In this unit we will learn to do some physics with this system. This unit is about Newtonian mechanics. We will get up to simple problems of astronomy.

Before we start we need to spend a little time on physical constants and units. These can be obtained from within Mathematica. Let's look at this quickly, but in most cases we will not use it.

The command Quantity[magnitude, "item"] will give you some physical constant "item" with magnitude.

```
In[•]:= Quantity[2, "SpeedOfLight"]
Out[•]= 2 c

In[•]:= myspeed = UnitConvert[Quantity[1 / 10, "SpeedOfLight"], "Meters/Sec"]
Out[•]= 149 896 229 / 5 m/s

In[2]:= UnitConvert[Quantity[1, "ElectronMass"], "kg"]
Out[2]= 9.109384 × 10^-31 kg
```

You can extract both the magnitude and the units. This is useful when you want to get the numbers to setup or when you want to write a careful program. It gets cumbersome for day to day calculations. And so I tend to set up the units in the beginning of a calculation into simple variables.

```
In[•]:= QuantityMagnitude[myspeed]
Out[•]= 149 896 229 / 5
```

```
In[•]:= QuantityUnit[myspeed]
```

$$\text{Out[•]=}\ \frac{\text{Meters}}{\text{Seconds}}$$

Lets calculate the escape velocity from the Earth

First we setup some constants.

```
In[•]:= myG = QuantityMagnitude[
          UnitConvert[Quantity["GravitationalConstant"], "Newtons*meter^2/kg^2"]]
```

$$\text{Out[•]=}\ 6.674 \times 10^{-11}$$

```
In[•]:= radiusofearth =
         QuantityMagnitude[UnitConvert[Quantity["EarthMeanRadius"], "Meters"]]
```

$$\text{Out[•]=}\ 6.3710088 \times 10^{6}$$

```
In[•]:= massofearth = QuantityMagnitude[UnitConvert[Quantity["EarthMass"], "Kg"]]
```

$$\text{Out[•]=}\ 5.9721986 \times 10^{24}$$

Now we setup some equations.

```
In[•]:= KineticEnergy = m * v^2 / 2;
        PotentialEnergy = -myG * massofearth * m / r;
        totalenergy = KineticEnergy + PotentialEnergy;
```

If a particle barely escapes from any radius r from the center of the earth, then it must have total energy zero. What is the velocity ? Notice it is a quadratic equation and so it will have negative and positive

```
In[•]:= Solve[totalenergy == 0, v]
```

••• Solve: Solve was unable to solve the system with inexact coefficients. The answer was obtained by solving a corresponding exact system and numericizing the result.

$$\text{Out[•]=}\ \left\{\left\{v \to -\frac{2.823 \times 10^{7}}{\sqrt{r}}\right\}, \left\{v \to \frac{2.823 \times 10^{7}}{\sqrt{r}}\right\}\right\}$$

Let's take the positive solution

```
In[•]:= answer  = %[[2]]
```

$$\text{Out[•]=}\ \left\{v \to \frac{2.823 \times 10^{7}}{\sqrt{r}}\right\}$$

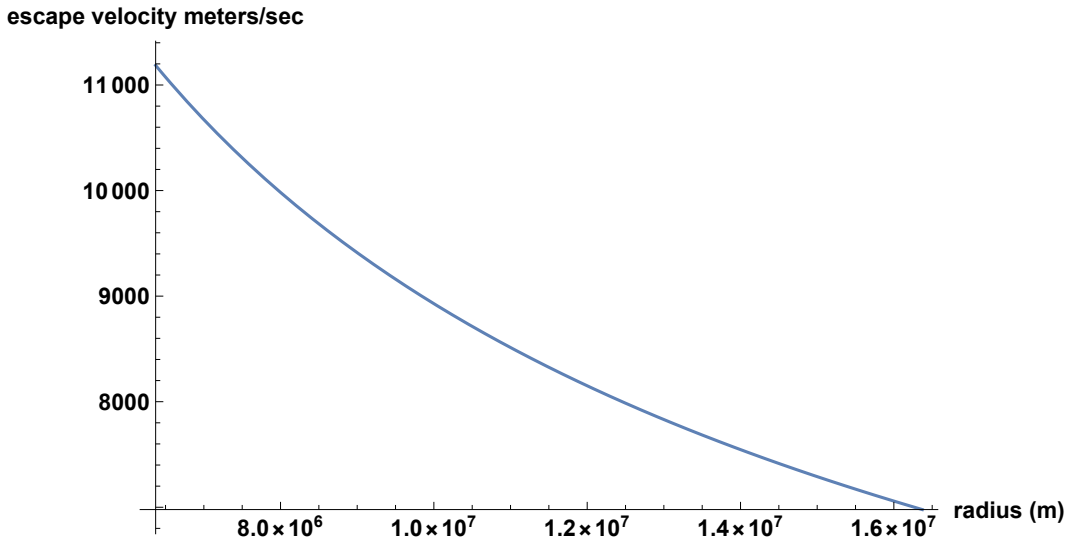
What is the answer at the Earth's surface in units of meters/sec

```
In[•]:= answer /. {r → radiusofearth}
Out[•]= {v → 1.119 × 10^4}
```

what about plotting it as a function of the height above the earth, up to 10000 km.

```
In[•]:= Plot[v /. answer, {r, radiusofearth, radiusofearth + 10 000 * 10 ^ 3},
  PlotStyle → Bold, AxesLabel → {"radius (m)", "escape velocity meters/sec"},
  LabelStyle → {Medium, Bold}]
```

Out[•]=



How about a projectile in a uniform gravitational field ? we are now going to solve Newton's equations in a uniform field. The units will be meters, seconds, and kg.

We have to learn a little bit about differential equations.

```
In[•]:= g = 9.8;  (* gravity acceleration  in meters/sec^2 *)
```

In[•]:= **? D**

D[f, x] gives the partial derivative $\partial f/\partial x$.
D[f, {x, n}] gives the multiple derivative $\partial^n f/\partial x^n$.
D[f, x, y, …] gives the partial derivative $\cdots (\partial/\partial y)(\partial/\partial x) f$.
D[f, {x, n}, {y, m}, …] gives the multiple partial derivative $\cdots (\partial^m/\partial y^m)(\partial^n/\partial x^n) f$.
D[f, {{x_1, x_2, …}}] for a scalar f gives the vector derivative $(\partial f/\partial x_1, \partial f/\partial x_2, \ldots)$.
D[f, {$array$}] gives an array derivative. ≫

Differentiation is indicated by either D[F[x],x] or by F''[t]. We use it for Newton's equations of motion. D[F[x],{x,n}] is the n'th derivative of F[x].

Let's imagine a ball tossed. z[t] is the vertical motion and x[t] is the horizontal motion. Remember that to define an equation you must use ==. An equation has a value of True or False that can be assigned to a variable !

In[•]:=
```
equationsofmotion = {m * D[z[t], {t, 2}] == -m g , m * D[x[t], {t, 2}] == 0}
```
Out[•]=
```
{m z''[t] == -9.8 m, m x''[t] == 0}
```

In[•]:=
```
vx0 = 20; vz0 = 20;
initialconditions = {z[0] == 0,  x[0] == 0, z'[0] == vz0, x'[0] == vx0 }
```
Out[•]=
```
{z[0] == 0, x[0] == 0, z'[0] == 20, x'[0] == 20}
```

In[•]:=
```
alltogether = Join[equationsofmotion, initialconditions]
```
Out[•]=
```
{m z''[t] == -9.8 m, m x''[t] == 0, z[0] == 0, x[0] == 0, z'[0] == 20, x'[0] == 20}
```

We use DSolve to solve the equations. But we have to join all the equations together.

In[•]:=
```
answer = DSolve[alltogether, {z[t], x[t]}, t]
```
Out[•]=
```
{{z[t] → 20. t - 4.9 t^2, x[t] → 20. t}}
```

when does the ball come back down to z=0 ? Solutions are in the form of a substitution rule.

In[3]:=
```
times = Solve[z[t] == 0 /. answer, t]
```
… ReplaceAll: {answer} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing.

… Solve: z[t] == 0 /. answer is not a quantified system of equations and inequalities.

Out[3]=
```
Solve[z[t] == 0 /. answer, t]
```

```
In[1]:= totaltime = t/.times[[2]]
```

... Part: Part specification times[[2]] is longer than depth of object.

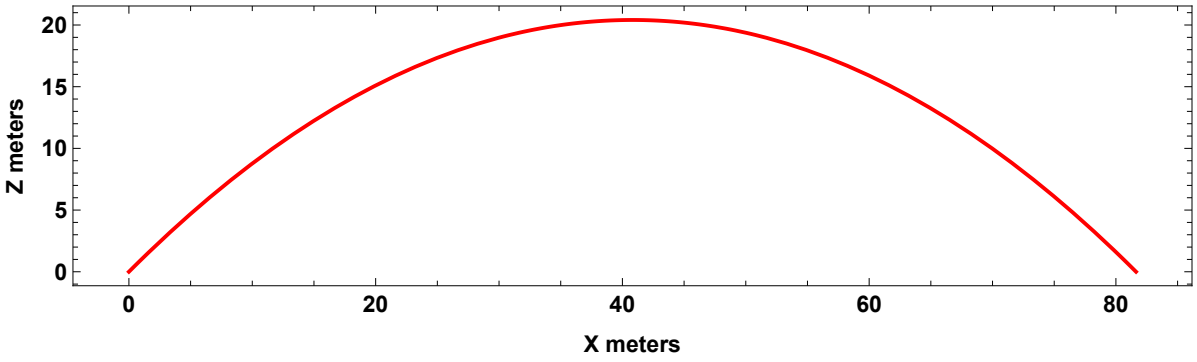
... ReplaceAll: {times[[2]]} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing.

```
Out[1]= t /. times[[2]]
```

Lets make a plot of the trajectory !

```
In[•]:= ParametricPlot[{x[t], z[t]} /. answer, {t, 0, totaltime}, PlotStyle → {Red, Thick},
  Frame → True, FrameLabel → {"X meters", "Z meters"}, LabelStyle → {Medium, Bold}]
```

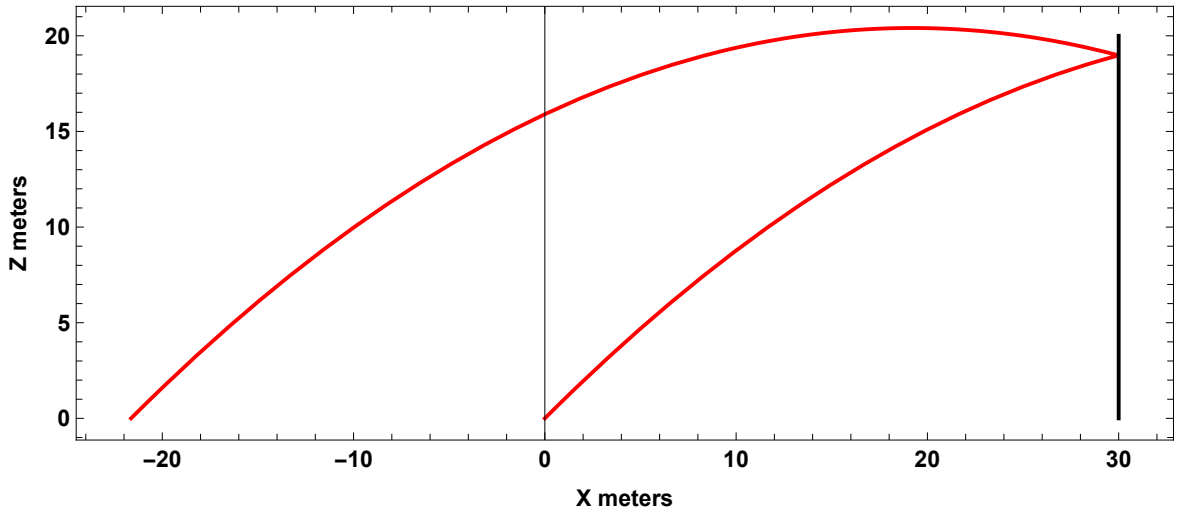
Out[•]=



Let's imagine that there is a wall at x = xwall < 80 meters. Then the time the ball hits the wall is
twall = xwall/vx0. The ball will reflect, but its upwards/downwards motion is not disturbed.
We can therefore replace x[t] with a If statement.

This is getting pretty sophisticated ! But it is not difficult to do. I even drew a line where the wall is.
Try changing the parameter xwall to see what happens to the plot.

```
In[•]:= xwall = 30;
  twall = xwall / vx0;
  ParametricPlot[{If[t < twall, x[t], 2 * xwall - x[t]], z[t]} /. answer,
   {t, 0, totaltime}, PlotStyle → {Red, Thick}, Frame → True,
   FrameLabel → {"X meters", "Z meters"}, LabelStyle → {Medium, Bold},
   Epilog → {Directive[Thick], Line[{{xwall, 0}, {xwall, 20}}]} ]
```

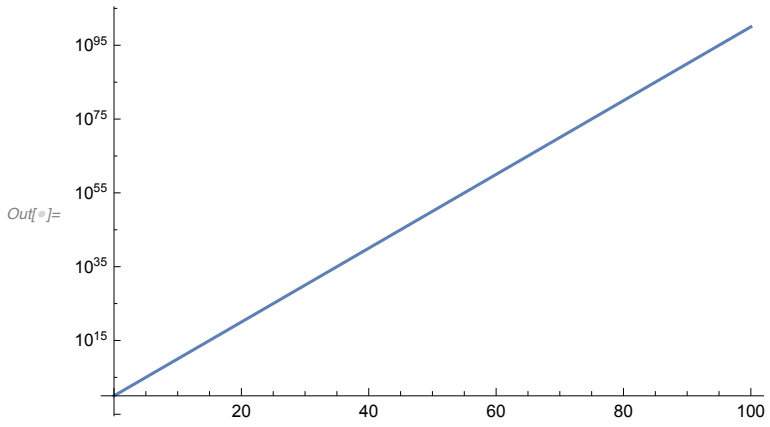

20
15
10
5
0
Z meters
−20
−10
0
10
20
30
X meters

Lets review some of the many plotting functions.

Plot - we already used this
LogPlot - makes a plot with vertical axes Log
LogLinearPlot - makes a plot with horizontal axes log.
LogLogPlot
Plot3D - plots a surface in 3D
ParametricPlot - plot a curve in 2D with a parameter (time)
ContourPlot - make a 3D plot with one dimension as contours
ListPlot - makes plots of lists of data points.

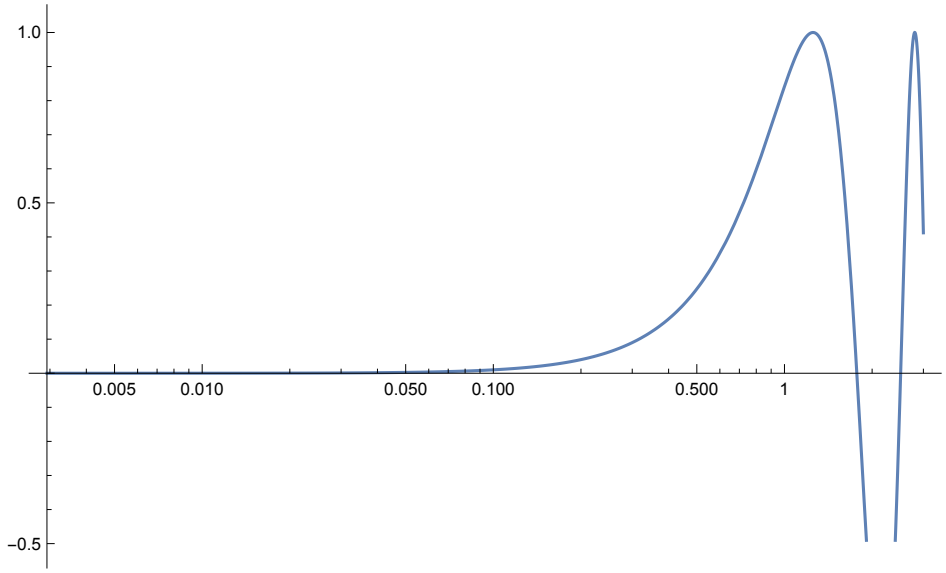
In[]:=
```
LogPlot[10^x, {x, 0, 100}]
```

Out[]=



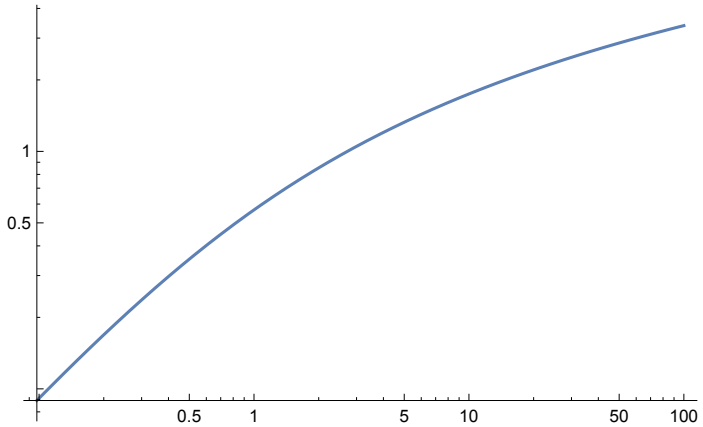
In[•]:=
```
LogLinearPlot[Sin[x^2], {x, 0, 3}]
```

Out[•]=



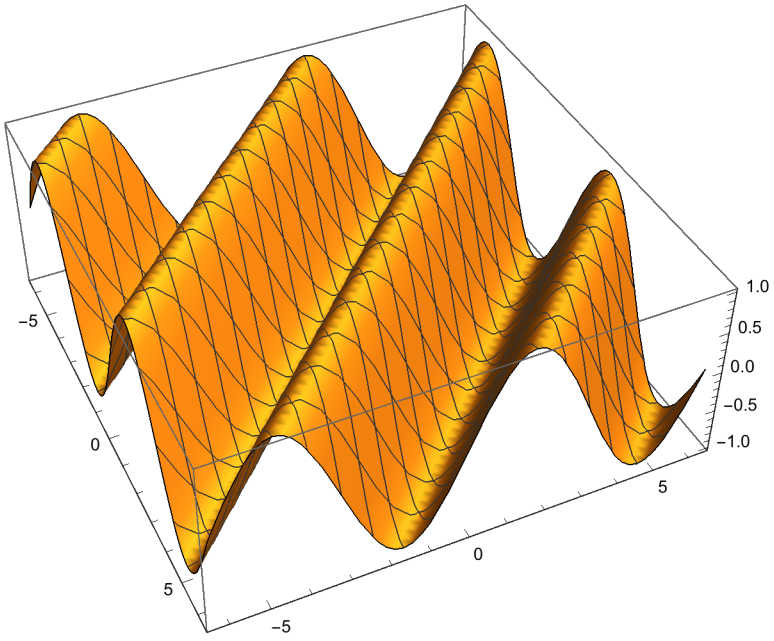

```
LogLogPlot[ProductLog[x], {x, 0, 100}]
(* this is a weird function it is also called LambertW *)
```

Out[•]=



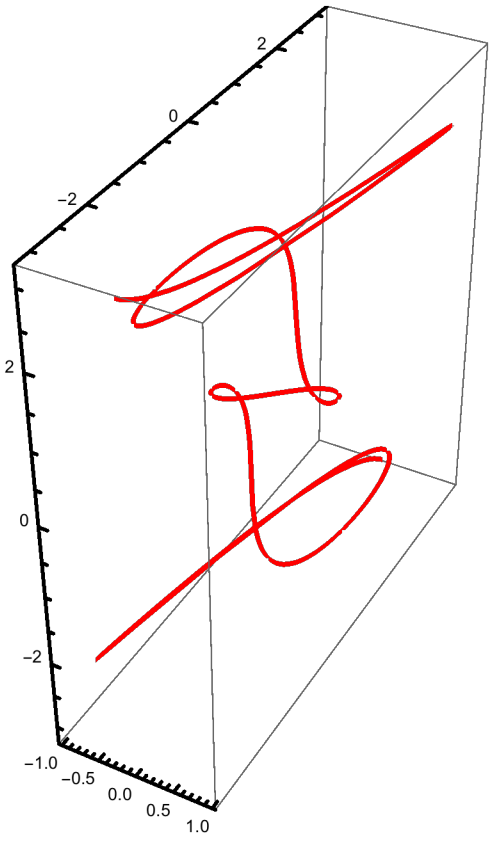
In[•]:=
```
Plot3D[Sin[x + y], {x, -2 Pi, 2 Pi}, {y, -2 Pi, 2 Pi}]
```

Out[•]=



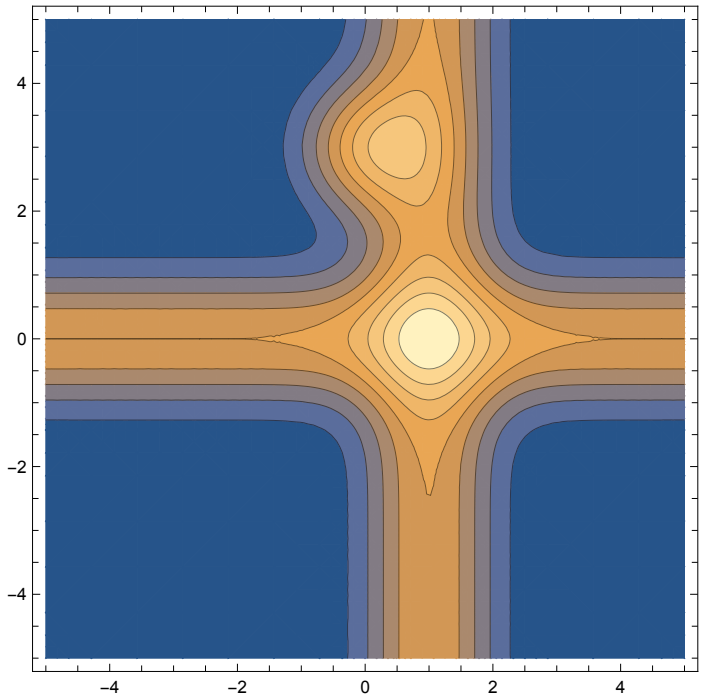
In[•]:=
```
ParametricPlot3D[{Sin[3 t], t Cos[5 t], t},
 {t, -Pi, Pi}, PlotStyle → {Red, Thick}, AxesStyle → Thick]
```

Out[•]=



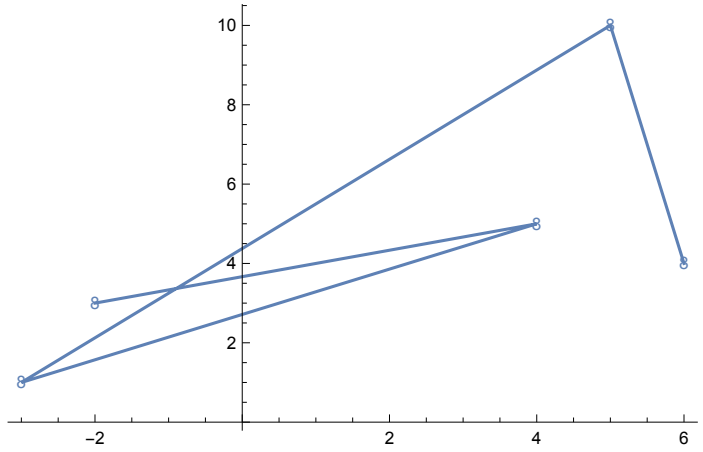
In[•]:=
```
ContourPlot[Exp[-x^2] Exp[-(y-3)^2] + Exp[-(x-1)^2] + Exp[-(y)^2],
 {x, -5, 5}, {y, -5, 5}]
```

Out[•]=



In[•]:=
```
ListPlot[{{-2, 3}, {4, 5}, {-3, 1}, {5, 10}, {6, 4}}, PlotMarkers → 8, Joined → True]
```

Out[•]=



Let's do a Rocket problem. This is something that can be done in the class. This one can be done with just algebra. But it is not simple.

A small rocket with mass m has a thruster that operates for a very brief time (dt) and the rocket flies to a height h.

How can we approximately figure out the average force (fave) in Newtons that was applied to the rocket by the thruster ?

We can do the experiment with a bottle rocket filled with vinegar and soda. We have to figure out both the time dt and the force fave and therefore we need two equations.

What if we double the mass of the rocket by attaching some small weight and measure the height again ?

In[34]:=
```
rocketmass = 0.010 ; (* in Kg *)
height1 = 20; (* meters with the first mass *)
height2 = 4.9 ;
(* meters with the mass doubled by attaching a coin for example *)
Clear[g];
```

Now we do a little bit of algebra. Remember that the rocket gets a short burst of thrust. and so
dt << time of flight.

```
v'[t] = F/mass - g;  (* this is just Newton's equation when F is on *)
(* recall that F is on only for a short
 time dt threfore integrating both sides *)
v0 = F * dt/mass - g dt;  (* we are trying to
 determine both F and dt. Here we use dt << t *)

vv  = v0 - g t;  (* velocity as a function
 of time after getting the initial velocity v0 *) ;
sl = Solve[vv == 0, t] // Flatten; (* Time to reach the top of the trjaectory *)
ttop = t /. sl
```

Out[•]= $\frac{dt\ F - dt\ g\ mass}{g\ mass}$

```
In[•]:= Z[t_] :=  v0 t - g t^2/2;
(* formula for the height of the rocket as a function of time *)
height = Z[ttop] // FullSimplify
```

Out[•]= $\frac{dt^2\ (F - g\ mass)^2}{2\ g\ mass^2}$

Lets setup the equations to be solved in terms of variables.
We will plug in the actual values at the end.
remember that the mass is doubled for height 2.

```
In[•]:= eq1 = h1 == height
```

Out[•]= $h1 == \frac{dt^2\ (F - g\ mass)^2}{2\ g\ mass^2}$

```
eq2 = h2 == height /. {mass → 2 * mass} (* mass is doubled *)
```

Out[•]= $h2 == \frac{dt^2\ (F - 2\ g\ mass)^2}{8\ g\ mass^2}$

Now solve the two equations for two unknowns. There should be 4 solutions !

```
In[•]:= solutions = Solve[eq1 && eq2, {dt, F}] // FullSimplify
```

$$\text{Out}[\bullet]= \left\{\left\{\text{dt} \to -\sqrt{2}\,\sqrt{\frac{\left(\sqrt{\text{h1}}-2\sqrt{\text{h2}}\right)^2}{\text{g}}},\ \text{F} \to \frac{2\,\text{g}\left(\text{h1}+\sqrt{\text{h1}}\,\sqrt{\text{h2}}-2\,\text{h2}\right)\,\text{mass}}{\text{h1}-4\,\text{h2}}\right\},\right.$$
$$\left\{\text{dt} \to \sqrt{2}\,\sqrt{\frac{\left(\sqrt{\text{h1}}-2\sqrt{\text{h2}}\right)^2}{\text{g}}},\ \text{F} \to \frac{2\,\text{g}\left(\text{h1}+\sqrt{\text{h1}}\,\sqrt{\text{h2}}-2\,\text{h2}\right)\,\text{mass}}{\text{h1}-4\,\text{h2}}\right\},$$
$$\left\{\text{dt} \to -\sqrt{2}\,\sqrt{\frac{\left(\sqrt{\text{h1}}+2\sqrt{\text{h2}}\right)^2}{\text{g}}},\ \text{F} \to \frac{2\,\text{g}\left(\sqrt{\text{h1}}+\sqrt{\text{h2}}\right)\,\text{mass}}{\sqrt{\text{h1}}+2\sqrt{\text{h2}}}\right\},$$
$$\left.\left\{\text{dt} \to \sqrt{2}\,\sqrt{\frac{\left(\sqrt{\text{h1}}+2\sqrt{\text{h2}}\right)^2}{\text{g}}},\ \text{F} \to \frac{2\,\text{g}\left(\sqrt{\text{h1}}+\sqrt{\text{h2}}\right)\,\text{mass}}{\sqrt{\text{h1}}+2\sqrt{\text{h2}}}\right\}\right\}$$

Now we plug in numbers and see which solution has the correct form.

```
In[•]:= finalanswer = solutions /. {g → 9.8, h1 → height1, h2 → height2, mass → rocketmass}
Out[•]= {{dt → -0.0203051, F → 9.84875}, {dt → 0.0203051, F → 9.84875},
  {dt → -4.02031, F → 0.147247}, {dt → 4.02031, F → 0.147247}}
```

Notice that two of the solutions have positive answers. This is because these are quadratic equations. Take the positive answer only. Only one of the positive solutions satisfies the condition that dt is very small. Therefore the correct answer is the second one. F is in Newtons and dt is in seconds.

Notice that it is possible to have no reasonable solution if height1, height2 are not in the correct ratio. Have fun thinking about this. It is amazing that we can infer short time bursts from a simple measurement.

Now what is the initial velocity obtained after the thrust. This will be in meters per second. and the time to the top in seconds.

```
In[•]:= InitialVelocity = v0 /. {g → 9.8, mass → rocketmass} /. finalanswer[[2]]
Out[•]= 19.799
```

In[•]:=
```
TimeToTop = ttop /. {g → 9.8, mass → rocketmass} /. finalanswer[[2]]
```

Out[•]=
```
2.02031
```

Let's now play with planet data that is in Mathematica. Much of this can be obtained by PlanetData["planet", "property"]. Get the details from Help.

In[•]:=
```
? PlanetData
```

PlanetData[*entity*, *property*] gives the value of the specified property for the planet *entity*.
PlanetData[{$entity_1$, $entity_2$, …}, *property*] gives a list of property values for the specified planet entities.
PlanetData[*entity*, *property*, *annotation*] gives the specified *annotation* associated with the property. ≫

In[•]:=
```
PlanetData["Jupiter", "Mass"]
```

Out[•]=
$$1.89813 \times 10^{27}\ \text{kg}$$

Let's make a table of "name", mass, diameter, orbit period, distancefromsun, and eccentricity, Pluto is no longer a planet !

In[•]:=
```
planets =
 {"Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"}
```

Out[•]=
```
{Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune}
```

In[•]:=
```
neededprop = {"Name", "Mass", "EquatorialDiameter",
    "OrbitPeriod", "DistanceFromSun", "Eccentricity"};
```

In[•]:=
```
data = Table[PlanetData[planets[[i]], neededprop[[j]]] , {i, 1, 8}, {j, 1, 6}]
```

Out[•]=
```
{{Mercury, 3.30104 × 10^23 kg , 3031.9 mi , 87.96926 days , 0.308307 au , 0.20563069},
 {Venus, 4.86732 × 10^24 kg , 7521.0 mi , 224.70080 days , 0.719164 au , 0.00677323},
 {Earth, 5.9721986 × 10^24 kg , 7926.3 mi , 365.25636 days , 1.0145 au , 0.016710220},
 {Mars, 6.41693 × 10^23 kg , 4220.6 mi , 1.8808476 a , 1.45024 au , 0.093412330},
 {Jupiter, 1.89813 × 10^27 kg , 88 846. mi , 11.862615 a , 5.40371 au , 0.048392660},
 {Saturn, 5.68319 × 10^26 kg , 74 898. mi , 29.447498 a , 10.0655 au , 0.0541506},
 {Uranus, 8.68103 × 10^25 kg , 31 763. mi , 84.016846 a , 19.8858 au , 0.047167710},
 {Neptune, 1.02410 × 10^26 kg , 30 775. mi , 164.79132 a , 29.942 au , 0.00858587}}
```

In[•]:=
```
Prepend[data, neededprop] // TableForm
```

Out[•]//TableForm=

Name	Mass	EquatorialDiameter	OrbitPeriod	DistanceFrom
Mercury	3.30104×10^{23} kg	3031.9 mi	87.96926 days	0.308311 au
Venus	4.86732×10^{24} kg	7521.0 mi	224.70080 days	0.719164 au
Earth	5.9721986×10^{24} kg	7926.3 mi	365.25636 days	1.01451 au
Mars	6.41693×10^{23} kg	4220.6 mi	1.8808476 a	1.45021 au
Jupiter	1.89813×10^{27} kg	88 846. mi	11.862615 a	5.40371 au
Saturn	5.68319×10^{26} kg	74 898. mi	29.447498 a	10.0655 au
Uranus	8.68103×10^{25} kg	31 763. mi	84.016846 a	19.8858 au
Neptune	1.02410×10^{26} kg	30 775. mi	164.79132 a	29.942 au

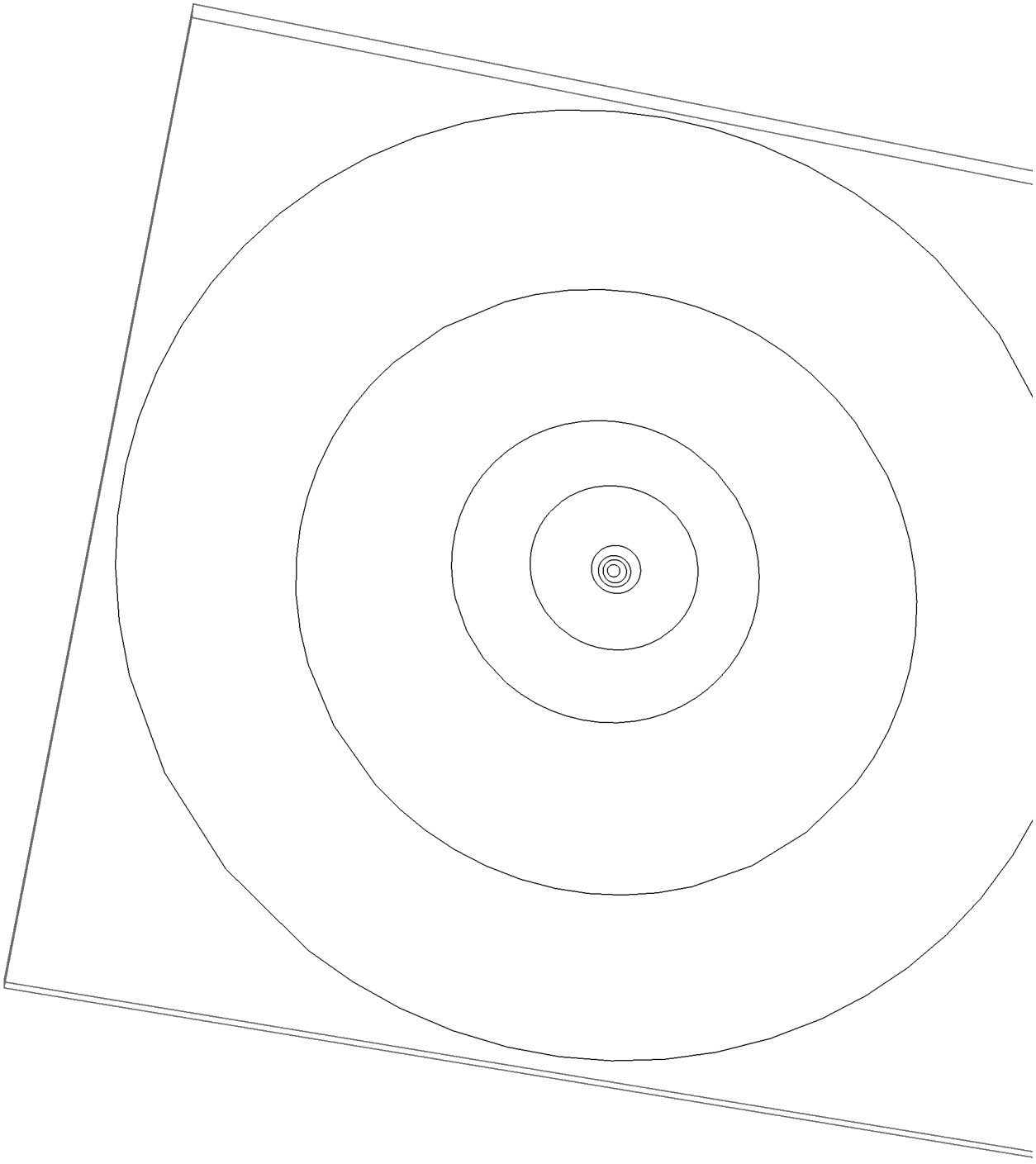
In[•]:=
```
PlanetData[]
```

Out[•]= { Mercury , Venus , Earth , Mars , Jupiter , Saturn , Uranus , Neptune }

I will leave it to you to convert units into units that you like.

In[•]:=
```
Graphics3D[PlanetData[PlanetData[], "OrbitPath"]]
```

Out[•]=



Before we finish let's learn how to do various operations

with lists.
Let's first define a list.

```
In[4]:= somenumbers = {2.0, 3.4, 4.5,  1.1, 1.2, 2.3, 10.3}
Out[4]= {2., 3.4, 4.5, 1.1, 1.2, 2.3, 10.3}

In[13]:= morenumbers = {4.5, 2.1, 90.0, 10^3, 4.3, 1.2, 0.34}
Out[13]= {4.5, 2.1, 90., 1000, 4.3, 1.2, 0.34}
```

Length of the list

```
In[5]:= listlength = Length[somenumbers]
Out[5]= 7
```

Join two lists.

```
In[14]:= Join[somenumbers, morenumbers]
Out[14]= {2., 3.4, 4.5, 1.1, 1.2, 2.3, 10.3, 4.5, 2.1, 90., 1000, 4.3, 1.2, 0.34}
```

make a list of lists

```
In[9]:= nestedlist  = {{2, 3}, {4, 5}, {5, 6}, {2.1, 1.1}}
Out[9]= {{2, 3}, {4, 5}, {5, 6}, {2.1, 1.1}}

In[10]:= Length[nestedlist]
Out[10]= 4
```

Flatten out the list

```
In[11]:= flatlist = Flatten[nestedlist]
Out[11]= {2, 3, 4, 5, 5, 6, 2.1, 1.1}

In[12]:= Length[flatlist]
Out[12]= 8
```

Some arithmatic operations on lists

```
In[15]:= addedlists = somenumbers + morenumbers
Out[15]= {6.5, 5.5, 94.5, 1001.1, 5.5, 3.5, 10.64}
```

In[16]:=
```
substracted = somenumbers - morenumbers
```
Out[16]=
```
{-2.5, 1.3, -85.5, -998.9, -3.1, 1.1, 9.96}
```

In[17]:=
```
squaredlist = somenumbers^2
```
Out[17]=
```
{4., 11.56, 20.25, 1.21, 1.44, 5.29, 106.09}
```

Map a function on a list. How about exponential ? Map[Exp,

In[19]:=
```
Map[Exp, somenumbers]
```
Out[19]=
```
{7.38906, 29.9641, 90.0171, 3.00417, 3.32012, 9.97418, 29732.6}
```

In[20]:=
```
Exp /@ somenumbers
```
Out[20]=
```
{7.38906, 29.9641, 90.0171, 3.00417, 3.32012, 9.97418, 29732.6}
```

In[21]:=
```
definedF[x_, y_] := x * y * Exp[x];
```

How do I apply this function to a list of two elements ?

In[22]:=
```
xylist = {3, 1}
```
Out[22]=
```
{3, 1}
```

In[23]:=
```
definedF @@ xylist
```
Out[23]=
$$3\, e^3$$

This mapping and applying can get very complicated. You have to try various ways. It is very helpful to make compact statement.

In[24]:=
```
nestedlist
```
Out[24]=
```
{{2, 3}, {4, 5}, {5, 6}, {2.1, 1.1}}
```

This will Map and apply to a nest list.

In[32]:=
```
Apply[definedF, nestedlist, 1]
```
Out[32]=
$$\{6\, e^2, 20\, e^4, 30\, e^5, 18.8639\}$$

In[33]:=
```
definedF @@@ nestedlist
```
Out[33]=
$$\{6\, e^2, 20\, e^4, 30\, e^5, 18.8639\}$$

This will Thread the function over two lists: taking a number from each and joining them into a function.

```
In[26]:= MapThread[definedF, {somenumbers, morenumbers}]
Out[26]= {66.5015, 213.944, 36 456.9, 3304.58, 17.1318, 27.5287, 104 124.}
```