_IN\_DAY

• x = distance \* np.cos(theta)

• y = distance \* np.sin(theta)

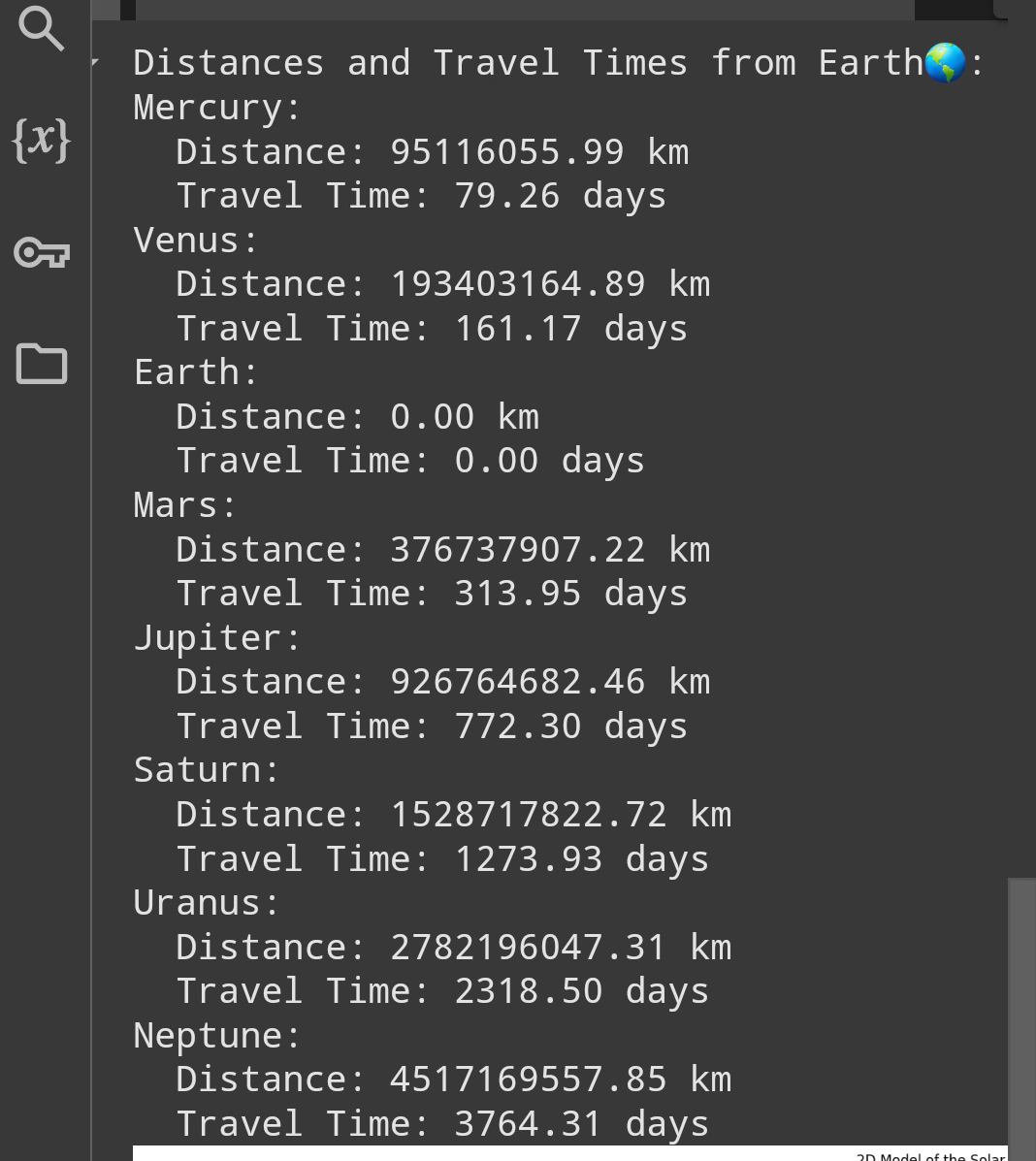
• calculate\_distance\_and\_time Function:

• This function calculates the straight-line distance (in km) and travel time (in days) between Earth and another planet. **(This for hypothetical travel and not relevant to actual planetary motion).**

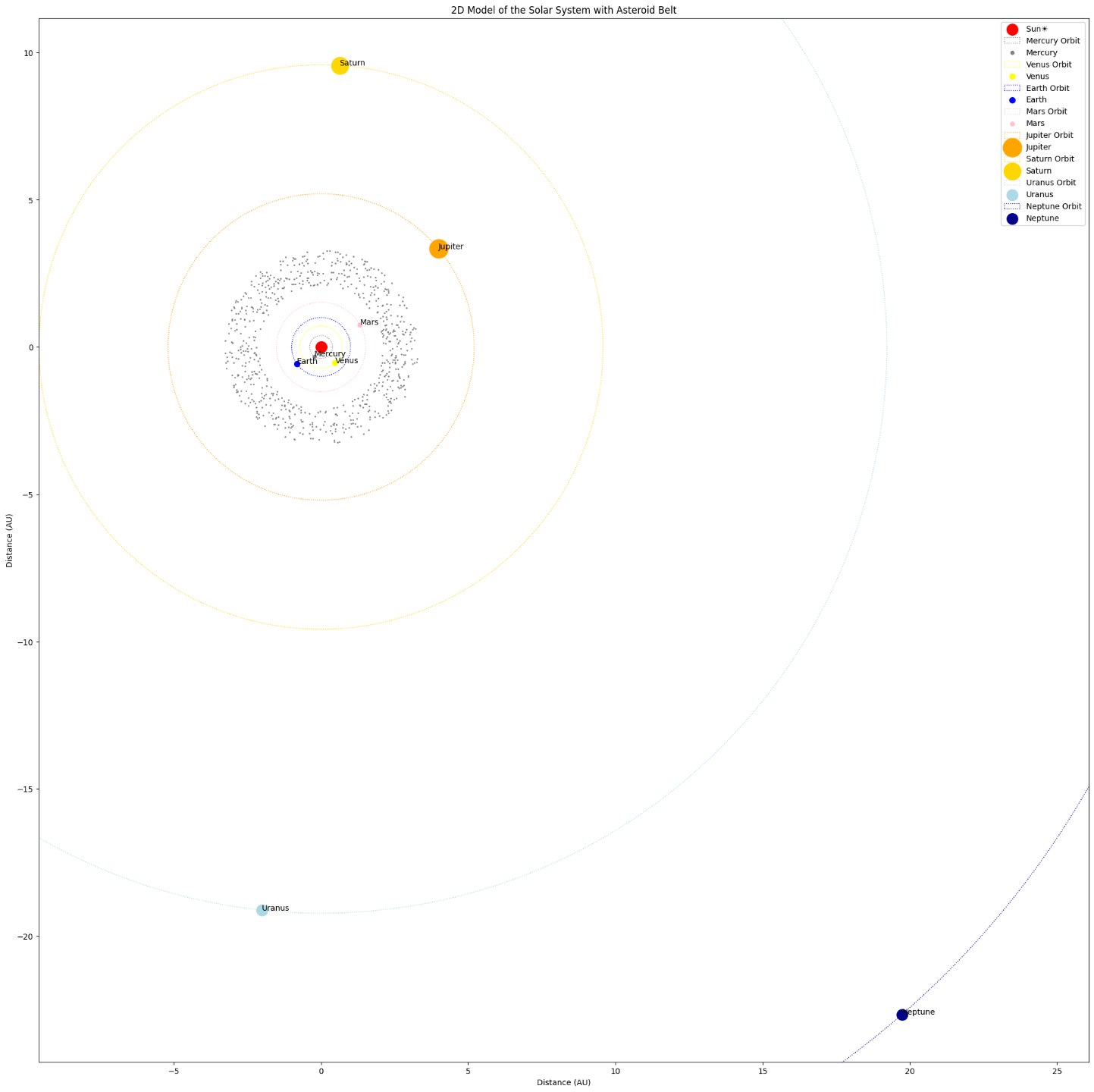
**Results after running the program.**

**We have to take hypothetical data to run this program.**

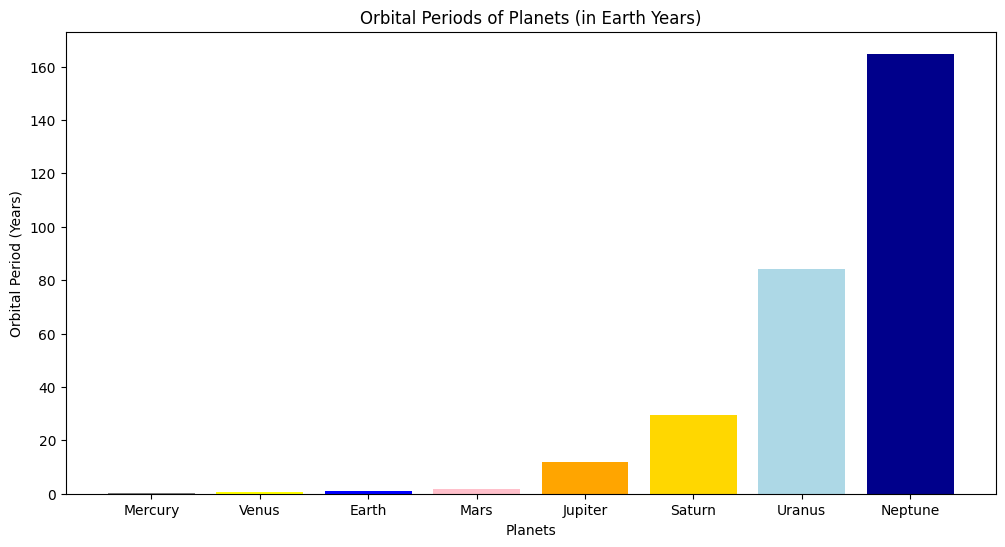
**● Let Date is 2047 - 5 - 21.**



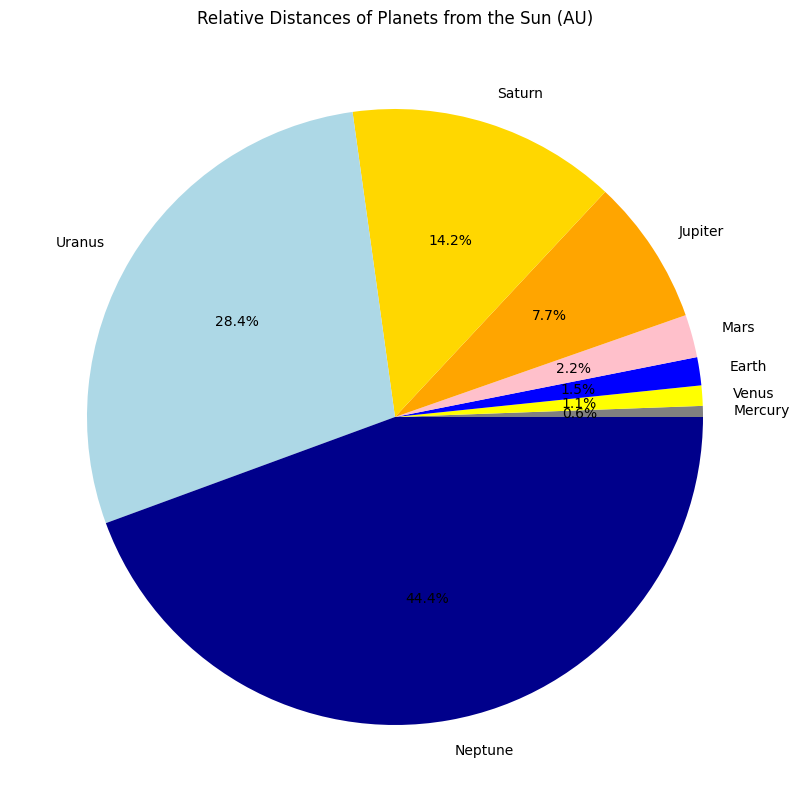
**Time taken by us to reach these planets at given timeline.**



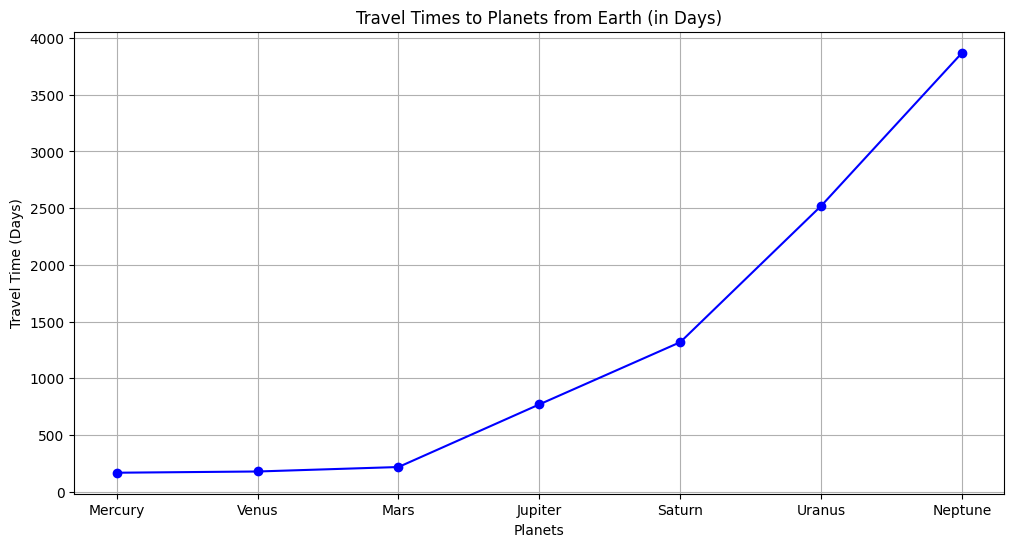
● 2d stimulation for timeline (2047 - 5- 21).



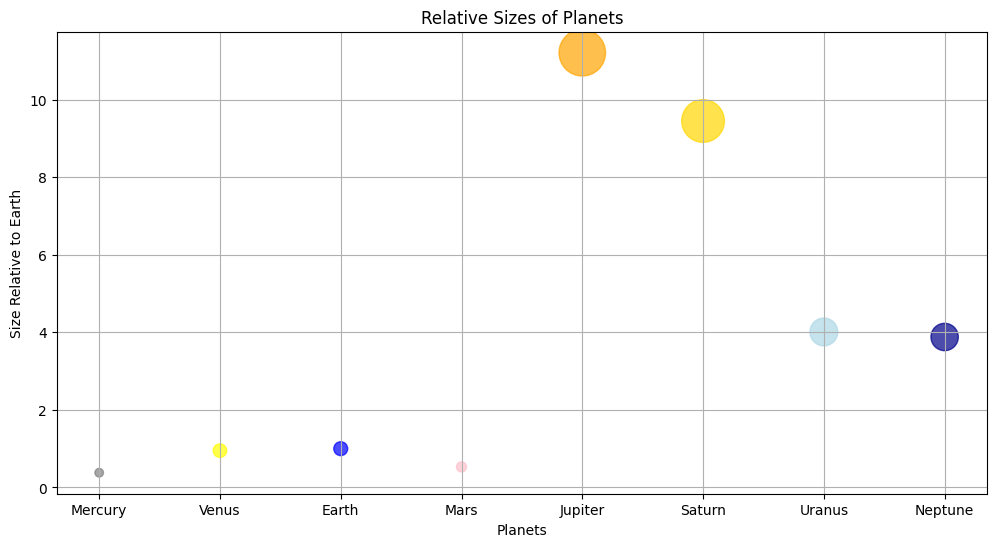
● Relative between orbital\_periods of earth and other planets.



● Distance of others planet from sun.



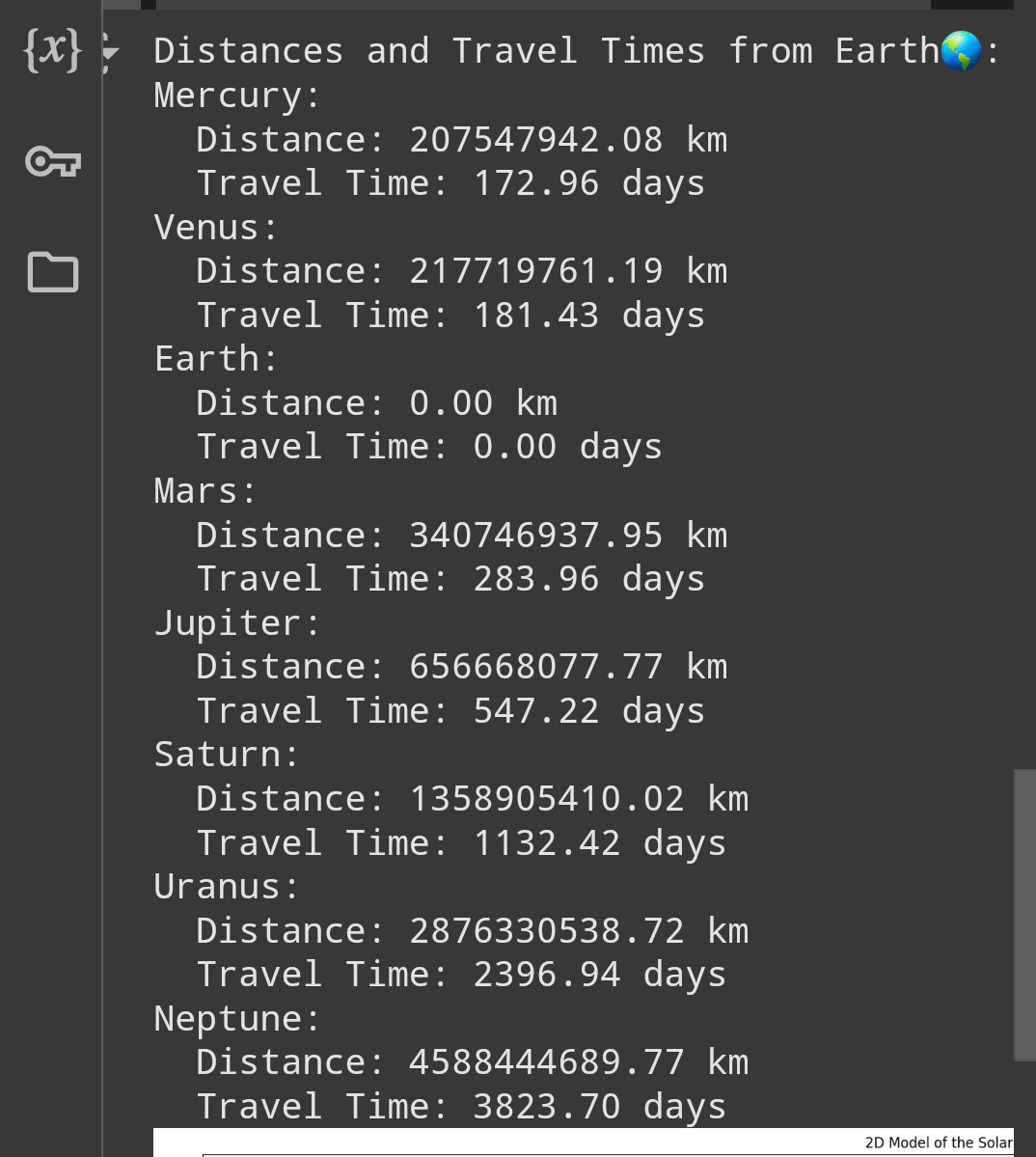
● Time taken to reach these planets



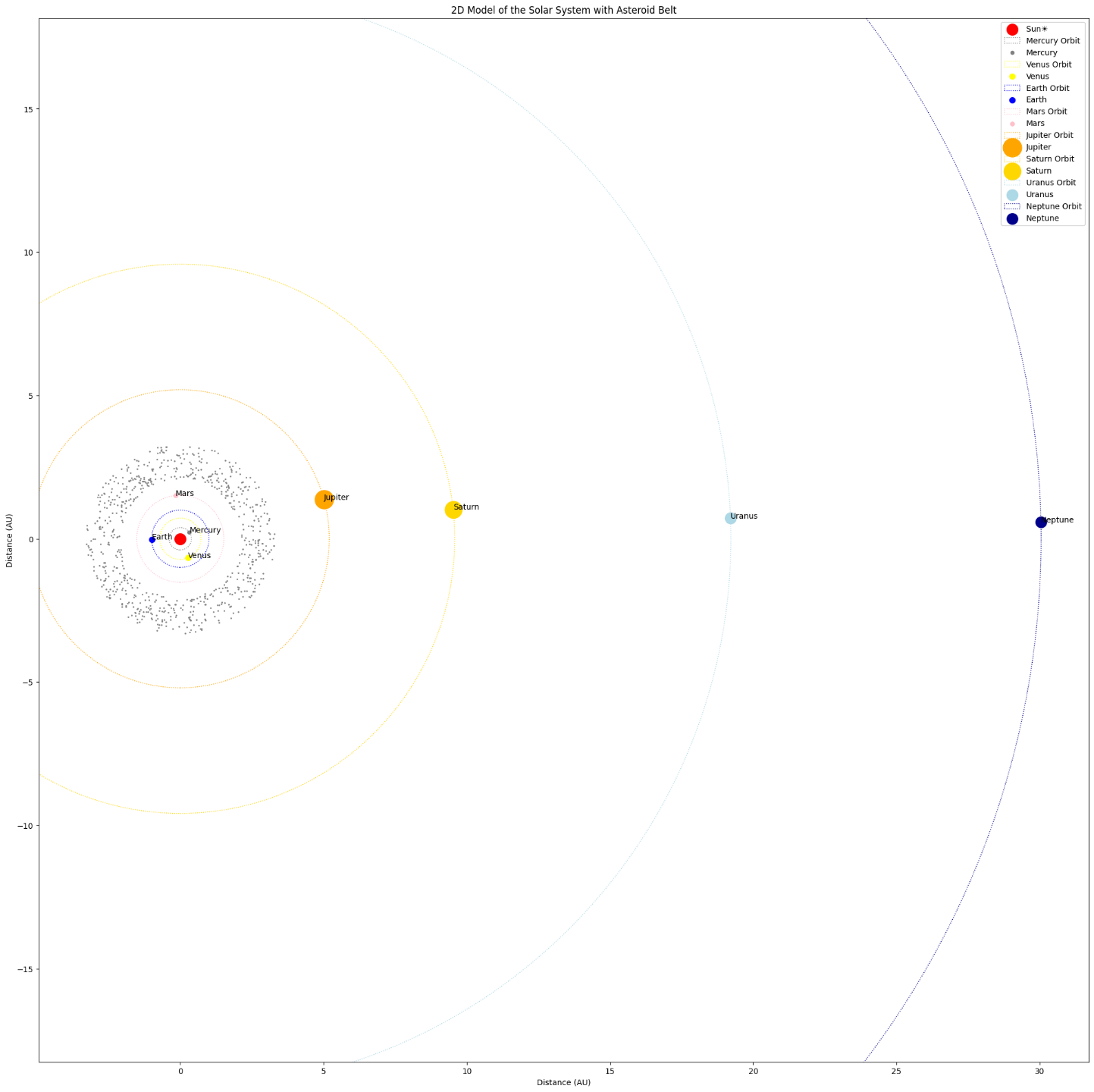
Relation of size between planets.

**Note : Tables will remain same for other timeslines.**

**● Let Date is 2006 -7 -26)**

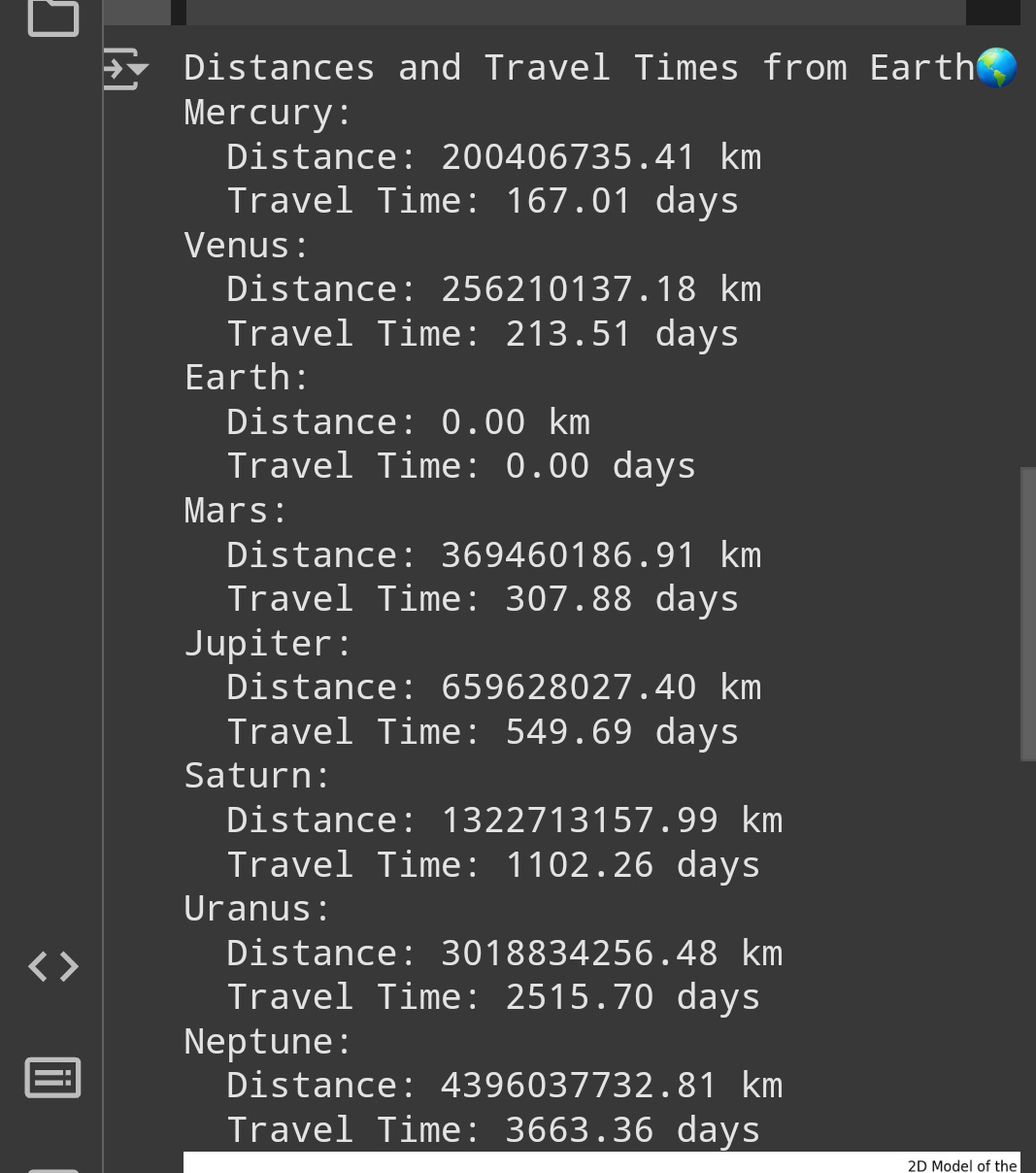


**Time taken by us to reach these planets at given timeline.**

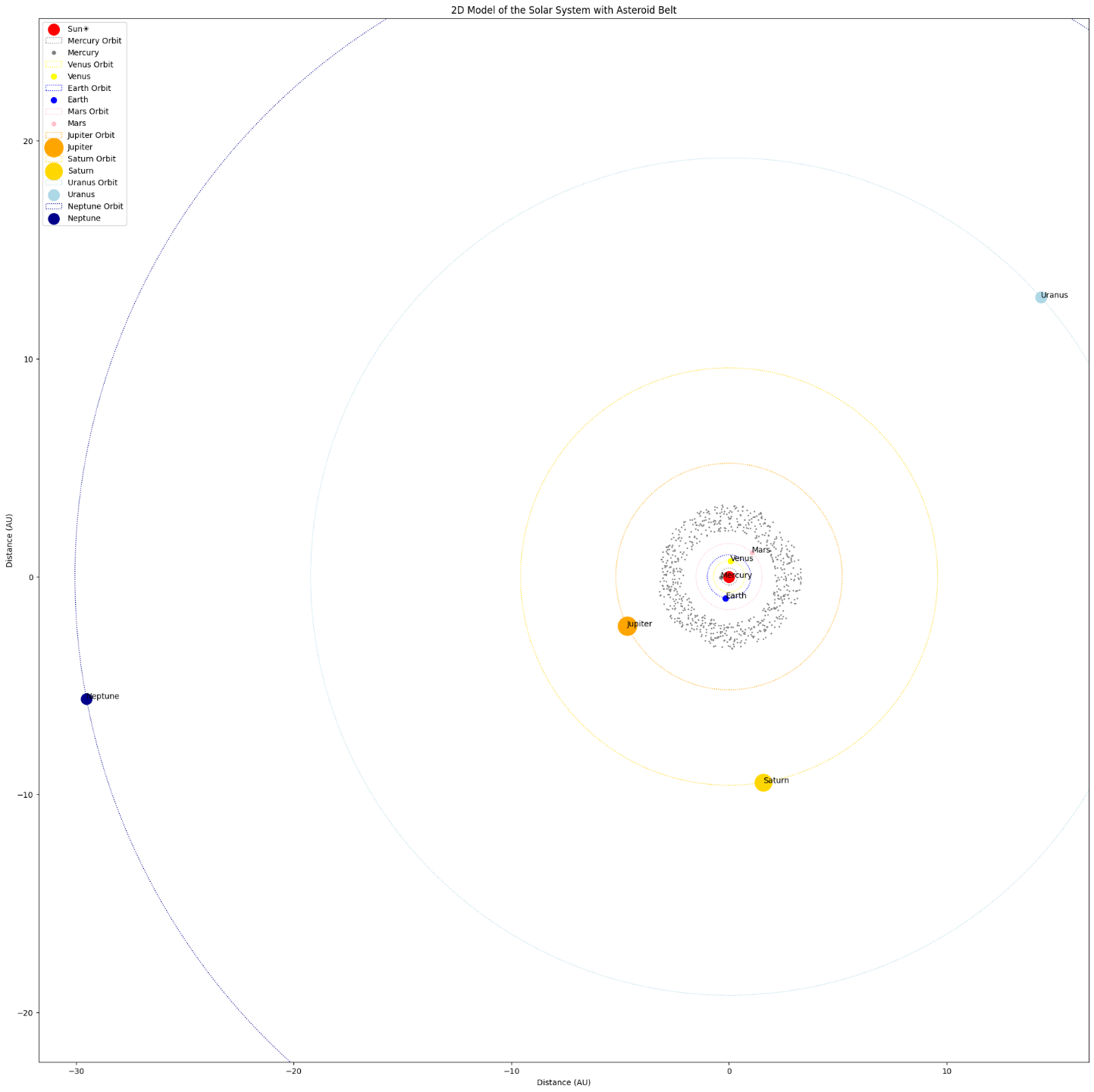


2d stimulation for timeline (2006- 7- 26) **.**

**● Let Date is 2267-4 -6)**



**Time taken by us to reach these planets at given timeline.** 2d stimulation for timeline (2267- 4- 6) **.**



●Conclusion:

This program Summarize our learnings about the Solar System, planetary motion, to understand the core principles of governing planetary motion and how scientific concepts can be translated into Python code for visualization and basic calculations. this program changes the possibilities of space exploration.

2) The 3-Body Problem 3D Simulation.

● Introduction:

The three-body problem is a classical problem in physics and astronomy that describes the motion of three celestial objects under their mutual gravitational influences. Unlike the simpler two-body problem, which has an exact solution, the three-body problem is chaotic and does not have a general analytical solution. This program numerically solves the three-body problem and visualizes the trajectories of three masses in a three-dimensional space.

● Objective and Applications in Science and Astronomy:

1. Astrophysics and Orbital Mechanics:

•Understanding the motion of celestial bodies such as stars in a triple-star system (e.g., Alpha Centauri) or moons in a planetary system.

•Studying chaotic systems and resonance phenomena in celestial mechanics.

2. Space Exploration:

•Predicting the behavior of spacecraft in multi-body gravitational fields.

•Designing stable orbits for satellites and space probes.

This program utilizes concepts from Physics and Astronomy to simulate the Three-body problem. Here's a breakdown of the relevant concepts and how they're applied in the program.

● **Concepts**.

□ Newton's Law of Universal Gravitation.

• Newton's 1st law.

The program is based on Newton's Law of Universal Gravitation, which states:

F = G \* m1 m2/r²

•F is the gravitational force between two masses.

• G is the gravitational constant.

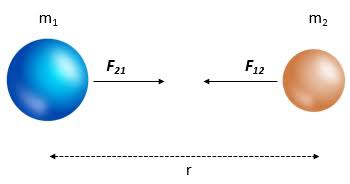
• m¹ and m² are the masses of the objects.

• r is the distance between the centers of the two masses.

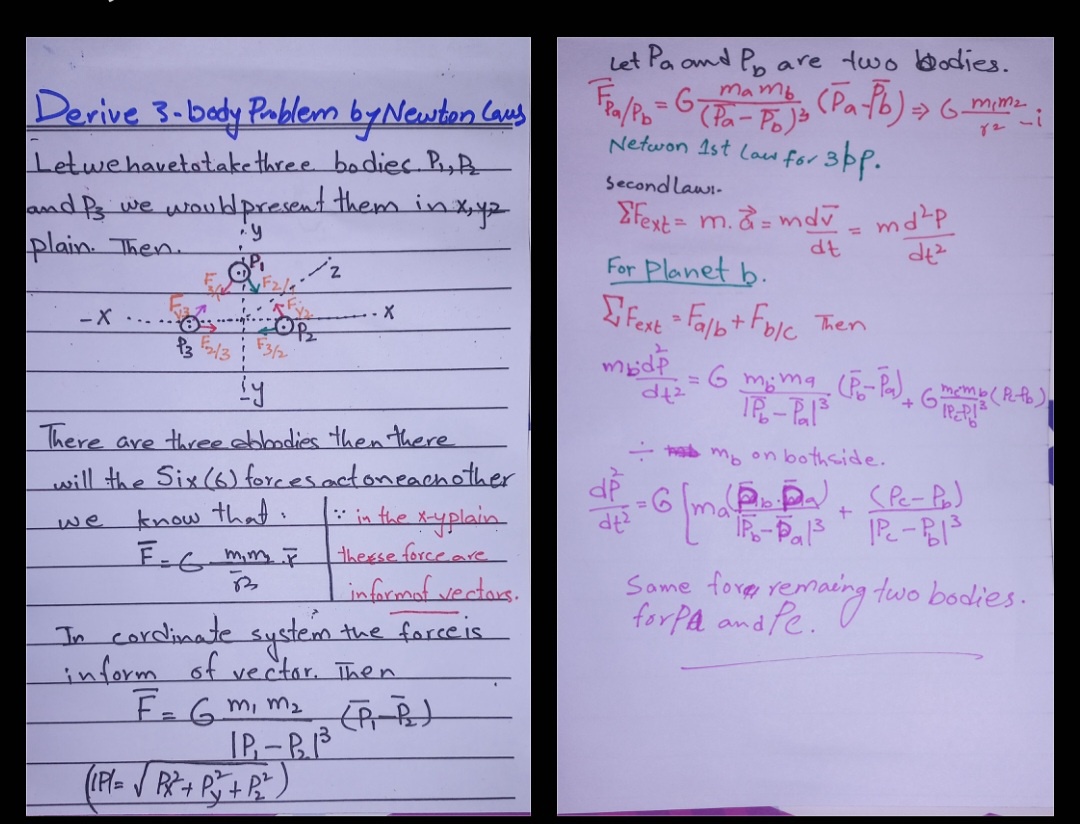
This law is applied to calculate the gravitational forces acting on each of the three bodies due to the other two.

Newton's Second Law of Motion

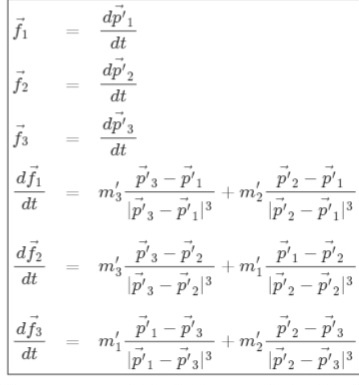
Newton's Second Law, , is used to relate the forces to the accelerations of the bodies. Combining this with the gravitational force gives the equations of motion for the system.



That is possible for 2 bodies. For three-body problem we have to take 3 hypothetical bodies. To find unknown forces on each bodies. Then



That is derivatives for finding the forces. After that we obtain following forces for remain planets.



There are many cases of three-body problem.

• All three bodies have the same masses.

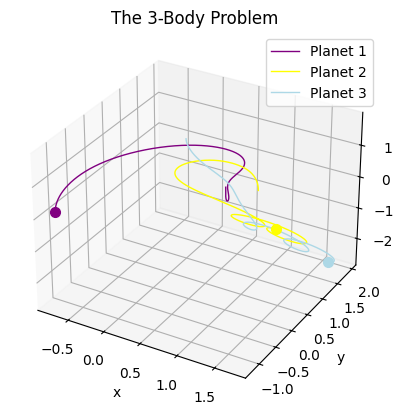
• Two bodies having same masses and one having different mass.

• no same masses of all bodies.

**Results after running the program.**

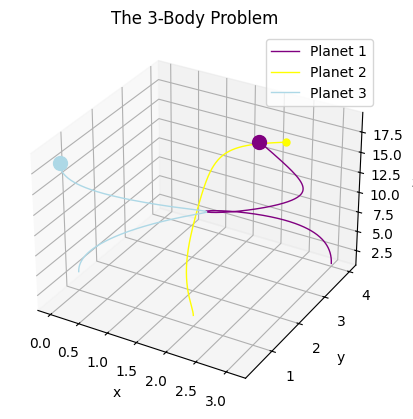
**We have to take hypothetical data to run this program.**

**• If all bodies have the same masses.**

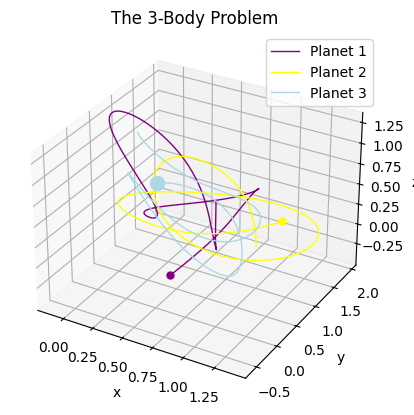


**3d for 3body problem**

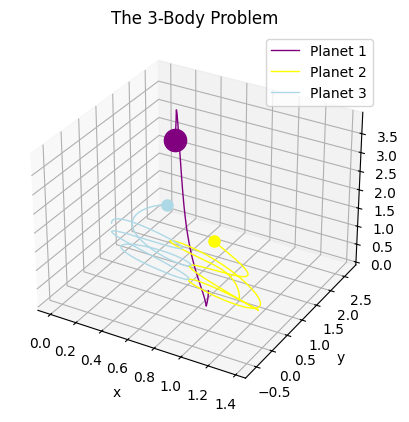
**• Two bodies having same masses and one having different mass.**

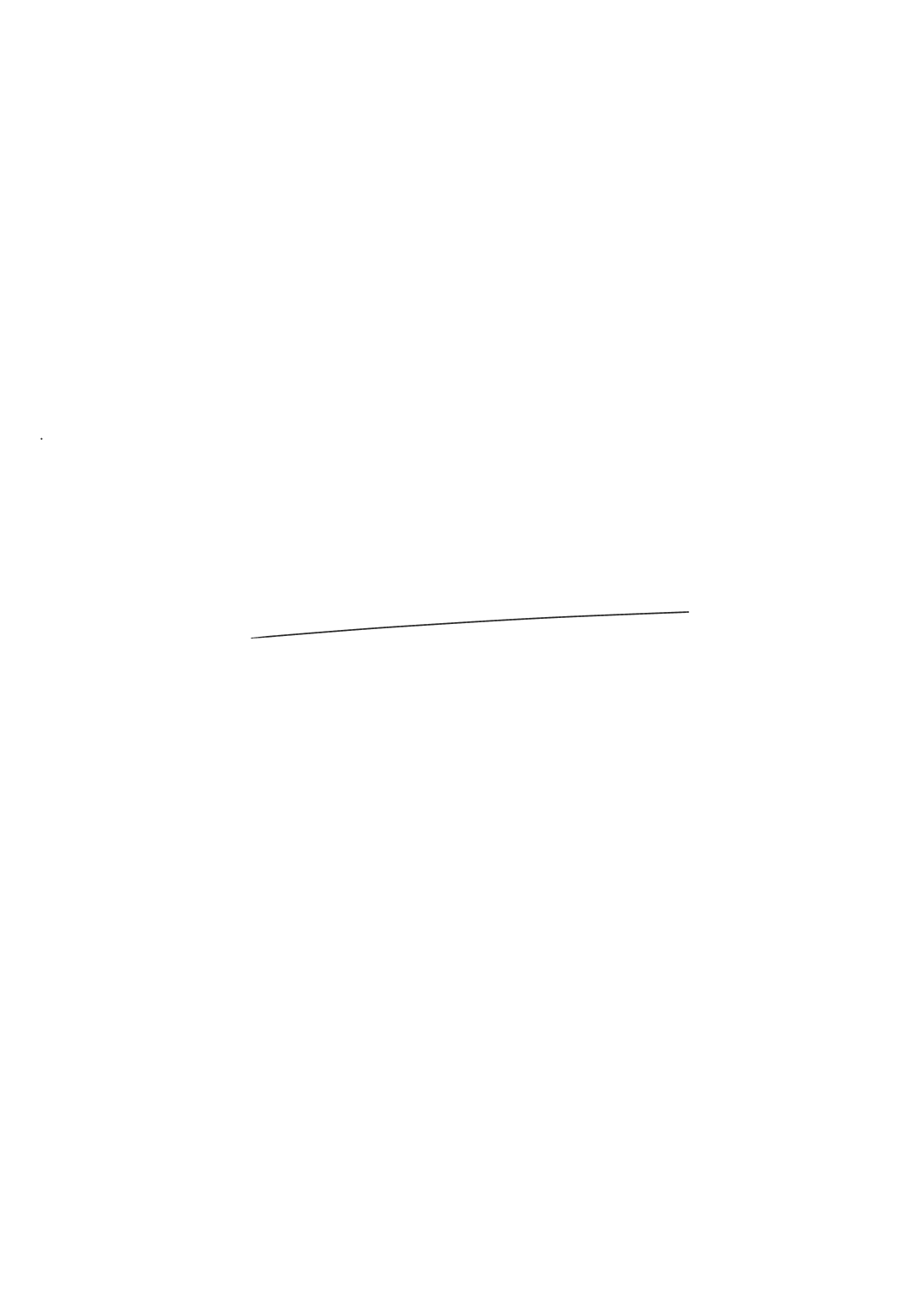


• **no same masses of all bodies.**



**• One body having the largest mass then other.**





● **Conclusion:**

This Python program successfully demonstrates the principles of celestial mechanics through a numerical solution of the three-body problem. By incorporating foundational laws of physics and leveraging computational techniques, it provides valuable insights into the behavior of gravitational systems. This simulation highlights the power of programming in solving complex problems in physics and astronomy, bridging theoretical concepts and practical applications.

For programs .

https://github.com/Tahsinnazar/Ai-cadmey-projects/blob/main/3\_Bodyproblem\_and\_Cosmic\_Voyage\_estimator.ipynb

