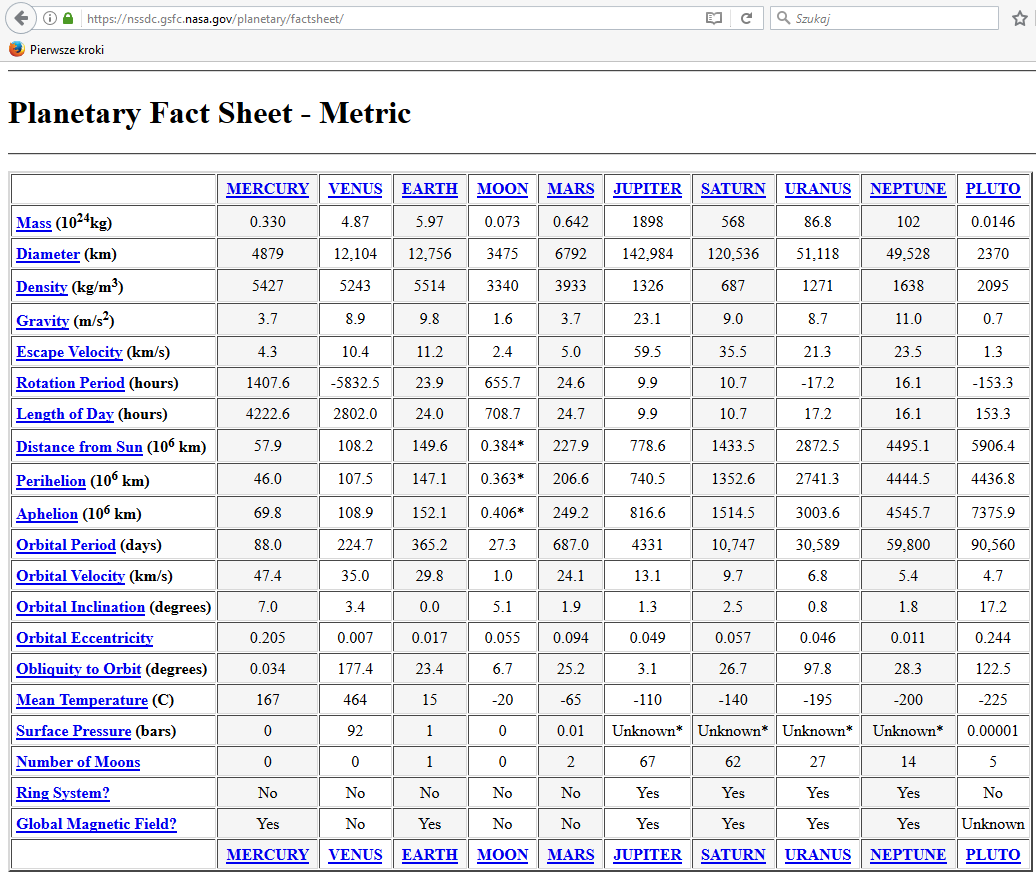
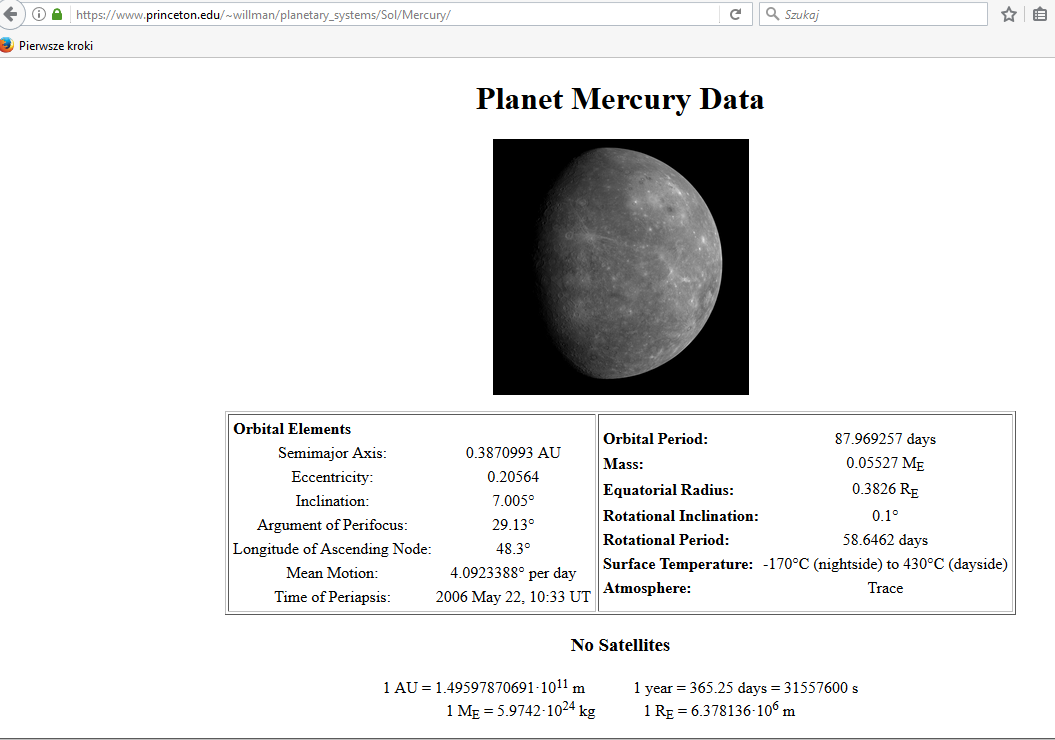
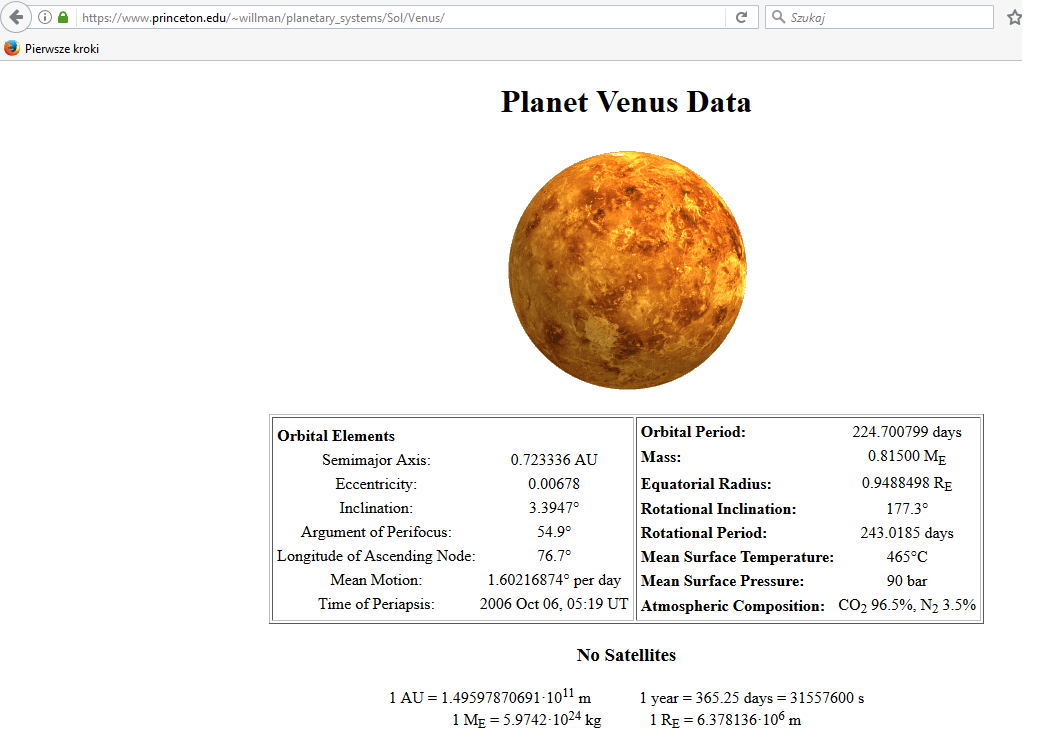
**Dokumentacja**

Wniosek: Potrzebna prędkość do opuszczenia układu słonecznego z ziemi potrzebna jest prędkość wystrzelenia mieszcząca się w granicach 11.76m/s , a 12.00m/s.

**Dane odnośnie planet:**

****

****

**1.Ruch planet**

Obliczanie powierzchnii

**C:\Users\Krystian\Desktop\Dokumentacja\CodeCogsEqn(1).gif**

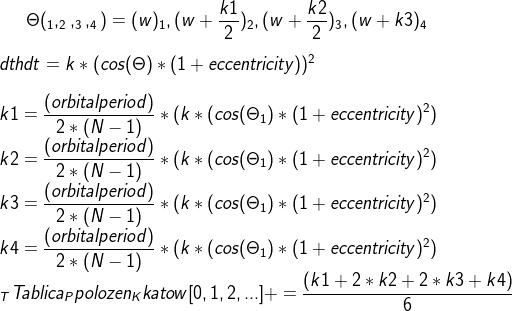
**C:\Users\Krystian\Desktop\Dokumentacja\CodeCogsEqn(3).gif**

**C:\Users\Krystian\Desktop\Dokumentacja\CodeCogsEqn(2).gif**

Pochodna orbity – w zależności od powierzchni i argumentu Theta

**C:\Users\Krystian\Desktop\Dokumentacja\CodeCogsEqn(6).gif**

**Ustalanie trajektorii orbit – Runge kutta 4th**

****

K1 = orbitDt \* dthdt (w);

k2 = orbitDt \* dthdt (w + k1 / 2);

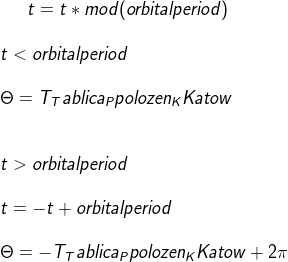
k3 = orbitDt \* dthdt (w + k2 / 2);

k4 = orbitDt \* dthdt (w + k3);

w = w + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

katy [i + 1] = w;

**Obliczanie \Theta – potrzebne do wyliczenia argumentu(theta) - itp. wyznaczanie właściwej pozycji w oparciu o poprzednie wyliczenia z algorytmu rk4**

****

**Parametryzacja Orbit – w oparciu o kąty theta**

**Heliocentric coordinates of the planet**

Having found the true anomaly and the radius vector of the planet, we can go on to find the position of the planet with respect to the plane of the ecliptic. The formulas below are a combination of 'resolving' to find components and rotations around various axes to transform the coordinates to the Ecliptic frame. We might expect the formulas to involve the inclination of the planet's orbit (i), and various angles within the plane of the orbit, as well as the longitude of the ascending node (o).

**X = r \* [cos(o) \* cos(v + p - o) - sin(o) \* sin(v + p - o) \* cos(i) ]** //Nie brane pod uwage

**Y = r \* [sin(o) \* cos(v + p - o) + cos(o) \* sin(v + p - o) \* cos(i)]** // Z = r \* [sin(v + p - o) \* sin(i)]

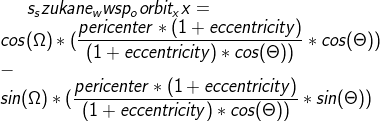
**r is radius vector**

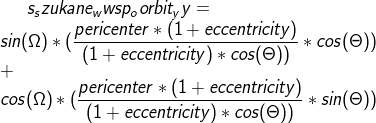
**v is true anomaly**

**o is longitude of ascending node**

**p is longitude of perihelion**

**i is inclination of plane of orbit**

** C:\Users\Krystian\Desktop\Dokumentacja\CodeCogsEqn-1.gif**

** C:\Users\Krystian\Desktop\Dokumentacja\CodeCogsEqn-2.gif**

The **longitude of the ascending node** (☊ or Ω) is one of the [orbital elements](https://en.wikipedia.org/wiki/Orbital_elements) used to specify the [orbit](https://en.wikipedia.org/wiki/Orbit) of an object in space. It is the angle from a reference direction, called the [*origin of longitude*](https://en.wikipedia.org/w/index.php?title=Origin_of_longitude&action=edit&redlink=1), to the direction of the [ascending node](https://en.wikipedia.org/wiki/Ascending_node), measured in a [reference plane](https://en.wikipedia.org/wiki/Reference_plane).[[1]](https://en.wikipedia.org/wiki/Longitude_of_the_ascending_node#cite_note-1) The ascending node is the point where the orbit of the object passes through the plane of reference, as seen in the adjacent image

**2.Rotacja Planet**

**Wykorzystana przechylenie planet**

Description

Quaternions are used to represent rotations.

They are compact, don't suffer from gimbal lock and can easily be interpolated. Unity internally uses Quaternions to represent all rotations.  
  
They are based on complex numbers and are not easy to understand intuitively. You almost never access or modify individual Quaternion components (x,y,z,w); most often you would just take existing rotations (e.g. from the [Transform](https://docs.unity3d.com/ScriptReference/Transform.html)) and use them to construct new rotations (e.g. to smoothly interpolate between two rotations). The Quaternion functions that you use 99% of the time are: [Quaternion.LookRotation](https://docs.unity3d.com/ScriptReference/Quaternion.LookRotation.html), [Quaternion.Angle](https://docs.unity3d.com/ScriptReference/Quaternion.Angle.html), [Quaternion.Euler](https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html), [Quaternion.Slerp](https://docs.unity3d.com/ScriptReference/Quaternion.Slerp.html), [Quaternion.FromToRotation](https://docs.unity3d.com/ScriptReference/Quaternion.FromToRotation.html), and [Quaternion.identity](https://docs.unity3d.com/ScriptReference/Quaternion-identity.html). (The other functions are only for exotic uses.)

# [Quaternion](https://docs.unity3d.com/ScriptReference/Quaternion.html).Euler

## Description

Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis (in that order).

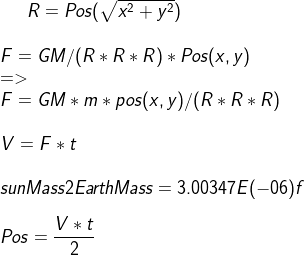
using UnityEngine;

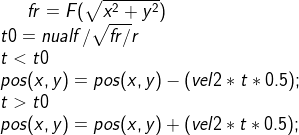
using System.Collections;  
  
public class ExampleClass : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html) {

public [Quaternion](https://docs.unity3d.com/ScriptReference/Quaternion.html) rotation = [Quaternion.Euler](https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html)(0, 30, 0);

}

**3.Ruch rakiety**





**Zródła**

http://www.bogan.ca/orbits/kepler/orbteqtn.html

<https://www.math.ksu.edu/math240/book/capstone/kepler.php>

<http://www.braeunig.us/space/orbmech.htm>

https://www.classe.cornell.edu/~seb/celestia/orbital-parameters.html