Maple 9.5 Application Paper

Units and Dimensional Management

Maple provides the most comprehensive package in the software industry for managing units and dimensions. Problems in science and engineering can now be fully managed with appropriate dimensions in any modern unit system (and even some historical systems!), including MKS, FPS, CGS, Atomic and more. Over 500 standard units are recognized by Maple's Units package. A convenient dialog box located in the Edit menu converts quantities between unit systems automatically.

The Units package offers far more than simple conversions between units of various systems. It preserves the user's chosen units throughout complex computations. It knows the base dimensions of all standard quantities measured in science and engineering. Users also have the option to create their own units and dimensions.

The following techniques are highlighted:

- *Different unit systems, for example SI, CGS, MKS, FPS, etc.*
- *Definition of new unit systems*
- *Solving problems involving unit*

Maplesoft™

command the brilliance

Units and Dimensional Management

The intent of this application example is to illustrate Maple techniques in a real world application context. Maple is a general-purpose environment capable of solving problems in any field that depends on mathematics and data. This application illustrates one possibility for this particular field. Note that there are many options within the Maple system to optimize the computations for specific problems.

Introduction

Maple provides the most comprehensive package in the software industry for managing units and dimensions. Problems in science and engineering can now be fully managed with appropriate dimensions in any modern unit system (and even some historical systems!), including MKS, FPS, CGS, Atomic and more. Over 500 standard units are recognized by Maple's **Units** package. A convenient dialog box located in the **Edit** menu converts quantities between unit systems automatically.

The **Units** package offers far more than simple conversions between units of various systems. It preserves the user's chosen units throughout complex computations. It knows the base dimensions of all standard quantities measured in science and engineering. Users also have the option to create their own units and dimensions.

The following techniques are highlighted:

- Different unit systems, for example SI, CGS, MKS, FPS, etc.
- Definition of new unit systems
- Solving problems involving unit

Initialization

```
> restart:
  interface(warnlevel=0):
  interface(displayprecision=4):
  with(Units):
  with(Units[Natural]):
```

Overview of the Units Package

The **Natural** option tells Maple that the unit variables will be expressed as ordinary variables without special notation. Consequently, one should take care not to create a new user variable called km, or miles, etc. There is an option to work in a "protected" mode where such ambiguity is prevented.

Some simple examples

Consider the following definition for a length. Notice that Maple casts the answer in meters [m]. SI is the

default unit system used by the **Units** package.

```
> myDist := 4*ft + 3*inches;
```

$$myDist := \frac{6477}{5000}\,[m]$$

You can easily express the above in standard US (FPS - Foot Pound Second) system using the convert command

```
> convert( myDist, system, FPS);
  evalf( % );
```

$$\frac{17}{4}\,[ft]$$

$$4.250000000\ [ft]$$

The **Units** package knows that a $\frac{kg\ m}{s^2}$ is a **Newton**

```
> myForce := 5*g * 3 * m/s^2;
```

$$myForce := \frac{3}{200}\,[N]$$

Maple is smart enough to understand when units conflict -- for example, adding a length (ft) to a mass (kg) does not make sense, and Maple returns an error if the user tries to do so.

Which units systems and dimensions does the Units package know?

The 8 standard unit systems recognized by the **Units** package.

```
> GetSystems();
```

$$Atomic, CGS, EMU, ESU, FPS, MKS, MTS, SI$$

The **Units** package also knows the dimensions of all standard quantities, for example area, frequency, torque, etc.

```
> GetDimension( area );
  GetDimension( frequency );
  GetDimension( torque );
```

$$length^2$$

$$\frac{1}{time}$$

$$\frac{mass\ length\ \mathrm{length}(radius)}{time^2}$$

You can ask Maple to list the base quantities the Units package knows.

```
> GetDimensions();
```

absorbed_dose , *acceleration* , *action* , *amount_of_information* , *amount_of_substance* , *angular_acceleration* , *angular_jerk* , *angular_speed* , *area* , *currency* , *dose_equivalent* , *dynamic_viscosity* , *electric_capacitance* , *electric_charge* , *electric_conductance* , *electric_current* , *electric_dipole_moment* , *electric_displacement* , *electric_field_strength* , *electric_permittivity* , *electric_polarizability* , *electric_potential* , *electric_resistance* , *electric_resistivity* , *energy* , *exposure* , *force* , *frequency* , *heat_capacity* , *heat_insulation_coefficient* , *heat_transfer_coefficient* , *illuminance* , *jerk* , *kinematic_viscosity* , *length* , *linear_frequency* , *linear_mass_density* , *logarithmic_gain* , *luminous_flux* , *luminous_intensity* , *luminous_luminance* , *magnetic_dipole_moment* , *magnetic_flux* , *magnetic_flux_density* , *magnetic_inductance* , *magnetic_permeability* , *magnetic_polarizability* , *magnetizing_force* , *mass* , *mass_density* , *molar_electric_charge* , *moment_of_inertia* , *momentum* , *plane_angle* , *power* , *pressure* , *solid_angle* , *specific_heat_capacity* , *speed* , *surface_energy_density* , *surface_power_density* , *thermal_conductivity* , *thermodynamic_temperature* , *time* , *torque* , *volume* , *volume_flow*

What units does the **Units** package recognize? Specifiy the dimension, in this case "force", and you can find out using the **GetUnits** function.

```
> GetUnits( dimension=force );
```

dyne $_{standard}$, *gramforce* $_{standard}$, *kipf* $_{standard}$, *newton* $_{SI}$, *ounceforce* $_{standard}$, *pond* $_{standard}$, *poundal* $_{standard}$, *poundforce* $_{standard}$, *sthene* $_{standard}$, *tonforce* $_{standard}$

Here are all 569 units known by the package

```
> GetUnits();
```

British_thermal_unit $_{39degF}$, *British_thermal_unit* $_{59degF}$, *British_thermal_unit* $_{60degF}$, *British_thermal_unit* $_{IT}$, *British_thermal_unit* $_{mean}$, *British_thermal_unit* $_{thermochemical}$, *Calorie* $_{15degC}$, *Calorie* $_{20degC}$, *Calorie* $_{IT}$, *Calorie* $_{mean}$, *Calorie* $_{thermochemical}$, *Celsius_heat_unit* $_{15degC}$, *Celsius_heat_unit* $_{20degC}$, *Celsius_heat_unit* $_{IT}$, *Celsius_heat_unit* $_{mean}$, *Celsius_heat_unit* $_{thermochemical}$, *Hefner_unit* $_{standard}$, *Q_unit* $_{IT}$, *Q_unit* $_{US}$, *Q_unit* $_{mean}$, *Q_unit* $_{thermochemical}$, *Rvalue* $_{SI}$, *Rvalue* $_{standard}$, *Ufactor* $_{SI}$, *Ufactor* $_{standard}$, *abampere* $_{EMU}$, *abbe* $_{standard}$, *abcoulomb* $_{EMU}$, *abfarad* $_{EMU}$, *abhenry* $_{EMU}$, *abohm* $_{EMU}$, *absiemens* $_{EMU}$, *abtesla* $_{EMU}$, *abvolt* $_{EMU}$, *abwatt* $_{EMU}$, *abweber* $_{EMU}$, *acceleration_of_free_fall* $_{standard}$, *acre* $_{US_survey}$, *acre* $_{standard}$, *ampere* $_{SI}$, *angstrom* $_{standard}$, *apostilb* $_{standard}$, *are* $_{SI}$, *arpent* $_{PQ_area}$, *arpent* $_{PQ_length}$, *astronomical_unit* $_{standard}$, *atmosphere* $_{standard}$, *atmosphere* $_{technical}$, *atomic_mass_unit* $_{SI}$, *bar* $_{SI}$, *barge* $_{US_petroleum}$, *barleycorn* $_{US_survey}$, *barleycorn* $_{standard}$, *barn* $_{SI}$, *barrel* $_{UK}$, *barrel* $_{UK_petroleum}$, *barrel* $_{US_dry}$,

barrel $_{US\_liquid}$, barrel $_{US\_petroleum}$, barrel $_{beer}$, barrel $_{bulk}$, barrique $_{wine}$, barye $_{standard}$,
bath $_{Sacred}$, bath $_{Talmudic}$, becquerel $_{SI}$, beka $_{Sacred}$, beka $_{Talmudic}$, bel $_{standard}$, belshazzar $_{metric}$,
bicron $_{standard}$, biennium $_{Gregorian}$, biennium $_{tropical}$, biot $_{standard}$, bit $_{information}$,
bit_per_scond $_{information}$, blink $_{standard}$, block $_{information}$, bohr $_{Atomic}$, bottle $_{metric}$,
brewster $_{standard}$, bucket $_{UK}$, bucket $_{US\_liquid}$, bushel $_{UK}$, bushel $_{US\_dry}$, bushel $_{US\_dry\_heaped}$,
butt $_{UK}$, butt $_{beer}$, byte $_{information}$, cab $_{Sacred}$, cab $_{Talmudic}$, cable $_{UK\_nautical}$, cable $_{US\_nautical}$,
calibre $_{standard}$, calorie $_{15degC}$, calorie $_{20degC}$, calorie $_{IT}$, calorie $_{mean}$, calorie $_{nutrition}$,
calorie $_{thermochemical}$, candela $_{SI}$, candle $_{international}$, candle $_{standard}$, carat $_{diamond}$, carat $_{metric}$,
carcel $_{standard}$, cental $_{avoirdupois}$, centare $_{SI}$, century $_{Gregorian}$, century $_{tropical}$, chain $_{Gunter}$,
chain $_{Gunter\_US\_survey}$, chain $_{Ramsden}$, chain $_{Ramsden\_US\_survey}$, chain $_{UK\_nautical}$, circle $_{angle}$,
clo $_{standard}$, clove $_{avoirdupois}$, coombe $_{UK}$, coombe $_{US\_dry}$, cor $_{Sacred}$, cor $_{Talmudic}$, cord $_{UK}$,
coulomb $_{SI}$, crith $_{standard}$, crumb $_{information}$, cubic_foot_per_minute $_{standard}$, cubit $_{Sacred}$,
cubit $_{Talmudic}$, cubit $_{US\_survey}$, cubit $_{standard}$, cup $_{UK}$, cup $_{US\_liquid}$, curie $_{radiation}$, dalton $_{standard}$,
darcy $_{standard}$, dash $_{US\_liquid}$, day $_{SI}$, day $_{sidereal}$, debye $_{standard}$, decade $_{Gregorian}$, decade $_{tropical}$,
degree $_{Celsius}$, degree $_{Fahrenheit}$, degree $_{Rankine}$, degree $_{Reaumur}$, degree $_{angle}$, degree $_{centigrade}$,
denier $_{standard}$, dessertspoon $_{UK}$, dessertspoon $_{US\_liquid}$, digit $_{US\_survey}$, digit $_{standard}$,
diopter $_{standard}$, dollar $_{US}$, donkeypower $_{standard}$, double_word $_{information}$, dozen $_{Baker}$,
dozen $_{standard}$, drachm $_{UK}$, drachm $_{US\_liquid}$, drachm $_{apothecary}$, drachm $_{avoirdupois}$, drex $_{standard}$,
drop $_{UK}$, drop $_{US\_liquid}$, dyne $_{standard}$, electron $_{Atomic}$, electron_mass $_{Atomic}$, electronvolt $_{SI}$,
ell $_{US\_survey}$, ell $_{standard}$, eon $_{tropical}$, ephah $_{Sacred}$, ephah $_{Talmudic}$, erg $_{standard}$, farad $_{SI}$,
faraday $_{carbon12}$, faraday $_{chemical}$, faraday $_{physical}$, fathom $_{US\_survey}$, fathom $_{standard}$,
fermi $_{standard}$, fifth $_{US\_liquid}$, finger $_{US\_survey}$, finger $_{standard}$, firkin $_{beer}$, foot $_{PQ}$, foot $_{US\_survey}$,
foot $_{standard}$, foot_mercury $_{32degF}$, foot_mercury $_{60degF}$, foot_mercury $_{standard}$,
foot_water $_{39.2degF}$, foot_water $_{standard}$, footcandle $_{standard}$, footlambert $_{standard}$,
fortnight $_{standard}$, franklin $_{standard}$, fresnel $_{standard}$, furlong $_{US\_survey}$, furlong $_{standard}$,
gal $_{standard}$, gallon $_{UK}$, gallon $_{US\_dry}$, gallon $_{US\_liquid}$, gallon $_{beer}$, gallon $_{old\_CA}$,
gallon_per_minute $_{UK}$, gallon_per_minute $_{US\_dry}$, gallon_per_minute $_{US\_liquid}$, γ_{metric},
$\gamma_{standard}$, gauss $_{standard}$, gawble $_{information}$, gerah $_{Sacred}$, gerah $_{Talmudic}$, gilbert $_{standard}$, gill $_{UK}$,
gill $_{US\_liquid}$, gon $_{angle}$, grade $_{angle}$, grain $_{apothecary}$, grain $_{avoirdupois}$, grain $_{metric}$, grain $_{troy}$,
gram $_{SI}$, gramforce $_{standard}$, gray $_{SI}$, hand $_{US\_survey}$, hand $_{standard}$, handbreadth $_{Sacred}$,
handbreadth $_{Talmudic}$, hartree $_{Atomic}$, hectare $_{SI}$, hemisphere $_{angle}$, henry $_{SI}$, hertz $_{SI}$,
hin $_{Sacred}$, hin $_{Talmudic}$, hogshead $_{UK}$, hogshead $_{beer}$, horsepower $_{UK}$, horsepower $_{boiler}$,

horsepower$_{electric}$, horsepower$_{metric}$, horsepower$_{water}$, hour$_{SI}$, hour$_{angle}$, hour$_{sidereal}$, hubble$_{Julian}$, hubble$_{tropcial}$, hundredweight$_{long}$, hundredweight$_{metric}$, hundredweight$_{short}$, hyl$_{standard}$, inch$_{US_survey}$, inch$_{standard}$, inch_mercury$_{32degF}$, inch_mercury$_{60degF}$, inch_mercury$_{standard}$, inch_water$_{39.2degF}$, inch_water$_{standard}$, jansky$_{standard}$, jar$_{standard}$, jeroboam$_{metric}$, joule$_{SI}$, jupiter$_{standard}$, katal$_{SI}$, kayser$_{standard}$, kelvin$_{SI}$, kip$_{avoirdupois}$, kipf$_{standard}$, kipf_per_square_inch$_{standard}$, kitten$_{waterloo}$, knot$_{SI}$, kyne$_{standard}$, λ_{Volume}, lambert$_{standard}$, langley$_{energy}$, langley$_{power}$, last$_{UK}$, last$_{US_dry}$, league$_{US_survey}$, league$_{nautical}$, league$_{standard}$, lethech$_{Sacred}$, lethech$_{Talmudic}$, light_year$_{Julian}$, light_year$_{tropical}$, line$_{US_survey}$, line$_{standard}$, link$_{Gunter}$, link$_{Gunter_US_survey}$, link$_{Ramsden}$, link$_{Ramsden_US_survey}$, liter$_{SI}$, $\log_{Sacred}$, $\log_{Talmudic}$, lot$_{US_survey}$, lot$_{standard}$, lumen$_{SI}$, lune$_{lunar}$, lunour$_{lunar}$, lux$_{SI}$, mach$_{standard}$, magnum$_{metric}$, mancus$_{standard}$, marathon$_{standard}$, mast$_{standard}$, maxwell$_{standard}$, methusalah$_{metric}$, meter$_{SI}$, meter_mercury$_{0degC}$, meter_mercury$_{standard}$, meter_water$_{4degC}$, meter_water$_{standard}$, microinch$_{standard}$, micron$_{standard}$, mil$_{US_survey}$, mil$_{angle}$, mil$_{circular}$, mil$_{military}$, mil$_{standard}$, mile$_{US_survey}$, mile$_{nautical}$, mile$_{standard}$, mile_per_gallon$_{UK}$, mile_per_gallon$_{US_liquid}$, mile_per_hour$_{standard}$, millennium$_{Gregorian}$, millennium$_{tropical}$, millier$_{standard}$, mina$_{Sacred}$, mina$_{Talmudic}$, minute$_{SI}$, minute$_{angle}$, minute$_{sidereal}$, mite$_{standard}$, mole$_{SI}$, month$_{anomalistic}$, month$_{lunar}$, month$_{nodical}$, month$_{sidereal}$, nail$_{US_survey}$, nail$_{standard}$, nautical_mile$_{SI}$, nebuchadnezzar$_{metric}$, neper$_{SI}$, newton$_{SI}$, nibble$_{information}$, nickle$_{information}$, nit$_{standard}$, nox$_{standard}$, oersted$_{standard}$, ohm$_{SI}$, omer$_{Sacred}$, omer$_{Talmudic}$, ounce$_{UK}$, ounce$_{US_liquid}$, ounce$_{apothecary}$, ounce$_{avoirdupois}$, ounce$_{troy}$, ounceforce$_{standard}$, pace$_{US_survey}$, pace$_{standard}$, palm$_{US_survey}$, palm$_{standard}$, parsec$_{standard}$, part_per_billion$_{EU}$, part_per_billion$_{US}$, part_per_million$_{standard}$, pascal$_{SI}$, peck$_{UK}$, peck$_{US_dry}$, pennyweight$_{troy}$, pentad$_{standard}$, perch$_{PQ_area}$, perch$_{PQ_length}$, perch$_{US_survey}$, perch$_{standard}$, perm$_{0degC}$, perm$_{23degC}$, permittivity$_{Atomic}$, phot$_{standard}$, pica$_{TeX}$, pica$_{computer}$, pica$_{printer}$, pied_measure_planche$_{standard}$, pieze$_{standard}$, pim$_{Sacred}$, pim$_{Talmudic}$, pin$_{beer}$, pinch$_{US_liquid}$, pint$_{UK}$, pint$_{US_dry}$, pint$_{US_liquid}$, planck$_{Atomic}$, planck$_{standard}$, point$_{TeX}$, point$_{angle}$, point$_{computer}$, point$_{printer}$, poise$_{standard}$, pole$_{US_survey}$, pole$_{standard}$, poncelet$_{standard}$, pond$_{standard}$, pottle$_{UK}$, pottle$_{US_dry}$, pottle$_{US_liquid}$, pound$_{apothecary}$, pound$_{avoirdupois}$, pound$_{merchant}$, pound$_{troy}$, pound_per_square_foot$_{standard}$, pound_per_square_inch$_{standard}$, poundal$_{standard}$, poundforce$_{standard}$, preece$_{resistance}$, preece$_{resistivity}$, puncheon$_{beer}$, quad$_{II}$, quad$_{US}$, quad$_{mean}$, quad$_{thermochemical}$,

$$quadrant_{angle}, quadrant_{electrical}, quadrant_{standard}, quadrennium_{Gregorian},$$
$$quadrennium_{tropical}, quart_{UK}, quart_{US\_dry}, quart_{US\_liquid}, quarter_{avoirdupois},$$
$$quinquennium_{Gregorian}, quinquennium_{tropical}, quintal_{avoirdupois}, quintal_{metric}, rad_{radiation},$$
$$radian_{SI}, rehoboam_{metric}, rem_{radiation}, revolution_{angle}, revolution\_per\_minute_{angle},$$
$$reyn_{standard}, rhe_{standard}, rod_{US\_survey}, rod_{standard}, roentgen_{radiation}, rood_{US\_survey},$$
$$rood_{standard}, rutherford_{radiation}, rutherford_{standard}, sack_{UK}, sack_{US\_dry}, salmarazd_{metric},$$
$$scruple_{apothecary}, seah_{Sacred}, seah_{Talmudic}, seam_{UK}, seam_{US\_dry}, second_{Atomic}, second_{SI},$$
$$second_{angle}, second_{sidereal}, section_{US\_survey}, section_{standard}, semicircle_{angle}, shake_{standard},$$
$$shed_{standard}, shekel_{Sacred}, shekel_{Talmudic}, siemens_{SI}, sievert_{SI}, sign_{angle}, skot_{standard},$$
$$slug_{metric}, slug_{standard}, span_{Sacred}, span_{Talmudic}, span_{US\_survey}, span_{standard}, spat_{standard},$$
$$sphere_{angle}, statampere_{ESU}, statcoulomb_{ESU}, statfarad_{ESU}, stathenry_{ESU}, statohm_{ESU},$$
$$statsiemens_{ESU}, stattesla_{ESU}, statvolt_{ESU}, statwatt_{ESU}, statweber_{ESU}, step_{US\_survey},$$
$$step_{standard}, steradian_{SI}, stere_{standard}, sthene_{standard}, stilb_{standard}, stokes_{standard},$$
$$stone_{avoirdupois}, strike_{UK}, strike_{US\_dry}, strontium\_unit_{standard}, svedberg_{standard},$$
$$tablespoon_{AU}, tablespoon_{UK}, tablespoon_{US\_liquid}, talbot_{standard}, talent_{Sacred},$$
$$talent_{Talmudic}, teaspoon_{UK}, teaspoon_{US\_liquid}, tesla_{Atomic}, tesla_{SI}, tex_{standard}, therm_{EU},$$
$$therm_{IT}, therm_{US}, therm_{mean}, therm_{thermochemical}, tog_{standard}, ton_{TNT}, ton_{assay}, ton_{freight},$$
$$ton_{long}, ton_{long\_assay}, ton_{metric}, ton_{petroleum}, ton_{refrigeration}, ton_{register}, ton_{short},$$
$$tonforce_{standard}, tonne_{SI}, torr_{standard}, township_{US\_survey}, township_{standard},$$
$$triennium_{Gregorian}, triennium_{tropical}, tsubo_{JP}, tun_{beer}, unit\_pole_{standard}, vac_{standard},$$
$$volt_{Atomic}, volt_{SI}, watt_{SI}, watt\_hour_{standard}, weber_{SI}, week_{standard}, wey_{avoirdupois},$$
$$wink_{standard}, word_{information}, x\_unit_{copper}, x\_unit_{molybdenum}, yard_{US\_survey}, yard_{standard},$$
$$year_{Gregorian}, year_{Julian}, year_{Orthodox}, year_{anomalistic}, year_{ellipse}, year_{galactic}, year_{leap},$$
$$year_{mean}, year_{sidereal}, year_{standard}, year_{tropical}$$

Adding your own units

You can extend the set of units recognized by the **Units** package. In this example, we use the **Units** package to encapsulate the following information about army groups:

1 platoon = 24 soldiers
1 company = 3 platoons
1 battalion = 3 companies
1 brigade = 3 battalions
1 division = 3 brigades
1 corps = 3 divisions

We reinitialize the **Units** environment.

```
> restart:
```

```
with(Units):
```
Define the army groups as units.
```
> AddBaseUnit(soldier, context = military, dimension = humanlife,
      spellings = soldiers);
  AddUnit(platoon, context = military, conversion = 24 * soldier,
      spellings = platoons);
  AdddUnit(company, context = military, conversion = 3 * platoon,
      spellings = companies);
  AddUnit(battalion, context = military, conversion = 3 * company,
      spellings = battalions);
  AddUnit(brigade, context = military, conversion = 3 * battalion,
      spellings = brigades);
  AddUnit(corps, context = military, conversion = 3 * brigade,
      spellings = corps);
```

Now we add the "military" system to the 8 systems already recognized by the package.
```
> AddSystem(military, check, soldier);
  with(Units[Natural]):
  UseSystem( military ):
```
How many soldiers are in 5 battalions?
```
> convert(5*battalion, units, soldier);
```
$$1080\ [\mathit{soldier}\ ]$$

After a fierce battle, your fighting forces are down to 2 corps and a half-strength company. How many soldiers do you have left?
```
> 2*corps + (1/2)*company;
```
$$3924\ [\mathit{soldier}\ ]$$

How many platoons is that?
```
> evalf[4](convert(%, units, platoon));
```
$$163.5\ [\mathit{platoon}\ ]$$

Example Problems Involving Units

Reinitialize
```
> restart;
  with(Units[Natural]):
  with (ScientificConstants):
```

How much room does 1 billion dollars worth of gold take up?
First, define the dollar value of the gold, cost per troy ounce (assume $268), and the density.
```
> val := 10^9 * USD;
  costPerOz := 268.00 * USD/oz[troy];
  density := 19.3 * g/cm^3;
```
$$\mathit{val} := 1000000000\ [\mathit{USD}\ ]$$

$$\mathit{costPerOz} := 8616.400080\ \left[\frac{\mathit{USD}}{\mathit{kg}}\right]$$

$$\mathit{density} := 19300.00000\ \left[\frac{\mathit{kg}}{m^3}\right]$$

The resulting mass and volume are:
```
> mass := val/costPerOz;
  volume := mass/density;
```

$$mass := 116057.7493 \ [kg]$$

$$volume := 6.013354886 \ [m^3]$$

What is the length of one side of a cube having this volume?

```
> length_side := root[3](volume);
```

$$length\_side := 1.818467785 \ [m]$$

What is the sin of 45 degrees?

Users might initially try the following (which is incorrect):

```
> evalf(sin(45));
```

$$0.8509035245$$

The correct but tedious way **without** the **Units** package.

```
> evalf(sin((Pi/180)*45));
```

$$0.7071067810$$

With Maple's Units package, you can simply state **45*degrees**, and functions will understand how to manage the information. This one example will likely save many a high school student at least 20 minutes of grief.

```
> evalf(sin(45*degrees));
```

$$0.7071067810$$

How many seconds in 3 weeks?

```
> t := 3 * week;
```

$$t := 3 \ [wk]$$

```
> convert( t, units, s);
```

$$1814400 \ [s]$$

How far do you travel in 35 minutes going 55 miles per hour? Express answer in meters and miles.

```
> d := 55*mph * 35*minutes;
```

$$d := \frac{1290828}{25} \ [m]$$

```
> convert( d, units, mi );
```

$$\frac{385}{12} \ [mi]$$

```
> evalf( d );
```

$$51633.12000 \ [m]$$

Given a molar energy, find the mass energy in Btu's per pound.

```
> molar_energy := 523.432*Btu/mol(carbon);
```

$$molar\_energy := 551880.6676 \left[\frac{J}{\mathrm{mol}(carbon)} \right]$$

```
> carbon_mass := 12*kg/mol(carbon);
```

$$carbon\_mass := 12 \left[\frac{kg}{\mathrm{mol}(carbon)} \right]$$

```
> mass_energy := molar_energy / carbon_mass;
```

$$mass\_energy := 45990.05563 \left[\frac{m^2}{s^2}\right]$$

```
> convert(%, units, 'Btu/lb');
```

$$19.78539678 \left[\frac{Btu}{lb}\right]$$

Given a torque of 3 newton meters, how much energy is required to move a lever through 10 degrees?

```
> energy := torque * angle;
```

$$energy := torque\ angle$$

```
> eval(energy, [torque = 3*N*m(radius), angle = 10*deg]);
```

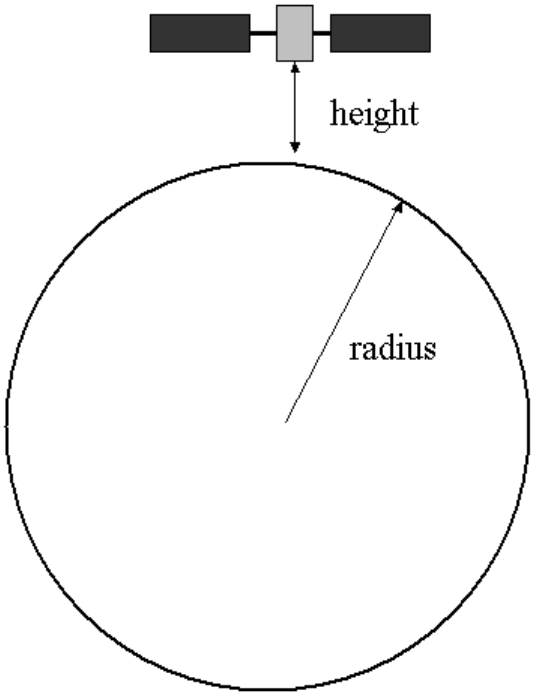
$$\frac{1}{6}\pi\,[J]$$

```
> evalf(%);
```

$$0.5235987758\ [J]$$

What is the force of gravity on a space vehicle

What is the gravitational force on a 500lb satellite orbiting 100 miles above ground.



We have the following definitions:

m1	mass of satellite
m2	mass of earth
G	gravitational constant
height	height of the satellite
radius	radius of earth (given as: 25000*5280 feet is the circumference of earth)

```
> m1 := 100*lb;
  m2 := 6.0*10^24*kg;
  G_const := Constant (G, units);
  height := 100*mile;
  radius := 25000*5280*ft/(2*Pi);
```

$$m1 := 100\ [lb]$$

$$m2 := 0.6000000000\ \ 10^{25}\ [kg]$$

$$G\_const := Constant_{SI}(G)\left[\frac{m^3}{kg\ s^2}\right]$$

$$height := 100\ [mi]$$

$$radius := 66000000\ \frac{[ft]}{\pi}$$

Distance from the center of mass of the satellite to the center of mass of earth.

```
> dist := radius+height;
```

$$dist := \left(\frac{20116800}{\pi} + \frac{804672}{5}\right)[m]$$

```
> Force := simplify (G_const*m1*m2/dist^2);
```

$$Force := 421.4633896\ \ [N]$$

What is the energy of an electron?

```
> mass := 9.11e-31 * kg;
  c := 3.00e8 * m/s;
  E := mass * c^2;
```

$$mass := 0.911\ \ 10^{-30}\ [kg]$$

$$c := 0.300\ \ 10^{9}\left[\frac{m}{s}\right]$$

$$E := 0.81990000\ \ 10^{-13}\ [J]$$

For a given phenomena with a frequency of 1.42... GHz, what are:

1. The period,
2. The number of cycles per year, and
3. The number of cycles since the beginning of the earth

```
> frequency := 1.420405761 * GHz;
```

$$frequency := 1.420405761\ \ [GHz]$$

The period in seconds and nano seconds.

```
> 1/frequency;
  convert(%, units, nanoseconds);
```

$$0.7040241792\ \ 10^{-9}\ [s]$$

$$0.7040241792\ [ns]$$

Cycles per year:

```
> convert(frequency, units, '1/yr');
```

$$0.4482363838\ 10^{17}\left[\frac{1}{yr}\right]$$

Given inductance and capacitance, find the resistance in microohms.

```
> resistance := sqrt(inductance / capacitance);
```

$$resistance := \sqrt{\frac{inductance}{capacitance}}$$

```
> eval(resistance, [inductance = 124.*nH, capacitance = 3.52 * uF]):
  evalf(%);
```

$$0.1876892984\ [\Omega]$$

```
> convert(%, units, uOmega);
```

$$187689.2984\ [uOmega]$$

Given an power gain from 332 microwatts to 23 milliwatts, what is the gain in decibels? What would the gain have been had the increase had been a voltage increase?

Current gain.

```
> gain := 23*mW / (332*uW(base));
```

$$gain := \frac{5750}{83}\left[\frac{W}{W(base)}\right]$$

```
> convert(ln (gain), units, dB):
  evalf (%);
```

$$18.40589752\ [dB]$$

Voltage gain.

```
> voltage_gain := 23*mV / (332*uV(base));
```

$$voltage\_gain := \frac{5750}{83}\left[\frac{V}{V(base)}\right]$$

```
> convert(ln(voltage_gain), units, dB):
  evalf (%);
```

$$36.81179504\ [dB]$$