The LUA-PHYSICAL library

Version 0.1

Thomas Jenni

March 17, 2019

Abstract

`lua-physical` is a pure Lua library which provides functions and object for doing computation with physical quantities. It has been written to simplify the creation of physics problem sets. The package provides units of the SI and the imperial system. Furthermore an almost complete set of international currencies are supported, however without online exchange rates. In order to display the numbers with measurement uncertainties, the package is able to perform gaussian error propagation.

Contents

1 Introduction

The author of this package is a teacher at the high school *Kantonsschule Zug* in Switzerland. The main use of this package is to write physics problem sets. LuaLaTeXdoes make it possible to integrate physical calculations directly into the LuaLaTeXfile. The package has been in use since 2016. Since then many bugs have been found and crushed. Nevertheless it still could be possible, that some were not found. Therefore the author recommends not to use this package industry or science. If one does so, it's the responsability of the user to check results for plausability. If the user finds some bugs, they can be reported at github.com or directly to the author (`thomas.jenni(at)ksz.ch`).

2 Loading

This package is a pure Lua library. Therefore one has to require it explicitly by calling `require("physical")`. For printing calculation results the `siunitx` package can be used. It's recommended to define a macro like `\q` to convert the lua quantity object to a `siunitx` expression.
The following Latex preambel loads the `lua-physical` package and creates a macro `\q` for printing physical quantities.

Listing 1: basic preamble

```latex
\usepackage{lua-physical}
\usepackage{siunitx}

% configure siunitx
\sisetup{
  output-decimal-marker = {.},
  per-mode = symbol,
  separate-uncertainty = true,
  add-decimal-zero = true,
  exponent-product = \cdot,
  round-mode = off
}

% load lua-physical package
\begin{luacode*}
  physical = require("physical")

  _N = physical.Number
\end{luacode*}

% print physical quantities
\newcommand{\q}[1]{%
  \directlua{tex.print(physical.Quantity.tosiunitx(#1,"scientific-notation=fixed,exponent-to-prefix=false"))}%
}
```

2.1 Dependencies

This package is standalone. If a pretty print to LuaLaTeX is wanted, the package `siunitx` sould be installed.

3 Usage

Given the basic preamble, units can be used in lua code directly. By convention, all units have an underscore in front of them, i.e. Meter is `_m`, Second is `_s`. All available units are listed in chapter 4. The Result of the calculation can be printed to LuaLaTeX by using the macro `\q{}`.

Listing 2: The velocity of a car.

```
\begin{luacode}
  s = 10 * _m
  t = 2 * _s
  v = s/t
\end{luacode}

A car travels $\q{s}$ in $\q{t}$. calculate its velocity.
$$
  v=\frac{s}{t} = \frac{\q{s}}{\q{t}} = \q{v}
$$
```

A car travels 10 m in 2 s. Calculate its velocity.

$$v = \frac{s}{t} = \frac{10\,\mathrm{m}}{2\,\mathrm{s}} = 5\,\mathrm{m/s}$$

In the above listing 2, the variable `s` stands for displacement and has the unit meter `_m`. The variable `t` stands for time and is given in second `_s`. By executing mathematical operations on them, new physical quantities are created. In the problem above, the velocity `v` is calculated by dividing `s` by `t`. The created instance `v` has the derived unit m/s. By using the macro `\q{}` all quantities can be printed to the LuaLaTeX code.

3.1 Unit conversion

Very often the result of a calculation is needed in different physical unit, than the given quantities. In the following listing 3 the task is to calculate the volume of a cuboid with lengths given in different units. If the volume is calculated by multipling all three lengths, the unit of the result is cm mm m. If the unit cm^3 is preferred, it has to be converted explicitly. The conversion function is called `to()` and is available on all physical quantitiy instances. At first this looks a bit cumbersome. The reason of this behaviour is, that the software is not able to guess the unit of the result. In many cases, like in the example here, it's not clear what unit the result sould have. Therefore the user has always to give the target unit explicitly.

Listing 3: The volume of a cuboid.

```
\begin{luacode}
  a = 12 * _cm
  b = 150 * _mm
  c = 1.5 * _m

  V = a*b*c
\end{luacode}

Find the volume of a rectangular cuboid with lengths $\q{a}$,
$\q{b}$ and $\q{c}$.
$$
  V= a \cdot b \cdot c
  = \q{a} \cdot \q{b} \cdot \q{c}
  = \q{V}
  = \uuline{\q{V:to(_dm^3)}}
$$
```

Find the volume of a rectangular cuboid with lengths 12 cm, 150 mm and 1.5 m.

$$V = a \cdot b \cdot c = 12\,\mathrm{cm} \cdot 150\,\mathrm{mm} \cdot 1.5\,\mathrm{m} = 2700\,\mathrm{cm\,mm\,m} = \underline{\underline{27\,\mathrm{dm}^3}}$$

3.1.1 Temperature Conversion

In the following problem, listing 4 , the task is to convert a temperature given in the unit degree Celsius to Kelvin. As can be seen in the listing, the conversion function has two parameters.

The first argument is the target unit. The second is a boolean that tells the `to`-function to call a unit specific conversion function. By default the second argument is `false`.

Most units do not have a conversion function. Exceptions are the unit degree Celsius `_degC` and degree Fahrenheit `_degF`. These units are ambigous and can be interpreted as temperature differences or as an absolute temperatures. In the latter case, the conversion to base units is not a linear, but an affine transformation. This is because degree Celsius and degree Fahrenheit scales have their zero points at different temperatures compared to the unit Kelvin. Therefore these units have their own conversion functions.

By default `_degC` and `_degF` units are standing for temperature differences. If one wants to have it converted absolutely, the conversion function `to()` should have the second argument set to true.

Listing 4: Temperature conversion.

```
\begin{luacode}
```

```
  T = 20 * _degC
\end{luacode}

A thermometer shows $\q{T}$. Converte this quantity to Kelvin.
$$
  T = \q{T:to(_K)} + \q{273.15 * _K}
  =  \q{T:to(_K,true)}
$$
```

A thermometer shows 20 °C. Converte this quantity to Kelvin.

$$T = 20\,\mathrm{K} + 273.15\,\mathrm{K} = 293.15\,\mathrm{K}$$

3.1.2 Uncertainty

The package supports uncertainty propagation. To create a number with an uncertainty, an instance of `physical.Number` has to be created, see listing 5. It has to be remembered, that `N` is a alias for `physical.Number`. The first argument of the constructor `N(mean, uncertainty)` is the mean value and the second one the uncertainty of the measurement. If the proposed preamble is used, the uncertainty is by default seperated from the mean value by a plus-minus sign.
For the uncertainty propagation the gaussian formula

$$\Delta f = \sqrt{\left(\frac{\partial f}{x_1} \cdot \Delta x_1\right)^2 + \ldots + \left(\frac{\partial f}{x_n} \cdot \Delta x_2\right)^2}$$

is used. This formula is a good estimation for the uncertainty Δf, if the quantities $x_1, \ldots, x_n$ the function f depends on, have no correlation. Further, the function f has to behave linear, if the quantities x_i are changed in the range of their uncertainties.

Listing 5: Uncertainty in area calculation.

```
\begin{luacode}
  a = N(2,0.1) * _m
  b = N(3,0.1) * _m

  A = (a*b):to(_m^2)
\end{luacode}

Calculate the area of a rectangle with lengths $\q{a}$ and $\q{b}$.
$$
  A = a \cdot b
  = \q{a} \cdot \q{b}
  = \uuline{\q{A}}
$$
```

Calculate the area of a rectangle with lengths $(2.00 \pm 0.10)\,\mathrm{m}$ and $(3.00 \pm 0.10)\,\mathrm{m}$.

$$A = a \cdot b = (2.00 \pm 0.10)\,\mathrm{m} \cdot (3.00 \pm 0.10)\,\mathrm{m} = \underline{\underline{(6.0 \pm 0.4)\,\mathrm{m}^2}}$$

Instead of printing always the uncertainties, one can use the uncertainty calculation to provide significant numbers.
In the following problem, listing 6 , the task is to calculate the volume of an ideal gas. Given are pressure `p` in `_bar`, amount of substance `n` in `_mol` and temperature `T` in degree celsius `_degC`. In order to do the calculation, one has to convert `T`, which is given as an absolute temperature in degree celsius to the base unit Kelvin first. By setting `N.omitUncertainty = true`, all uncertainties are not printed.

Listing 6: Volume of an ideal gas.

```
\begin{luacode}
  N.omitUncertainty = true
  p = N(1.013,0.0001) * _bar
  n = N(1,0.01) * _mol
  T = N(30,0.1) * _degC

  V = ( n * _R * T:to(_K,true) / p ):to(_L)
\end{luacode}

An ideal gas ($\q{n}$) has a pressure of $\q{p}$ and a temperature of $\q{T}$. Calculate the volume of the gas.
$$
  V=\frac{ \q{n} \cdot \q{_R} \cdot \q{T:to(_K,true)} }{ \q{p} }
  = \q{V}
  = \uuline{\q{V}}
$$
```

An ideal gas ($1.0\,\mathrm{mol}$) has a pressure of $1.013\,\mathrm{bar}$ and a temperature of $30\,^\circ\mathrm{C}$. Calculate the volume of the gas.

$$V = \frac{1.0\,\mathrm{mol} \cdot 8.31\,\mathrm{J/(mol\,K)} \cdot 303\,\mathrm{K}}{1.013\,\mathrm{bar}} = \underline{\underline{25\,\mathrm{L}}}$$

4 Supported Units

In this chapter, all supported units are listed.

4.1 Base Units

From the units listed in this section, all other units are derived from.

Quantity	Unit	Symbol	Dim.	Definition
length	meter	`_m`	L	The distance light travels in vacuum during 1/299 792 458 second.
mass	kilogram	`_kg`	M	The mass of the international protoype of the kilogram.
time	second	`_s`	T	Is 9 192 631 770 times the period of the radiation from the transition between the two hyperfine levels of the ground state of caesium-133.
electric current	ampere	`_A`	I	The constant current which, if maintained in two straight parallel conductors of infinte length, of negligible circular crosss-section, and placed 1 m apart in vacuum, would produce between these conductors a force equal to $2 \cdot 10^{-7}$ N/m.
thermodynamic temperature	kelvin	`_K`	Θ	Is the fraction 1/273.16 of the thermodynamic temperature of the triple point of water.
amount of substance	mole	`_mol`	N	Amount of substance that contains as many particles as there are atoms in 0.012 kg of carbon-12.
luminous intensity	candela	`_cd`	J	The luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequenc $540 \cdot 10^{12}$ Hz and has a radiant intensity in that direction of (1/683) W/sr

Quantity	Unit	Symbol	Dim.	Definition
number	–	`_1`	1	The dimensionless number one.
information	bit	`_bit`	B	The smallest amount of information.
currency	euro	`_EUR`	L	The value of the currency Euro.

Table 1: Base units of the International System of Units (SI), information, currency and the dimensionless number one.

4.2 Derived Units of the International System of Units (SI)

All units in this section are derived from the base units.

Quantity	Unit	Symbol	Dimension	Definition
plane angle	radian	`_rad`	1	`_1`
solid angle	steradian	`_sr`	1	`_rad^2`
frequency	hertz	`_Hz`	T^{-1}	`1/_s`
force	newton	`_N`	$\mathrm{M\,L\,T}^{-2}$	`_kg*_m/_s^2`
pressure	pascal	`_Pa`	$\mathrm{M\,L}^{-1}\,\mathrm{T}^{-2}$	`_N/_m^2`
energy	joule	`_J`	$\mathrm{M\,L}^{2}\,\mathrm{T}^{-2}$	`_N*_m`
power	watt	`_W`	$\mathrm{M\,L}^{2}\,\mathrm{T}^{-3}$	`_J/_s`
electric charge	coulomb	`_C`	$\mathrm{T\,I}$	`_A*_s`
electric potential difference	volt	`_V`	$\mathrm{M\,L}^{2}\,\mathrm{T}^{-3}\,\mathrm{I}^{-1}$	`_J/_C`
capacitance	farad	`_F`	$\mathrm{L}^{-2}\,\mathrm{M}^{-1}\,\mathrm{T}^{4}\,\mathrm{I}^{2}$	`_C/_V`
electric resistance	ohm	`_Ohm`	$\mathrm{L}^{2}\,M\mathrm{T}^{-3}\,\mathrm{I}^{-2}$	`_V/_A`
electric conductance	siemens	`_S`	$\mathrm{L}^{-2}\,\mathrm{M}^{-1}\,T^{3}I^{2}$	`_A/_V`
magnetic flux	weber	`_Wb`	$\mathrm{L}^{2}\,\mathrm{M\,T}^{-2}\,\mathrm{I}^{-1}$	`_V*_s`
magnetic flux density	tesla	`_T`	$\mathrm{M\,T}^{-2}\,\mathrm{I}^{-1}$	`_V*_s`
inductance	henry	`_H`	$\mathrm{L}^{2}\,\mathrm{M\,T}^{-2}\,\mathrm{I}^{-2}$	`_Wb/_A`
Celsius temperature	degree Celsius	`_degC`	Θ	`_K`

Quantity	Unit	Symbol	Dimension	Definition
luminous flux	lumen	`_lm`	J	`_cd*_sr`
illuminance	lux	`_lux`	L^{-2} J	`_lm/_m^2`
activity	becquerel	`_Bq`	T^{-1}	`1/_s`
absorbed dose	gray	`_Gy`	L^2 T^{-2}	`_J/_kg`
dose equivalent	sievert	`_Sv`	L^2 T^{-2}	`_J/_kg`
catalytic activity	katal	`_kat`	T^{-1} N	`_mol/_s`

Table 2: Derived units of the International System of Units (SI)

4.3 Units outside of the International System of Units (SI)

There are a few units with dimension 1. The unit Bel is only available with prefix decibel, because `_B` is the unit byte.

Quantity	Unit	Symbol	Dimension	Definition
	percent %	`_percent`	1	`1e-2*_1`
	permille ‰	`_permille`	1	`1e-3*_1`
	parts-per-million	`_ppm`	1	`1e-6*_1`
	parts-per-billion	`_ppb`	1	`1e-9*_1`
	parts-per-trillion	`_ppt`	1	`1e-12*_1`
	parts-per-quadrillion	`_ppq`	1	`1e-15*_1`
	decibel	`_dB`	1	`_1`
plane angle	degree	`_deg`	1	`(Pi/180)*_rad`
	arc minute	`_arcmin`	1	`_deg/60`
	arc second	`_arcsec`	1	`_arcmin/60`
	gradian	`_gon`	1	`(Pi/200)*_rad`
	turn	`_tr`	1	`2*Pi*_rad`
solid angle	spat	`_sp`	1	`4*Pi*_sr`
length	astronomical unit	`_au`	L	`149597870700*_m`
	lightyear	`_ly`	L	`_c*_a`
	parsec	`_pc`	L	`(648000/Pi)*_au`
	angstrom	`_angstrom`	L	`1e-10*_m`
	fermi	`_fermi`	L	`1e-15*_m`
area	are	`_ar`	L^2	`1e2*_m^2`
	hectare	`_hectare`	L^2	`1e4*_m^2`
	barn	`_barn`	L^2	`1e-28*_m^2`
volume	liter	`_L`	L^3	`0.001*_m^3`
	metric teaspoon	`_tsp`	L^3	`0.005*_L`
	metric tablespoon	`_Tbsp`	L^3	`3*_tsp`

Quantity	Unit	Symbol	Dimension	Definition
time	minute	`_min`	T	`_60*_s`
	hour	`_h`	T	`_60*_min`
	day	`_d`	T	`_24*_h`
	week	`_wk`	T	`_7*_d`
	year	`_a`	T	`365.25*_d`
	svedberg	`_svedberg`	T	`1e-13*_s`
mass	tonne	`_t`	M	`1000*_kg`

Table 3: Units outside of the International System of Units (SI)

4.4 Imperial Units

Quantity	Unit	Symbol	Dimension	Definition
length	inch	_in	L	0.0254*_m
	thou	_th	L	0.001*_in
	pica	_pica	L	_in/6
	point	_pt	L	_in/72
	hand	_hh	L	4*_in
	foot	_ft	L	12*_in
	yard	_yd	L	3*_ft
	rod	_rd	L	5.5*_yd
	chain	_ch	L	4*_rd
	furlong	_fur	L	10*_ch
	mile	_mi	L	8*_fur
	league	_lea	L	3*_mi
	nautical mile	_nmi	L	1852 * _m
	nautical league	_nlea	L	3*_nmi
	cable	_cbl	L	_nmi/10
	fathom	_ftm	L	6*_ft
velocity	knot	_kn	$L^1 T^{-1}$	_nmi/_h
area	acre	_ac	L^2	43560*_ft^2
volume	gallon	_gal	L^3	4.54609*_L
	quart	_qt	L^3	_gal/4
	pint	_pint	L^3	_qt/2
	cup	_cup	L^3	_pint/2
	gill	_gi	L^3	_pint/4
	fluid ounce	_fl_oz	L^3	_gi/5
	fluid dram	_fl_dr	L^3	_fl_oz/8

Quantity	Unit	Symbol	Dimension	Definition
mass	grain	`_gr`	M	`64.79891*_mg`
	pound	`_lb`	M	`7000*_gr`
	ounce	`_oz`	M	`_lb/16`
	dram	`_dr`	M	`_lb/256`
	stone	`_st`	M	`14*_lb`
	quarter	`_qtr`	M	`2*_st`
	hundredweight	`_cwt`	M	`4*_qtr`
	long ton	`_ton`	M	`20*_cwt`

Table 4: Imperial units

4.5 U.S. customary units

In the U.S., the length units are bound to the meter differently than in the imperial system. The followin definitions are taken from `https://en.wikipedia.org/wiki/United_States_customary_units`.

Quantity	Unit	Symbol	Dimension	Definition
length	U.S. survey inch	`_in_US`	L	`_m/39.37`
	U.S. survey hand	`_hh_US`	L	`4*_in_US`
	U.S. survey foot	`_ft_US`	L	`3*_hh_US`
	U.S. survey link	`_li_US`	L	`0.66*_ft_US`
	U.S. survey yard	`_yd_US`	L	`3*_ft_US`
	U.S. survey rod	`_rd_US`	L	`5.5*_yd_US`
	U.S. survey chain	`_ch_US`	L	`4*_rd_US`
	U.S. survey furlong	`_fur_US`	L	`10*_ch_US`
	U.S. survey mile	`_mi_US`	L	`8*_fur_US`
	U.S. survey league	`_lea_US`	L	`3*_mi_US`
	U.S. survey fathom	`_ftm_US`	L	`72*_in_US`
	U.S. survey cable	`_cbl_US`	L	`120*_ftm_US`
area	U.S. acre	`_ac_US`	L^2	`_ch_US*_fur_US`
volume	U.S. gallon	`_gal_US`	L^3	`231*_in^3`
	U.S. quart	`_qt_US`	L^3	`_gal_US/4`
	U.S. pint	`_pint_US`	L^3	`_qt_US/2`
	U.S. cup	`_pint_US`	L^3	`_pint_US/2`
	U.S. gill	`_gi_US`	L^3	`_pint_US/4`
	U.S. fluid ounce	`_fl_oz_US`	L^3	`_gi_US/4`
	U.S. table spoon	`_Tbsp_US`	L^3	`_fl_oz_US/2`
	U.S. tea spoon	`_tsp_US`	L^3	`_Tbsp_US/3`
	U.S. fluid dram	`_fl_dr_US`	L^3	`_fl_oz_US/8`

Quantity	Unit	Symbol	Dimension	Definition
mass	U.S. quarter	`_qtr_US`	L^3	`25*_lb`
	U.S. hundredweight	`_qtr_US`	L^3	`4*_qtr_US`
	U.S. short ton	`_ton_US`	L^3	`20*_cwt_US`

Table 5: U.S. customary units

4.6 International Currencies

Quantity	Unit	Symbol	Dimension	Definition
currency	Afghan afghani	`_AFN`	C	`0.012*_EUR`
	Albanian lek	`_ALL`	C	`0.008*_EUR`
	Armenian Dram	`_AMD`	C	`0.0018*_EUR`
	Angolan Kwanza	`_AOA`	C	`0.0028*_EUR`
	Argentine Peso	`_ARS`	C	`0.021*_EUR`
	U.S. dollar	`_USD`	C	`0.89*_EUR`
	Japanese yen	`_JPY`	C	`0.008*_EUR`
	British pound	`_GBP`	C	`1.17*_EUR`
	Australian dollar	`_AUD`	C	`0.63*_EUR`
	Canadian dollar	`_CAD`	C	`0.66*_EUR`
	Swiss franc	`_CHF`	C	`0.88*_EUR`
	Chinese yuan	`_CNY`	C	`0.13*_EUR`
	Swedish krona	`_SEK`	C	`0.094*_EUR`
	New Zealand dollar	`_NZD`	C	`0.60*_EUR`

Table 6: Currency units based on exchange rates from 7.3.2019, 21:00 UTC.

5 Lua Documentation

In this chapter, the following shortcuts will be used.

```
local D = physical.Dimension
local U = physical.Unit
local N = physical.Number
local Q = physical.Quantity
```

The term `number` refers to a lua integer or a lua float number. By `string` a lua string is meant and by `bool` a lua boolean.

5.1 physical.Quantity

The quantity class is the main part of the library. Each physical Quantity and all units are represented by an instance of this class.

`Q.new(q=nil)`

Copy Constuctor

Parameters

`q` : `Q` or `number`, optional
    Optional argument is either `Q`, a `number` or `nil`.

`return` : `Q`
    The created `Q` instance

Note
As an argument it takes `Q`, `number` or `nil`. If `Q` is given, a copy of it is made and returned. If a `number` is given, the function creates a dimeensionless quantity with that value. In the case `nil` is given, the quantity `_1` is returned.

Example

```
myOne = Q()
myNumber = Q(42)
myLength = Q(73*_m)
```

`Q.defineBase(symbol,name,dimension)`

This function is used to declare the base units. Units are represented as `Q` instances.

Parameters

`symbol` : `string`
: symbol of the base quantity

`name` : `string`
: name of the base quantity

`dimension` : `D`
: Instance of the `D` class, which represents the dimension of the quantity.

`return` : `Q`
: The created `Q` instance.

Note

The function creates a global variable, an underscore concatenated with the `symbol` argument, e. g. `m` becomes the global variable `_m`.

The `name` is used for example in the siunitx conversion function, e.g `meter` will be converted to `\meter`.

Each quantity has a dimension associated with it. The argument `dimension` allows any dimension to be associated to base quantities. By default, the SI convention is used.

Example

```
Q.defineBase("m", "meter", L)
Q.defineBase("kg", "kilogram", M)
```

`Quantity.define(symbol, name, q, tobase=nil, frombase=nil)`

Creates a new derived unit from an expression of other units. For affine quantities like the temperature in celcius, one can give convertion functions to and from base units.

Parameters

`symbol` : `string`
: Symbol of the base quantity

`name` : `string`
: Name of the base quantity

`q` : `physical.Quantity`
: Definition of the unit

`tobase` : `function`, optional
: to convert a quantity to base units

`frombase` : `function`, optional
: to convert a quantity from the base units

`return` : `Quantity`
The defined quantity

Examples

```
Q.define("L", "liter", _dm^3)
Q.define("Pa", "pascal", _N/_m^2)
Q.define("C", "coulomb", _A*_s)

Q.define(
  "degC",
  "celsius",
  _K,
  function(q)
    q.value = q.value + 273.15
    return q
  end,
  function(q)
    q.value = q.value - 273.15
    return q
  end
)
```

Quantity.definePrefix(symbol,name,factor)

Defines a new prefix.

`symbol` : `string`, Symbol of the base quantity

`name` : `string`, Name of the base quantity

`factor` : `number`, the factor which corresponds to the prefix

```
Q.definePrefix("c", "centi", 1e-2)
Q.definePrefix("a", "atto", 1e-18)
```

Quantity.addPrefix(prefixes, units)

Create several units with prefixes from a given unit.

`prefixes` : `string`, list of unit symbols

`units` : `Quantity`, list of quantities

```
Q.addPrefix({"n","u","m","k","M","G"},{_m,_s,_A})
```

Quantity.to(self,q,usefunction=false)

Converts the quantity self to the unit of the quantity `q`. If the boolean `usefunction` is true, the convertion function is used for conversion.

`self` : `Quantity`

`q` : `Quantity`

`usefunction` : `Bool`

```
s = 1.9 * _km
print( s:to(_m) )
1900 * _m

T = 10 * _degC
print( T:to(_K) )
10 * _K
print( T:to(_K,true) )
283.15 * _K
```

Quantity.tosiunitx(self,param,mode)

Converts the quantity into a siunitx string.

`self` : `Quantity`

`param` : `string`

`mode` : `Number`, 0:\SI, 1:\num, 2:\si

```
s = 1.9 * _km

print( s:tosiunitx() )
\SI{1.9}{\kilo\meter}

print( s:tosiunitx(nil,1) )
\num{1.9}

print( s:tosiunitx(nil,2) )
\si{\kilo\meter}
```

Quantity.isclose(self,q,r)

Checks if this quantity is close to another one. The argument `r` is the maximal relative deviation.

`self` : `Quantity`

`q` : `Quantity,Number`

`r` : `Number`

```
s_1 = 1.9 * _m
s_2 = 2.0 * _m
print( s_1:isclose(s_2,0.1) )
true
print( s_1:isclose(s_2,0.01) )
false
```

`Quantity.min(q1, q2, ...)`

Returns the smallest quantity of several given ones. The function returns `q1` if the Quantities are equal.

`q1` : `Quantity`,`Number`, first argument

`q2` : `Quantity`,`Number`, second argument

```
s_1 = 15 * _m
s_2 = 5 * _m
print(s_1:min(s_2))
5 * _m
```

`Quantity.max(q1, q2, ...)`

Returns the biggest quantity of several given ones. The function returns `q1` if the Quantities are equal.

`q1` : `Quantity`,`Number`, first argument

`q2` : `Quantity`,`Number`, second argument

```
s_1 = 15 * _m
s_2 = 5 * _m
print(s_1:max(s_2))
15 * _m
```

`Quantity.abs(q)`

Returns the absolute value of the given quantity `q`.

`q` : `Quantity`,`Number`, argument

```
U = -5 * _V
print(U)
-5 * _V
print(U:abs())
5 * _V
```

`Quantity.sqrt(q)`

Returns the square root of the given quantity.

`q` : `Quantity,Number` argument

```
A = 25 * _m^2
s = sqrt(A)
print(s)

```

`Quantity.log(q, base)`

Returns the logarithm of the given quantitiy. If no base is given, the natural logarithm is calculated.

`q` : `Quantity,Number` dimensionless argument

`base` : `Quantity,Number` dimensionless argument

```
I = 1 * _W/_m^2
I_0 = 1e-12 * _W/_m^2
print(10 * (I/I_0):log(10) * _dB )
120 * _dB
```

`Quantity.exp(q)`

Returns the value of the exponential function of the given quantitiy.

`q` : `Quantity,Number` dimensionless argument

```
x = 2 * _1
print( x:exp() )
7.3890560989307
```

`Quantity.sin(q)`

Returns the value of the sinus function of the given quantitiy.

`q` : `Quantity,Number` dimensionless argument

```
alpha = 30 * _deg
print( alpha:sin() )
0.5
```

`Quantity.cos(q)`

Returns the value of the cosinus function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity,Number` dimensionless argument

```
alpha = 60 * _deg
print( alpha:cos() )
0.5
```

`Quantity.tan(q)`

Returns the value of the tangent function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity,Number` dimensionless argument

```
alpha = 45 * _deg
print( alpha:tan() )
1
```

`Quantity.asin(q)`

Returns the value of the arcus sinus function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity,Number` dimensionless argument

```
x = 0.5 * _1
print( x:asin():to(_deg) )
30 * _deg
```

`Quantity.acos(q)`

Returns the value of the arcus cosinus function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity,Number` dimensionless argument

```
x = 0.5 * _1
print( x:acos():to(_deg) )
60 * _deg
```

`Quantity.atan(q)`

Returns the value of the arcus tangent function of the given quantity. The quantity has to be dimensionless.

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:atan():to(_deg) )
45 * _deg
```

`Quantity.sinh(q)`

Returns the value of the hyperbolic sine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\sinh(x) = 0.5 \cdot e^x - 0.5/e^x \quad .$$

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:sinh() )
1.1752011936438
```

`Quantity.cosh(q)`

Returns the value of the hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\cosh(x) = 0.5 \cdot e^x + 0.5/e^x \quad .$$

`q` : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:cosh() )
1.5430806348152
```

Quantity.tanh(q)

Returns the value of the hyperbolic tangent function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad .$$

q : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:tanh() )
0.76159415595576
```

Quantity.asinh(q)

Returns the value of the inverse hyperbolic sine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{asinh}(x) = \ln\left(x + \sqrt{x^2 + 1}\right) \quad .$$

q : `Quantity,Number` dimensionless argument

```
x = 1 * _1
print( x:asinh() )
0.88137358701954
```

Quantity.acosh(q)

Returns the value of the inverse hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\operatorname{acosh}(x) = \ln\left(x + \sqrt{x^2 - 1}\right) \quad , x > 1 \quad .$$

q : `Quantity,Number` dimensionless argument bigger than or equal to one.

```
x = 2 * _1
print( x:acosh() )
1.3169578969248
```

`Quantity.atanh(q)`

Returns the value of the inverse hyperbolic cosine function of the given quantity. The quantity has to be dimensionless. Since lua doesn't implement the hyperbolic functions the following formula is used

$$\mathrm{atanh}(x) = \ln\left(\frac{1+x}{1-x}\right) \quad , -1 < x < 1 \quad .$$

`q` : `Quantity,Number` dimensionless argument with magnitude smaller than one.

```
x = 0.5 * _1
print( x:atanh() )
0.54930614433405
```

5.2 physical.Dimension

All physical quantities do have a physical dimension. For example the quantity *Area* has the dimension L^2 (lenght to the power of two). In the SI-System there are seven base dimensions, from which all other dimensions are derived. Each dimension is represented by an n-tuple, where n is the number of base dimensions. Each physical quantity has an associated dimension object. It is used two check if two quantities can be added or subtraced and if they are equal.

`Dimension.new(q=nil)`

Constructor of the `Dimension` class.

Parameters

`q` : `Dimension` or `string`, optional
The name or symbol of the dimension. If `q` is a dimension, a copy of it is made. If no argument ist given, a dimension *zero* is created.

`return` : `Dimension`
The created `Quantity` object

Notes

–

Examples

```
V_1 = D("Velocity")
L = D("L")
V_2 = D(L/T)
```

5.3 physical.Unit

The task of this class is keeping track of the unit term. The unit term is a fraction of units. The units in the enumerator and denominator can have an exponent.

`Unit.new(u=nil)`

Copy Constructor. It copies a given unit object. If nothing is given, an empty unit is created.

Parameters

`u` : `Unit`
The unit object which will be copied.

`return` : `Unit`
The created `Unit` object

`Unit.new(symbol, name, prefixsymbol=nil, prefixname=nil)`

Constructor. A new `Unit` object with symbol is created. The prefixsymbol and prefixname are optional.

Parameters

`symbol` : `String`
The symbol of the unit.

`name` : `String`
The name of the unit.

`prefixsymbol` : `String`
The optional symbol of the prefix.

`prefixname` : `String`
The optional name of the prefix.

`return` : `Unit`
The created `Unit` object

`Unit.tosiunitx(self)`

The unit term will be compiled into a string, which the LaTeX package siunitx can understand.

Parameters

`return` : `String`
The siunitx representation of the unit term.