omdl

v0.6

Generated by Doxygen 1.8.9.1

Tue Apr 4 2017 20:15:43

Contents

1 omdl

It is an OpenSCAD mechanical design library (omdl) that provides open-source high-level design primitives with coherent documentation generated by Doxygen using openscad-amu.

With Doxygen, the code documentation is written within the code itself, and is thus easy to keep current. Moreover, it provides a standard way to both write and present OpenSCAD design documentation, compilable to common output formats (html, pdf, etc). With omdl, all library primitives are *parametric* with minimal, mostly zero, global variable dependencies and all library API's include markups that describe its parameters, behavior, and use.

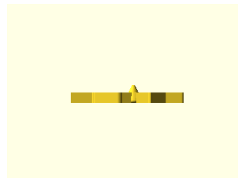
Validation scripts are used to verify that the core operations work as expected across evolving OpenSCAD versions (validation performed when building the documentation). The library uses a common set of conventions for specifying data types and is divided into individual component modules of functionality, organized into groups, that may be `included` as desired.

Example:

```
include <shapes/shapes2de.scad>;
include <shapes/shapes3d.scad>;

$fn = 36;

frame = triangle_lp2ls( [ [30,0], [0,40], [30,40] ] );
core  = 2 * frame / 3;
vrnd  = [1, 2, 4];

cone( h=20, r=10, vr=2 );
rotate([0, 0, 360/20])
radial_repeat( n=5, angle=true )
  etriangle_ls_c( vs=frame, vc=core, vr=vrnd, h=10 );
```

```
translate([0, -50,0])
linear_extrude(height=10)
text( text="omdl", size=20, halign="center", valign="center" );
```

Table 1: Example Result

Bottom	Diagonal	Right	Top
			

1.1 Using

To use omdl, the library files must be copied to an OpenSCAD library location. This can be done manually or can be done using using openscad-amu.

The ladder has several advantages and is recommended. When using openscad-amu, the library documentation is installed together with the library source code. This documentation is also added to a local browsable index, which facilitates reference use. Moreover, with openscad-amu installed, one can develop documentation for other OpenSCAD design scripts.

See the recommended installation method for more information. Library releases are periodically made available in the omdl repository under snapshots.

1.2 Contributing

omdl uses git for development tracking, and is hosted on GitHub following the usual practice of forking and submitting pull requests to the source repository.

As it is released under the GNU Lesser General Public License, any file you change should bear your copyright notice alongside the original authors' copyright notices typically located at the top of each file.

Ideas, requests, comments, contributions, and constructive criticism are welcome.

1.3 Support

In case you have any questions or would like to make feature requests, you can contact the maintainer of the project or file an issue.

2 Conventions

- Data types

2.1 Data types

2.1.1 Built-in

`omdl` assumes a `value` is either a number, a boolean, a string, a list, a range, or the undefined value. What is called a vector in the `OpenSCAD types` documentation is refereed to as a *list* here in order to distinguish between sequential lists of values and `Euclidean vectors`.

type	description
`value`	any valid OpenSCAD storable datum
number	an arithmetic value
boolean	a binary logic value (true or false)
string	a sequential list of of character values
list	a sequential list of arbitrary values
range	an arithmetic sequence
undef	the undefined value

2.1.1.1 Special numerical values

value	description
`nan`	a numerical value which is not a number
`inf`	a numerical value which is infinite

2.1.2 Additional conventions

When a list has an expected number of elements 'n', the *count* is appended following a '-'. When there is a range of expected elements, the lower and upper bounds are separated by a ':' and appended (order of bounds may be reversed). When the elements values are of an expected type, that *type* is prepended. Combinations are used as needed as in the following table:

name	description
list-n	a list of of n elements
list-l:u	a list of l to u elements
type-list	a list of elements with an expected type
type-list-n	a list of n elements with an expected type

2.1.2.1 Distinctions

`omdl` make the following distinctions on variable types.

name	description
`scalar`	a single non-iterable value
`iterable`	a multi-part sequence of values
`empty`	an iterable value with zero elements
`even`	an even numerical value
`odd`	an odd numerical value

2.1.2.2 General

From the fixed built-in set of `data types`, `omdl` adds the following general type specifications and conventions.

name	description
`bit`	a binary numerical value (0 or 1)
`integer`	a positive, negative, or zero whole number
`decimal`	integer numbers with a fractional part

index	a list index sequence
datastruct	a defined data structure
data	an arbitrary data structure

2.1.2.2.1 Index sequence

The data type **index** refers to a specified sequence of list element indexes. A list index sequence may be specified in one of the following forms.

value / form	description
true	All index positions of the list [0:size-1]
false	No index positions
"all"	All index positions of the list [0:size-1]
"none"	No index positions
"rands"	Random index selection of the list [0:size-1]
"even"	The even index of the list [0:size-1]
"odd"	The odd index of the list [0:size-1]
<integer>	The single position given by an <integer>
<range>	The range of positions given by a <range>
<integer-list>	The list of positions give in <integer-list>

The function get_index() can be used to convert a value of this data type into a sequence of list element indexes.

Example

```
// list
l1 = [a,b,c,d,e,f]

// index sequence
get_index(l1)          = [0,1,2,3,4,5]
get_index(l1, "rands") = [0,2,5]
```

2.1.2.3 Geometric

For geometric specifications and geometric algebra, omdl adds the following type specifications and conventions.

name	description
point	a list of numbers to identify a location in space
vector	a direction and magnitude in space
line	a start and end point in space (line wiki)
normal	a vector that is perpendicular to a given object
pnorm	a vector that is perpendicular to a plane
plane	a flat 2d infinite surface (plane wiki)
coords	a list of points in space
matrix	a rectangular array of values

When a particular dimension is expected, the dimensional expectation is appended to the end of the name after a '-' dash as in the following table.

name	description
point-Nd	a point in an 'N' dimensional space
vector-Nd	a vector in an 'N' dimensional space
line-Nd	a line in an 'N' dimensional space
coords-Nd	a coordinate list in an 'N' dimensional space
matrix-MxN	a 'M' by 'N' matrix of values

2.1.2.3.1 Lines and vectors

The data type **line** refers to a convention for specifying a line or a vector. A vector is a direction and magnitude in space. A line, too, has direction and magnitude, but also has location, as it starts at one point in space and ends at another. Operators in omdl make use of a common convention for specifying Euclidean vectors and straight lines as summarized in the following table:

Given two points 'p1' and 'p2', in space:

no.	form	description
1	p2	a line or vector from the origin to 'p2'
2	[p2]	a line or vector from the origin to 'p2'
3	[p1, p2]	line or vector from 'p1' to 'p2'

The functions get_line_dim(), get_line_tp(), get_line_ip(), and get_line2origin(), are available to identify the dimension of and convert a line into a vector or point.

Example

```
// points
p1 = [a,b,c]
p2 = [d,e,f]

// lines and vectors
v1 = p2       = [d,e,f]
v2 = [p2]     = [[d,e,f]]
v3 = [p1, p2] = [[a,b,c], [d,e,f]]

v1 == v2
v1 == v2 == v3, iff p1 == origin3d
```

2.1.2.3.2 Planes

Operators in omdl use a common convention for specifying planes. A **plane** is identified by a point on its surface together with its normal vector specified by pnorm, which is discussed in the following section. A list with a point and normal together specify the plane as follows:

name	form
plane	[point, pnorm]

2.1.2.3.3 Planes' normal

The data type **pnorm** refers to a convention for specifying a direction vector that is perpendicular to a plane. Given three points 'p1', 'p2', 'p3', and three vectors 'v1', 'v2', 'vn', the planes' normal can be specified in any of the following forms:

no.	form	description
1	vn	the predetermined normal vector to the plane
2	[vn]	the predetermined normal vector to the plane
3	[v1, v2]	two distinct but intersecting vectors
4	[p1, p2, p3]	three (or more) non-collinear coplanar points

The function get_pnorm2nv() can be used to convert a value of this data type into a normal vector.

Example

```
// points
p1 = [a,b,c];
p2 = [d,e,f];
p3 = [g,h,i];

// lines and vectors
v1 = [p1, p2] = [[a,b,c], [d,e,f]]
v2 = [p1, p3] = [[a,b,c], [g,h,i]]
vn = cross_ll(v1, v2)

// planes' normal
```

```
n1 = vn           = cross_ll(v1, v2)
n2 = [vn]         = cross_ll(v1, v2)
n3 = [v1, v2]     = [[[a,b,c],[d,e,f]], [[a,b,c],[g,h,i]]]
n4 = [p1, p2, p3] = [[a,b,c], [d,e,f], [g,h,i]]

n1 || n2 || n3 || n4

// planes
pn1 = [p1, n1]
pn2 = [p2, n2]
pn3 = [p3, n3]
pn4 = [n4[0], n4]
pn5 = [mean(n4), n4]

pn1 == pn4
```

3 Validation

- Datatypes
- Math

3.1 Datatypes

- Identification
- Operations

3.1.1 Identification

- Scalar
- Iterables
- Lists

3.1.1.1 Scalar

- Script
- Results

3.1.1.1.1 Script

```
include <datatypes.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
```

```
[
  ["t01", "The undefined value",        undef],
  ["t02", "An odd integer",             1],
  ["t03", "An small even integer",      10],
  ["t04", "A large integer",            100000000],
  ["t05", "A small decimal (epsilon)",  aeps],
  ["t06", "The max number",             number_max],
  ["t07", "The min number",             number_min],
  ["t08", "The max number^2",           number_max * number_max],
  ["t09", "The invalid number nan",     0 / 0],
  ["t10", "The boolean true",           true],
  ["t11", "The boolean false",          false],
  ["t12", "A character string",         "a"],
  ["t13", "A string",                   "This is a longer string"],
  ["t14", "The empty string",           empty_str],
  ["t15", "The empty list",             empty_lst],
  ["t16", "A 1-tuple list of undef",    [undef]],
  ["t17", "A 1-tuple list",             [10]],
  ["t18", "A 3-tuple list",             [1, 2, 3]],
  ["t19", "A list of lists",            [[1,2,3], [4,5,6], [7,8,9]]],
  ["t20", "A shorthand range",          [0:9]],
  ["t21", "A range",                    [0:0.5:9]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
t = true;   // shortcuts
f = false;
u = undef;
s = -1;     // skip test

good_r =
[ // function       01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
  ["is_defined",    f, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t],
  ["not_defined",   t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["is_nan",        f, f, f, f, f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f],
  ["is_inf",        f, f, f, f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["is_scalar",     t, t, t, t, t, t, t, t, t, t, t, f, f, f, f, f, f, f, f, s, s],
  ["is_iterable",   f, f, f, f, f, f, f, f, f, f, f, t, t, t, t, t, t, t, t, s, s],
  ["is_empty",      f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, f, f, f, f, f, f],
  ["is_number",     f, t, t, t, t, t, t, t, t, f, f, f, f, f, f, f, f, f, f, f, f],
  ["is_integer",    f, t, t, t, f, t, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["is_decimal",    f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["is_boolean",    f, f, f, f, f, f, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f],
  ["is_string",     f, f, f, f, f, f, f, f, f, f, f, t, t, t, f, f, f, f, f, f, f],
  ["is_list",       f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, t, t, t, s, s],
  ["is_range",      f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t],
  ["is_even",       s, f, t, t, f, t, t, s, s, s, s, s, s, s, s, s, s, s, s, s, s],
  ["is_odd",        s, t, f, f, f, f, f, s, s, s, s, s, s, s, s, s, s, s, s, s, s],
  ["is_between_MM", f, t, t, t, t, t, t, f, f, t, t, f, f, f, f, f, f, f, f, f, f]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
module run_test( fname, fresult, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");
  pass_value = get_table_v(good_r, good_c, fname, vid);

  test_pass = validate( cv=fresult, t=pass_value, pf=true );
  test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, pass_value
  );


  if ( pass_value != s )
  {
    if ( !test_pass )
      log_warn( str(vid, "(", value_text, ") ", test_text) );
    else if ( show_passing )
      log_info( str(vid, " ", test_text) );
  }
  else if ( show_skipped )
    log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
```

```
}

// Indirect function calls would be very useful here!!!
for (vid=test_ids) run_test( "is_defined", is_defined(get_value(vid)), vid );
for (vid=test_ids) run_test( "not_defined", not_defined(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_nan", is_nan(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_inf", is_inf(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_scalar", is_scalar(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_iterable", is_iterable(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_empty", is_empty(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_number", is_number(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_integer", is_integer(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_decimal", is_decimal(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_boolean", is_boolean(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_string", is_string(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_list", is_list(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_range", is_range(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_even", is_even(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_odd", is_odd(get_value(vid)), vid );
for (vid=test_ids) run_test( "is_between_MM", is_between(get_value(vid),
  number_min,number_max), vid );

// end-of-tests
```

3.1.1.1.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_defined(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_defined(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_defined(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_defined(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_defined(0.001)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_defined(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_defined(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_defined(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_defined(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_defined(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_defined(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_defined(a)=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_defined(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_defined()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_defined([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_defined([undef])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_defined([10])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_defined([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_defined([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_defined([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_defined([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'not_defined(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'not_defined(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'not_defined(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'not_defined(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'not_defined(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'not_defined(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'not_defined(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'not_defined(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'not_defined(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'not_defined(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'not_defined(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'not_defined(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'not_defined(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'not_defined()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'not_defined([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'not_defined([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'not_defined([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'not_defined([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'not_defined([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'not_defined([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'not_defined([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_nan(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_nan(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_nan(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_nan(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_nan(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_nan(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_nan(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_nan(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_nan(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_nan(true)=false'"
```

```
ECHO: "[ INFO ] run_test(); t11 passed: 'is_nan(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_nan(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_nan(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_nan()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_nan([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_nan([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_nan([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_nan([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_nan([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_nan([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_nan([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_inf(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_inf(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_inf(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_inf(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_inf(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_inf(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_inf(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_inf(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_inf(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_inf(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_inf(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_inf(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_inf(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_inf()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_inf([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_inf([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_inf([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_inf([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_inf([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_inf([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_inf([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_scalar(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_scalar(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_scalar(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_scalar(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_scalar(0.001)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_scalar(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_scalar(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_scalar(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_scalar(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_scalar(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_scalar(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_scalar(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_scalar(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_scalar()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_scalar([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_scalar([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_scalar([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_scalar([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_scalar([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_scalar(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_scalar(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_iterable(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_iterable(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_iterable(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_iterable(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_iterable(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_iterable(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_iterable(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_iterable(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_iterable(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_iterable(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_iterable(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_iterable(a)=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_iterable(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_iterable()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_iterable([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_iterable([undef])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_iterable([10])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_iterable([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_iterable([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=true'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_iterable(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_iterable(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_empty(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_empty(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_empty(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_empty(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_empty(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_empty(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_empty(-1e+308)=false'"
```

```
ECHO: "[ INFO ] run_test(); t08 passed: 'is_empty(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_empty(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_empty(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_empty(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_empty(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_empty(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_empty()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_empty([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_empty([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_empty([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_empty([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_empty([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_empty([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_empty([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_number(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_number(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_number(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_number(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_number(0.001)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_number(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_number(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_number(inf)=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_number(nan)=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_number(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_number(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_number(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_number(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_number()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_number([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_number([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_number([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_number([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_number([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_number([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_number([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_integer(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_integer(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_integer(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_integer(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_integer(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_integer(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_integer(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_integer(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_integer(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_integer(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_integer(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_integer(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_integer(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_integer()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_integer([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_integer([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_integer([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_integer([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_integer([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_integer([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_integer([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_decimal(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_decimal(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_decimal(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_decimal(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_decimal(0.001)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_decimal(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_decimal(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_decimal(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_decimal(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_decimal(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_decimal(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_decimal(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_decimal(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_decimal()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_decimal([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_decimal([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_decimal([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_decimal([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_decimal([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_decimal([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_decimal([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_boolean(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_boolean(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_boolean(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_boolean(1e+08)=false'"
```

```
ECHO: "[ INFO ] run_test(); t05 passed: 'is_boolean(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_boolean(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_boolean(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_boolean(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_boolean(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_boolean(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_boolean(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_boolean(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_boolean(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_boolean()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_boolean([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_boolean([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_boolean([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_boolean([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_boolean([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_boolean([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_boolean([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_string(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_string(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_string(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_string(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_string(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_string(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_string(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_string(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_string(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_string(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_string(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_string(a)=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_string(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_string()=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_string([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_string([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_string([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_string([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_string([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_string([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_string([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_list(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_list(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_list(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_list(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_list(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_list(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_list(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_list(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_list(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_list(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_list(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_list(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_list(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_list()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_list([])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_list([undef])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_list([10])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_list([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_list([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=true'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_list(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_list(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_range(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_range(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_range(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_range(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_range(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_range(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_range(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_range(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_range(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_range(true)=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_range(false)=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_range(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_range(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_range()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_range([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_range([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_range([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_range([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_range([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_range([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_range([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t01 *skip*: 'is_even(The undefined value)'"
```

```
ECHO: "[ INFO ] run_test(); t02 passed: 'is_even(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_even(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_even(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_even(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_even(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_even(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'is_even(The max number^2)'"
ECHO: "[ INFO ] run_test(); t09 *skip*: 'is_even(The invalid number nan)'"
ECHO: "[ INFO ] run_test(); t10 *skip*: 'is_even(The boolean true)'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'is_even(The boolean false)'"
ECHO: "[ INFO ] run_test(); t12 *skip*: 'is_even(A character string)'"
ECHO: "[ INFO ] run_test(); t13 *skip*: 'is_even(A string)'"
ECHO: "[ INFO ] run_test(); t14 *skip*: 'is_even(The empty string)'"
ECHO: "[ INFO ] run_test(); t15 *skip*: 'is_even(The empty list)'"
ECHO: "[ INFO ] run_test(); t16 *skip*: 'is_even(A 1-tuple list of undef)'"
ECHO: "[ INFO ] run_test(); t17 *skip*: 'is_even(A 1-tuple list)'"
ECHO: "[ INFO ] run_test(); t18 *skip*: 'is_even(A 3-tuple list)'"
ECHO: "[ INFO ] run_test(); t19 *skip*: 'is_even(A list of lists)'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_even(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_even(A range)'"
ECHO: "[ INFO ] run_test(); t01 *skip*: 'is_odd(The undefined value)'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_odd(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_odd(10)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_odd(1e+08)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_odd(0.001)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_odd(1e+308)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_odd(-1e+308)=false'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'is_odd(The max number^2)'"
ECHO: "[ INFO ] run_test(); t09 *skip*: 'is_odd(The invalid number nan)'"
ECHO: "[ INFO ] run_test(); t10 *skip*: 'is_odd(The boolean true)'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'is_odd(The boolean false)'"
ECHO: "[ INFO ] run_test(); t12 *skip*: 'is_odd(A character string)'"
ECHO: "[ INFO ] run_test(); t13 *skip*: 'is_odd(A string)'"
ECHO: "[ INFO ] run_test(); t14 *skip*: 'is_odd(The empty string)'"
ECHO: "[ INFO ] run_test(); t15 *skip*: 'is_odd(The empty list)'"
ECHO: "[ INFO ] run_test(); t16 *skip*: 'is_odd(A 1-tuple list of undef)'"
ECHO: "[ INFO ] run_test(); t17 *skip*: 'is_odd(A 1-tuple list)'"
ECHO: "[ INFO ] run_test(); t18 *skip*: 'is_odd(A 3-tuple list)'"
ECHO: "[ INFO ] run_test(); t19 *skip*: 'is_odd(A list of lists)'"
ECHO: "[ INFO ] run_test(); t20 *skip*: 'is_odd(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t21 *skip*: 'is_odd(A range)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'is_between_MM(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'is_between_MM(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'is_between_MM(10)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'is_between_MM(1e+08)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'is_between_MM(0.001)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'is_between_MM(1e+308)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'is_between_MM(-1e+308)=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'is_between_MM(inf)=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'is_between_MM(nan)=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'is_between_MM(true)=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'is_between_MM(false)=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'is_between_MM(a)=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'is_between_MM(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'is_between_MM()=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'is_between_MM([])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'is_between_MM([undef])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'is_between_MM([10])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'is_between_MM([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'is_between_MM([[1, 2, 3], [4, 5, 6], [7, 8, 9]])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'is_between_MM([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'is_between_MM([0 : 0.5 : 9])=false'"
```

3.1.1.2 Iterables

- Script
- Results

3.1.1.2.1 Script

```
include <datatypes.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
```

```
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "An odd integer",             1],
  ["t03", "The boolean true",           true],
  ["t04", "The boolean false",          false],
  ["t05", "A character string",         "a"],
  ["t06", "A string",                   "This is a longer string"],
  ["t07", "The empty string",           empty_str],
  ["t08", "The empty list",             empty_lst],
  ["t09", "A shorthand range",          [0:9]],
  ["t10", "A range",                    [0:0.5:9]],
  ["t11", "Test list 01",               [undef]],
  ["t12", "Test list 02",               [1]],
  ["t13", "Test list 03",               [1, 2, 3]],
  ["t14", "Test list 04",               [[1], [2], [3], [4], [5]]],
  ["t15", "Test list 05",               [[1,2], [2,3]]],
  ["t16", "Test list 06",               [[1,2], [2,3], [4,5], "ab"]],
  ["t17", "Test list 07",               [[1,2,3], [4,5,6], [7,8,9], ["a", "b", "c"]]],
  ["t18", "Test list 08",               [1, 2, 3, undef]],
  ["t19", "Test list 09",               [undef, undef, undef, undef]],
  ["t20", "Test list 10",               [[undef], [undef], [undef]]],
  ["t21", "Test list 11",               [true, true, true, true, false]],
  ["t22", "Test list 12",               [true, false, false, false, false]],
  ["t23", "Test list 13",               [true, true, true, true]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
t = true;   // shortcuts
f = false;
u = undef;
s = -1;     // skip test

good_r =
[ // function           01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
  ["all_equal_T",       f, f, t, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t],
  ["all_equal_F",       f, f, f, t, f, f, t, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["all_equal_U",       t, f, f, f, f, f, t, t, f, f, t, f, f, f, f, f, f, f, t, f, f, f, f],
  ["any_equal_T",       f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, t],
  ["any_equal_F",       f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, t, t, f],
  ["any_equal_U",       t, f, f, f, f, f, f, f, f, f, t, f, f, f, f, f, f, t, t, f, f, f, f],
  ["all_defined",       f, t, t, t, t, t, t, t, t, t, f, t, t, t, t, t, t, f, f, t, t, t, t],
  ["any_undefined",     t, f, f, f, f, f, f, f, f, f, t, f, f, f, f, f, f, t, t, f, f, f, f],
  ["all_scalars",       u, t, t, t, f, f, s, s, s, s, t, t, t, f, f, f, f, t, t, f, t, t, t],
  ["all_lists",       u, f, f, f, f, f, t, t, f, f, f, f, f, t, t, f, t, f, f, t, f, f, f],
  ["all_strings",       u, f, f, f, t, t, t, s, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["all_numbers",       u, t, f, f, f, f, s, s, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f],
  ["all_len_1",         u, f, f, f, t, t, s, s, f, f, f, f, f, t, f, f, f, f, f, t, f, f, f],
  ["all_len_2",         u, f, f, f, f, f, s, s, f, f, f, f, f, f, t, t, f, f, f, f, f, f, f],
  ["all_len_3",         u, f, f, f, f, f, s, s, f, f, f, f, f, f, f, f, t, f, f, f, f, f, f]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
module run_test( fname, fresult, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");
  pass_value = get_table_v(good_r, good_c, fname, vid);
```

```
      test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
      test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals",
      pass_value );

      if ( pass_value != s )
      {
        if ( !test_pass )
          log_warn( str(vid, "(", value_text, ") ", test_text) );
        else if ( show_passing )
          log_info( str(vid, " ", test_text) );
      }
      else if ( show_skipped )
        log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
    }

    // Indirect function calls would be very useful here!!!
    for (vid=test_ids) run_test( "all_equal_T", all_equal(get_value(vid),t), vid );
    for (vid=test_ids) run_test( "all_equal_F", all_equal(get_value(vid),f), vid );
    for (vid=test_ids) run_test( "all_equal_U", all_equal(get_value(vid),u), vid );
    for (vid=test_ids) run_test( "any_equal_T", any_equal(get_value(vid),t), vid );
    for (vid=test_ids) run_test( "any_equal_F", any_equal(get_value(vid),f), vid );
    for (vid=test_ids) run_test( "any_equal_U", any_equal(get_value(vid),u), vid );
    for (vid=test_ids) run_test( "all_defined", all_defined(get_value(vid)), vid );
    for (vid=test_ids) run_test( "any_undefined", any_undefined(get_value(vid)), vid );
    for (vid=test_ids) run_test( "all_scalars", all_scalars(get_value(vid)), vid );
    for (vid=test_ids) run_test( "all_lists", all_lists(get_value(vid)), vid );
    for (vid=test_ids) run_test( "all_strings", all_strings(get_value(vid)), vid );
    for (vid=test_ids) run_test( "all_numbers", all_numbers(get_value(vid)), vid );
    for (vid=test_ids) run_test( "all_len_1", all_len(get_value(vid),1), vid );
    for (vid=test_ids) run_test( "all_len_2", all_len(get_value(vid),2), vid );
    for (vid=test_ids) run_test( "all_len_3", all_len(get_value(vid),3), vid );

    // end-of-tests
```

3.1.1.2.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_equal_T(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_equal_T(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_equal_T(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_equal_T(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_equal_T(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_equal_T(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'all_equal_T()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'all_equal_T([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_equal_T([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_equal_T([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_equal_T([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_equal_T([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_equal_T([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_equal_T([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_equal_T([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_equal_T([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_equal_T([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_equal_T([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_equal_T([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_equal_T([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_equal_T([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_equal_T([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_equal_T([true, true, true, true])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_equal_F(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_equal_F(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_equal_F(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_equal_F(false)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_equal_F(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_equal_F(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'all_equal_F()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'all_equal_F([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_equal_F([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_equal_F([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_equal_F([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_equal_F([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_equal_F([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_equal_F([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_equal_F([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_equal_F([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_equal_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false'"
```

```
ECHO: "[ INFO ] run_test(); t18 passed: 'all_equal_F([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_equal_F([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_equal_F([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_equal_F([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_equal_F([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_equal_F([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_equal_U(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_equal_U(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_equal_U(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_equal_U(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_equal_U(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_equal_U(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'all_equal_U()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'all_equal_U([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_equal_U([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_equal_U([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_equal_U([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_equal_U([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_equal_U([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_equal_U([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_equal_U([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_equal_U([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_equal_U([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_equal_U([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_equal_U([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_equal_U([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_equal_U([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_equal_U([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_equal_U([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'any_equal_T(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'any_equal_T(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'any_equal_T(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'any_equal_T(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'any_equal_T(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'any_equal_T(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'any_equal_T()=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'any_equal_T([])=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'any_equal_T([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'any_equal_T([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'any_equal_T([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'any_equal_T([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'any_equal_T([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'any_equal_T([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'any_equal_T([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'any_equal_T([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'any_equal_T([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'any_equal_T([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'any_equal_T([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'any_equal_T([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'any_equal_T([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'any_equal_T([true, false, false, false, false])=true'"
ECHO: "[ INFO ] run_test(); t23 passed: 'any_equal_T([true, true, true, true])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'any_equal_F(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'any_equal_F(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'any_equal_F(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'any_equal_F(false)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'any_equal_F(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'any_equal_F(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'any_equal_F()=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'any_equal_F([])=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'any_equal_F([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'any_equal_F([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'any_equal_F([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'any_equal_F([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'any_equal_F([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'any_equal_F([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'any_equal_F([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'any_equal_F([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'any_equal_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'any_equal_F([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'any_equal_F([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'any_equal_F([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'any_equal_F([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'any_equal_F([true, false, false, false, false])=true'"
ECHO: "[ INFO ] run_test(); t23 passed: 'any_equal_F([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'any_equal_U(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'any_equal_U(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'any_equal_U(true)=false'"
```

```
ECHO: "[ INFO ] run_test(); t04 passed: 'any_equal_U(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'any_equal_U(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'any_equal_U(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'any_equal_U()=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'any_equal_U([])=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'any_equal_U([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'any_equal_U([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'any_equal_U([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'any_equal_U([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'any_equal_U([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'any_equal_U([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'any_equal_U([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'any_equal_U([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'any_equal_U([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'any_equal_U([1, 2, 3, undef])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'any_equal_U([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'any_equal_U([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'any_equal_U([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'any_equal_U([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'any_equal_U([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_defined(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_defined(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_defined(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_defined(false)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_defined(a)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_defined(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'all_defined()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'all_defined([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_defined([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_defined([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_defined([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_defined([1])=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_defined([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_defined([[1], [2], [3], [4], [5]])=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_defined([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_defined([[1, 2], [2, 3], [4, 5], "ab"])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_defined([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_defined([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_defined([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_defined([[undef], [undef], [undef]])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_defined([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_defined([true, false, false, false, false])=true'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_defined([true, true, true, true])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'any_undefined(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'any_undefined(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'any_undefined(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'any_undefined(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'any_undefined(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'any_undefined(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'any_undefined()=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'any_undefined([])=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'any_undefined([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'any_undefined([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'any_undefined([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'any_undefined([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'any_undefined([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'any_undefined([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'any_undefined([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'any_undefined([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'any_undefined([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'any_undefined([1, 2, 3, undef])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'any_undefined([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'any_undefined([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'any_undefined([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'any_undefined([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'any_undefined([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_scalars(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_scalars(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_scalars(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_scalars(false)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_scalars(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_scalars(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'all_scalars(The empty string)'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_scalars(The empty list)'"
ECHO: "[ INFO ] run_test(); t09 *skip*: 'all_scalars(A shorthand range)'"
ECHO: "[ INFO ] run_test(); t10 *skip*: 'all_scalars(A range)'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_scalars([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_scalars([1])=true'"
```

```
ECHO: "[ INFO ] run_test(); t13 passed: 'all_scalars([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_scalars([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_scalars([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_scalars([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_scalars([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_scalars([1, 2, 3, undef])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_scalars([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_scalars([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_scalars([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_scalars([true, false, false, false, false])=true'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_scalars([true, true, true, true])=true'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_lists(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_lists(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_lists(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_lists(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_lists(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_lists(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'all_lists()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'all_lists([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_lists([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_lists([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_lists([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_lists([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_lists([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_lists([[1], [2], [3], [4], [5]])=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_lists([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_lists([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_lists([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_lists([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_lists([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_lists([[undef], [undef], [undef]])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_lists([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_lists([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_lists([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_strings(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_strings(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_strings(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_strings(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_strings(a)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_strings(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'all_strings()=true'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_strings(The empty list)'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_strings([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_strings([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_strings([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_strings([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_strings([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_strings([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_strings([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_strings([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_strings([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_strings([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_strings([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_strings([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_strings([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_strings([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_strings([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_numbers(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_numbers(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_numbers(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_numbers(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_numbers(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_numbers(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'all_numbers(The empty string)'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_numbers(The empty list)'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_numbers([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_numbers([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_numbers([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_numbers([1])=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_numbers([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_numbers([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_numbers([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_numbers([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_numbers([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_numbers([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_numbers([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_numbers([[undef], [undef], [undef]])=false'"
```

```
ECHO: "[ INFO ] run_test(); t21 passed: 'all_numbers([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_numbers([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_numbers([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_len_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_len_1(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_len_1(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_len_1(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_len_1(a)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_len_1(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'all_len_1(The empty string)'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_len_1(The empty list)'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_len_1([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_len_1([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_len_1([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_len_1([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_len_1([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_len_1([[1], [2], [3], [4], [5]])=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_len_1([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_len_1([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_len_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_len_1([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_len_1([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_len_1([[undef], [undef], [undef]])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_len_1([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_len_1([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_len_1([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_len_2(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_len_2(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_len_2(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_len_2(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_len_2(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_len_2(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'all_len_2(The empty string)'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_len_2(The empty list)'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_len_2([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_len_2([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_len_2([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_len_2([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_len_2([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_len_2([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_len_2([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_len_2([[1, 2], [2, 3], [4, 5], "ab"])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_len_2([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_len_2([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_len_2([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_len_2([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_len_2([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_len_2([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_len_2([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'all_len_3(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'all_len_3(1)=false'"
ECHO: "[ INFO ] run_test(); t03 passed: 'all_len_3(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'all_len_3(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'all_len_3(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'all_len_3(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'all_len_3(The empty string)'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'all_len_3(The empty list)'"
ECHO: "[ INFO ] run_test(); t09 passed: 'all_len_3([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'all_len_3([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'all_len_3([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'all_len_3([1])=false'"
ECHO: "[ INFO ] run_test(); t13 passed: 'all_len_3([1, 2, 3])=false'"
ECHO: "[ INFO ] run_test(); t14 passed: 'all_len_3([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'all_len_3([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'all_len_3([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'all_len_3([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'all_len_3([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'all_len_3([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'all_len_3([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'all_len_3([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'all_len_3([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'all_len_3([true, true, true, true])=false'"
```

3.1.1.3 Lists

- Script

- Results

3.1.1.3.1 Script

```
include <datatypes.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "An odd integer",             1],
  ["t03", "The boolean true",           true],
  ["t04", "The boolean false",          false],
  ["t05", "A character string",         "a"],
  ["t06", "A string",                   "This is a longer string"],
  ["t07", "The empty string",           empty_str],
  ["t08", "The empty list",             empty_lst],
  ["t09", "A shorthand range",          [0:9]],
  ["t10", "A range",                    [0:0.5:9]],
  ["t11", "Test list 01",               [undef]],
  ["t12", "Test list 02",               [1]],
  ["t13", "Test list 03",               [1, 2, 3]],
  ["t14", "Test list 04",               [[1], [2], [3], [4], [5]]],
  ["t15", "Test list 05",               [[1,2], [2,3]]],
  ["t16", "Test list 06",               [[1,2], [2,3], [4,5], "ab"]],
  ["t17", "Test list 07",               [[1,2,3], [4,5,6], [7,8,9], ["a", "b", "c"]]],
  ["t18", "Test list 08",               [1, 2, 3, undef]],
  ["t19", "Test list 09",               [undef, undef, undef, undef]],
  ["t20", "Test list 10",               [[undef], [undef], [undef]]],
  ["t21", "Test list 11",               [true, true, true, true, false]],
  ["t22", "Test list 12",               [true, false, false, false, false]],
  ["t23", "Test list 13",               [true, true, true, true]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
t = true;   // shortcuts
f = false;
u = undef;
s = -1;     // skip test

good_r =
[ // function           01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
  ["n_almost_equal_AA", f, t, f, f, f, f, f, f, f, f, f, t, t, f, f, f, f, f, f, f, f, f, f],
  ["almost_equal_AA",   t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t],
  ["almost_equal_T",    f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["almost_equal_F",    f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["almost_equal_U",    t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f],
  ["compare_AA",        t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t, t]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
module run_test( fname, fresult, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");
```

```
      pass_value = get_table_v(good_r, good_c, fname, vid);

      test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
      test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals",
      pass_value );

      if ( pass_value != s )
      {
        if ( !test_pass )
          log_warn( str(vid, "(", value_text, ") ", test_text) );
        else if ( show_passing )
          log_info( str(vid, " ", test_text) );
      }
      else if ( show_skipped )
        log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
    }

    // Indirect function calls would be very useful here!!!
    for (vid=test_ids) run_test( "n_almost_equal_AA", n_almost_equal(get_value(vid),get_value
      (vid)), vid );
    for (vid=test_ids) run_test( "almost_equal_AA", almost_equal(get_value(vid),get_value(vid))
      , vid );
    for (vid=test_ids) run_test( "almost_equal_T", almost_equal(get_value(vid),t), vid );
    for (vid=test_ids) run_test( "almost_equal_F", almost_equal(get_value(vid),f), vid );
    for (vid=test_ids) run_test( "almost_equal_U", almost_equal(get_value(vid),u), vid );
    for (vid=test_ids) run_test( "compare_AA", compare(get_value(vid),get_value(vid)) == 0, vid );

    // end-of-tests
```

3.1.1.3.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run_test(); t01 passed: 'n_almost_equal_AA(undef)=false'"
ECHO: "[ INFO ] run_test(); t02 passed: 'n_almost_equal_AA(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'n_almost_equal_AA(true)=false'"
ECHO: "[ INFO ] run_test(); t04 passed: 'n_almost_equal_AA(false)=false'"
ECHO: "[ INFO ] run_test(); t05 passed: 'n_almost_equal_AA(a)=false'"
ECHO: "[ INFO ] run_test(); t06 passed: 'n_almost_equal_AA(This is a longer string)=false'"
ECHO: "[ INFO ] run_test(); t07 passed: 'n_almost_equal_AA()=false'"
ECHO: "[ INFO ] run_test(); t08 passed: 'n_almost_equal_AA([])=false'"
ECHO: "[ INFO ] run_test(); t09 passed: 'n_almost_equal_AA([0 : 1 : 9])=false'"
ECHO: "[ INFO ] run_test(); t10 passed: 'n_almost_equal_AA([0 : 0.5 : 9])=false'"
ECHO: "[ INFO ] run_test(); t11 passed: 'n_almost_equal_AA([undef])=false'"
ECHO: "[ INFO ] run_test(); t12 passed: 'n_almost_equal_AA([1])=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'n_almost_equal_AA([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'n_almost_equal_AA([[1], [2], [3], [4], [5]])=false'"
ECHO: "[ INFO ] run_test(); t15 passed: 'n_almost_equal_AA([[1, 2], [2, 3]])=false'"
ECHO: "[ INFO ] run_test(); t16 passed: 'n_almost_equal_AA([[1, 2], [2, 3], [4, 5], "ab"])=false'"
ECHO: "[ INFO ] run_test(); t17 passed: 'n_almost_equal_AA([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=false'"
ECHO: "[ INFO ] run_test(); t18 passed: 'n_almost_equal_AA([1, 2, 3, undef])=false'"
ECHO: "[ INFO ] run_test(); t19 passed: 'n_almost_equal_AA([undef, undef, undef, undef])=false'"
ECHO: "[ INFO ] run_test(); t20 passed: 'n_almost_equal_AA([[undef], [undef], [undef]])=false'"
ECHO: "[ INFO ] run_test(); t21 passed: 'n_almost_equal_AA([true, true, true, true, false])=false'"
ECHO: "[ INFO ] run_test(); t22 passed: 'n_almost_equal_AA([true, false, false, false, false])=false'"
ECHO: "[ INFO ] run_test(); t23 passed: 'n_almost_equal_AA([true, true, true, true])=false'"
ECHO: "[ INFO ] run_test(); t01 passed: 'almost_equal_AA(undef)=true'"
ECHO: "[ INFO ] run_test(); t02 passed: 'almost_equal_AA(1)=true'"
ECHO: "[ INFO ] run_test(); t03 passed: 'almost_equal_AA(true)=true'"
ECHO: "[ INFO ] run_test(); t04 passed: 'almost_equal_AA(false)=true'"
ECHO: "[ INFO ] run_test(); t05 passed: 'almost_equal_AA(a)=true'"
ECHO: "[ INFO ] run_test(); t06 passed: 'almost_equal_AA(This is a longer string)=true'"
ECHO: "[ INFO ] run_test(); t07 passed: 'almost_equal_AA()=true'"
ECHO: "[ INFO ] run_test(); t08 passed: 'almost_equal_AA([])=true'"
ECHO: "[ INFO ] run_test(); t09 passed: 'almost_equal_AA([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'almost_equal_AA([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'almost_equal_AA([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'almost_equal_AA([1])=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'almost_equal_AA([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'almost_equal_AA([[1], [2], [3], [4], [5]])=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'almost_equal_AA([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'almost_equal_AA([[1, 2], [2, 3], [4, 5], "ab"])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'almost_equal_AA([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'almost_equal_AA([1, 2, 3, undef])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'almost_equal_AA([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'almost_equal_AA([[undef], [undef], [undef]])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'almost_equal_AA([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'almost_equal_AA([true, false, false, false, false])=true'"
```

```
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_AA([true, true, true, true])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’almost_equal_T(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’almost_equal_T(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’almost_equal_T(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’almost_equal_T(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’almost_equal_T(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’almost_equal_T(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’almost_equal_T()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’almost_equal_T([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’almost_equal_T([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’almost_equal_T([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’almost_equal_T([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’almost_equal_T([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’almost_equal_T([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’almost_equal_T([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’almost_equal_T([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’almost_equal_T([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’almost_equal_T([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
      "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’almost_equal_T([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’almost_equal_T([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’almost_equal_T([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’almost_equal_T([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’almost_equal_T([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_T([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’almost_equal_F(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’almost_equal_F(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’almost_equal_F(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’almost_equal_F(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’almost_equal_F(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’almost_equal_F(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’almost_equal_F()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’almost_equal_F([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’almost_equal_F([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’almost_equal_F([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’almost_equal_F([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’almost_equal_F([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’almost_equal_F([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’almost_equal_F([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’almost_equal_F([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’almost_equal_F([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’almost_equal_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
      "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’almost_equal_F([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’almost_equal_F([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’almost_equal_F([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’almost_equal_F([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’almost_equal_F([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_F([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’almost_equal_U(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’almost_equal_U(1)=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’almost_equal_U(true)=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’almost_equal_U(false)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’almost_equal_U(a)=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’almost_equal_U(This is a longer string)=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’almost_equal_U()=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’almost_equal_U([])=false’"
ECHO: "[ INFO ] run_test(); t09 passed: ’almost_equal_U([0 : 1 : 9])=false’"
ECHO: "[ INFO ] run_test(); t10 passed: ’almost_equal_U([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t11 passed: ’almost_equal_U([undef])=false’"
ECHO: "[ INFO ] run_test(); t12 passed: ’almost_equal_U([1])=false’"
ECHO: "[ INFO ] run_test(); t13 passed: ’almost_equal_U([1, 2, 3])=false’"
ECHO: "[ INFO ] run_test(); t14 passed: ’almost_equal_U([[1], [2], [3], [4], [5]])=false’"
ECHO: "[ INFO ] run_test(); t15 passed: ’almost_equal_U([[1, 2], [2, 3]])=false’"
ECHO: "[ INFO ] run_test(); t16 passed: ’almost_equal_U([[1, 2], [2, 3], [4, 5], "ab"])=false’"
ECHO: "[ INFO ] run_test(); t17 passed: ’almost_equal_U([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
      "c"]])=false’"
ECHO: "[ INFO ] run_test(); t18 passed: ’almost_equal_U([1, 2, 3, undef])=false’"
ECHO: "[ INFO ] run_test(); t19 passed: ’almost_equal_U([undef, undef, undef, undef])=false’"
ECHO: "[ INFO ] run_test(); t20 passed: ’almost_equal_U([[undef], [undef], [undef]])=false’"
ECHO: "[ INFO ] run_test(); t21 passed: ’almost_equal_U([true, true, true, true, false])=false’"
ECHO: "[ INFO ] run_test(); t22 passed: ’almost_equal_U([true, false, false, false, false])=false’"
ECHO: "[ INFO ] run_test(); t23 passed: ’almost_equal_U([true, true, true, true])=false’"
ECHO: "[ INFO ] run_test(); t01 passed: ’compare_AA(undef)=true’"
ECHO: "[ INFO ] run_test(); t02 passed: ’compare_AA(1)=true’"
ECHO: "[ INFO ] run_test(); t03 passed: ’compare_AA(true)=true’"
ECHO: "[ INFO ] run_test(); t04 passed: ’compare_AA(false)=true’"
ECHO: "[ INFO ] run_test(); t05 passed: ’compare_AA(a)=true’"
ECHO: "[ INFO ] run_test(); t06 passed: ’compare_AA(This is a longer string)=true’"
ECHO: "[ INFO ] run_test(); t07 passed: ’compare_AA()=true’"
ECHO: "[ INFO ] run_test(); t08 passed: ’compare_AA([])=true’"
```

```
ECHO: "[ INFO ] run_test(); t09 passed: 'compare_AA([0 : 1 : 9])=true'"
ECHO: "[ INFO ] run_test(); t10 passed: 'compare_AA([0 : 0.5 : 9])=true'"
ECHO: "[ INFO ] run_test(); t11 passed: 'compare_AA([undef])=true'"
ECHO: "[ INFO ] run_test(); t12 passed: 'compare_AA([1])=true'"
ECHO: "[ INFO ] run_test(); t13 passed: 'compare_AA([1, 2, 3])=true'"
ECHO: "[ INFO ] run_test(); t14 passed: 'compare_AA([[1], [2], [3], [4], [5]])=true'"
ECHO: "[ INFO ] run_test(); t15 passed: 'compare_AA([[1, 2], [2, 3]])=true'"
ECHO: "[ INFO ] run_test(); t16 passed: 'compare_AA([[1, 2], [2, 3], [4, 5], "ab"])=true'"
ECHO: "[ INFO ] run_test(); t17 passed: 'compare_AA([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=true'"
ECHO: "[ INFO ] run_test(); t18 passed: 'compare_AA([1, 2, 3, undef])=true'"
ECHO: "[ INFO ] run_test(); t19 passed: 'compare_AA([undef, undef, undef, undef])=true'"
ECHO: "[ INFO ] run_test(); t20 passed: 'compare_AA([[undef], [undef], [undef]])=true'"
ECHO: "[ INFO ] run_test(); t21 passed: 'compare_AA([true, true, true, true, false])=true'"
ECHO: "[ INFO ] run_test(); t22 passed: 'compare_AA([true, false, false, false, false])=true'"
ECHO: "[ INFO ] run_test(); t23 passed: 'compare_AA([true, true, true, true])=true'"
```

3.1.2 Operations

- Scalar
- Iterables
- Lists

3.1.2.1 Scalar

- Script
- Results

3.1.2.1.1 Script

```
include <datatypes.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "The empty list",             empty_lst],
  ["t03", "A range",                    [0:0.5:9]],
  ["t04", "A string",                   "A string"],
  ["t05", "Test list 01",               ["orange","apple","grape","banana"]],
  ["t06", "Test list 02",               ["b","a","n","a","n","a","s"]],
  ["t07", "Test list 03",               [undef]],
  ["t08", "Test list 04",               [[1,2],[2,3]]],
  ["t09", "Test list 05",               ["ab",[1,2],[2,3],[4,5]]],
  ["t10", "Test list 06",               [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]]],
  ["t11", "Vector of integers 0 to 15", [for (i=[0:15]) i]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
```

```
skip = -1;  // skip test

good_r =
[ // function
  ["defined_or_D",
    "default",                                         // t01
    empty_lst,                                         // t02
    [0:0.5:9],                                         // t03
    "A string",                                        // 04
    ["orange","apple","grape","banana"],               // t05
    ["b","a","n","a","n","a","s"],                     // t06
    [undef],                                           // t07
    [[1,2],[2,3]],                                     // t08
    ["ab",[1,2],[2,3],[4,5]],                          // t09
    [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],           // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]            // t11
  ]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
module run_test( fname, fresult, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");
  pass_value = get_table_v(good_r, good_c, fname, vid);

  test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
  test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals",
  pass_value );

  if ( pass_value != skip )
  {
    if ( !test_pass )
      log_warn( str(vid, "(", value_text, ") ", test_text) );
    else if ( show_passing )
      log_info( str(vid, " ", test_text) );
  }
  else if ( show_skipped )
    log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
}

// Indirect function calls would be very useful here!!!
for (vid=test_ids) run_test( "defined_or_D", defined_or(get_value(vid),"default"), vid );
// circular_index() not tested

// end-of-tests
```

3.1.2.1.2 Results

```
1 ECHO: "OpenSCAD Version [2017, 2, 19]"
2 ECHO: "[ INFO ] run_test(); t01 passed: 'defined_or_D(undef)=default'"
3 ECHO: "[ INFO ] run_test(); t02 passed: 'defined_or_D([])=[]'"
4 ECHO: "[ INFO ] run_test(); t03 passed: 'defined_or_D([0 : 0.5 : 9])=[0 : 0.5 : 9]'"
5 ECHO: "[ INFO ] run_test(); t04 passed: 'defined_or_D(A string)=A string'"
6 ECHO: "[ INFO ] run_test(); t05 passed: 'defined_or_D(["orange", "apple", "grape", "banana"])=["orange",
      "apple", "grape", "banana"]'"
7 ECHO: "[ INFO ] run_test(); t06 passed: 'defined_or_D(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n",
      "a", "n", "a", "s"]'"
8 ECHO: "[ INFO ] run_test(); t07 passed: 'defined_or_D([undef])=[undef]'"
9 ECHO: "[ INFO ] run_test(); t08 passed: 'defined_or_D([[1, 2], [2, 3]])=[[1, 2], [2, 3]]'"
10 ECHO: "[ INFO ] run_test(); t09 passed: 'defined_or_D(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2,
      3], [4, 5]]'"
11 ECHO: "[ INFO ] run_test(); t10 passed: 'defined_or_D([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
      "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]'"
12 ECHO: "[ INFO ] run_test(); t11 passed: 'defined_or_D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
      15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
```

3.1.2.2 Iterables

- Script
- Results

3.1.2.2.1 Script

```
include <datatypes.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "The empty list",             empty_lst],
  ["t03", "A range",                    [0:0.5:9]],
  ["t04", "A string",                   "A string"],
  ["t05", "Test list 01",               ["orange","apple","grape","banana"]],
  ["t06", "Test list 02",               ["b","a","n","a","n","a","s"]],
  ["t07", "Test list 03",               [undef]],
  ["t08", "Test list 04",               [[1,2],[2,3]]],
  ["t09", "Test list 05",               ["ab",[1,2],[2,3],[4,5]]],
  ["t10", "Test list 06",               [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]]],
  ["t11", "Vector of integers 0 to 15", [for (i=[0:15]) i]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 's' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["edefined_or_DE3",
    "default",                                    // t01
    "default",                                    // t02
    "default",                                    // t03
    "t",                                          // t04
    "banana",                                     // t05
    "a",                                          // t06
    "default",                                    // t07
    "default",                                    // t08
    [4,5],                                        // t09
    ["a","b","c"],                                // t10
    3                                             // t11
  ],
  ["find_12",
    empty_lst,                                    // t01
    empty_lst,                                    // t02
    empty_lst,                                    // t03
    empty_lst,                                    // t04
    empty_lst,                                    // t05
    empty_lst,                                    // t06
    empty_lst,                                    // t07
    [0],                                          // t08
    [1],                                          // t09
    empty_lst,                                    // t10
    empty_lst                                     // t11
  ],
  ["count_S1",
    0,                                            // t01
    0,                                            // t02
    0,                                            // t03
    0,                                            // t04
    0,                                            // t05
    0,                                            // t06
    0,                                            // t07
    1,                                            // t08
```

```
    1,                                      // t09
    1,                                      // t10
    1                                       // t11
  ],
  ["exists_S1",
    false,                                  // t01
    false,                                  // t02
    false,                                  // t03
    false,                                  // t04
    false,                                  // t05
    false,                                  // t06
    false,                                  // t07
    true,                                   // t08
    true,                                   // t09
    true,                                   // t10
    true                                    // t11
  ],
  ["first",
    undef,                                  // t01
    undef,                                  // t02
    0,                                      // t03
    "A",                                    // t04
    "orange",                               // t05
    "b",                                    // t06
    undef,                                  // t07
    [1,2],                                  // t08
    "ab",                                   // t09
    [1,2,3],                                // t10
    0                                       // t11
  ],
  ["second",
    undef,                                  // t01
    undef,                                  // t02
    0.5,                                    // t03
    " ",                                    // t04
    "apple",                                // t05
    "a",                                    // t06
    undef,                                  // t07
    [2,3],                                  // t08
    [1,2],                                  // t09
    [4,5,6],                                // t10
    1                                       // t11
  ],
  ["third",
    undef,                                  // t01
    undef,                                  // t02
    9,                                      // t03
    "s",                                    // t04
    "grape",                                // t05
    "n",                                    // t06
    undef,                                  // t07
    undef,                                  // t08
    [2,3],                                  // t09
    [7,8,9],                                // t10
    2                                       // t11
  ],
  ["last",
    undef,                                  // t01
    undef,                                  // t02
    undef,                                  // t03
    "g",                                    // t04
    "banana",                               // t05
    "s",                                    // t06
    undef,                                  // t07
    [2,3],                                  // t08
    [4,5],                                  // t09
    ["a","b","c"],                          // t10
    15                                      // t11
  ],
  ["nfirst_1",
    undef,                                  // t01
    undef,                                  // t02
    undef,                                  // t03
    ["A"],                                  // t04
    ["orange"],                             // t05
    ["b"],                                  // t06
    [undef],                                // t07
    [[1,2]],                                // t08
    ["ab"],                                 // t09
    [[1,2,3]],                              // t10
    [0]                                     // t11
```

```
],
["nlast_1",
  undef,                                           // t01
  undef,                                           // t02
  undef,                                           // t03
  ["g"],                                           // t04
  ["banana"],                                      // t05
  ["s"],                                           // t06
  [undef],                                         // t07
  [[2,3]],                                         // t08
  [[4,5]],                                         // t09
  [["a","b","c"]],                                 // t10
  [15]                                             // t11
],
["nhead_1",
  undef,                                           // t01
  undef,                                           // t02
  undef,                                           // t03
  ["A"," ","s","t","r","i","n"],                   // t04
  ["orange","apple","grape"],                      // t05
  ["b","a","n","a","n","a"],                       // t06
  empty_lst,                                       // t07
  [[1,2]],                                         // t08
  ["ab",[1,2],[2,3]],                              // t09
  [[1,2,3],[4,5,6],[7,8,9]],                       // t10
  [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]             // t11
],
["ntail_1",
  undef,                                           // t01
  undef,                                           // t02
  undef,                                           // t03
  [" ","s","t","r","i","n","g"],                   // t04
  ["apple","grape","banana"],                      // t05
  ["a","n","a","n","a","s"],                       // t06
  empty_lst,                                       // t07
  [[2,3]],                                         // t08
  [[1,2],[2,3],[4,5]],                             // t09
  [[4,5,6],[7,8,9],["a","b","c"]],                 // t10
  [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]            // t11
],
["reverse",
  undef,                                           // t01
  empty_lst,                                       // t02
  undef,                                           // t03
  ["g","n","i","r","t","s"," ","A"],               // t04
  ["banana","grape","apple","orange"],             // t05
  ["s","a","n","a","n","a","b"],                   // t06
  [undef],                                         // t07
  [[2,3],[1,2]],                                   // t08
  [[4,5],[2,3],[1,2],"ab"],                        // t09
  [["a","b","c"],[7,8,9],[4,5,6],[1,2,3]],         // t10
  [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]          // t11
],
["rselect_02",
  undef,                                           // t01
  empty_lst,                                       // t02
  undef,                                           // t03
  ["A"," ","s"],                                   // t04
  ["orange","apple","grape"],                      // t05
  ["b","a","n"],                                   // t06
  undef,                                           // t07
  undef,                                           // t08
  ["ab",[1,2],[2,3]],                              // t09
  [[1,2,3],[4,5,6],[7,8,9]],                       // t10
  [0,1,2]                                          // t11
],
["nssequence_31",
  empty_lst,                                       // t01
  empty_lst,                                       // t02
  empty_lst,                                       // t03
  [
    ["A"," ","s"],[" ","s","t"],["s","t","r"],
    ["t","r","i"],["r","i","n"],["i","n","g"]
  ],                                               // t04
  [
    ["orange","apple","grape"],
    ["apple","grape","banana"]
  ],                                               // t05
  [
    ["b","a","n"],["a","n","a"],["n","a","n"],
    ["a","n","a"],["n","a","s"]
```

```
    ],                                                      // t06
    empty_lst,                                              // t07
    empty_lst,                                              // t08
    [["ab",[1,2],[2,3]],[[1,2],[2,3],[4,5]]],               // t09
    [
      [[1,2,3],[4,5,6],[7,8,9]],
      [[4,5,6],[7,8,9],["a","b","c"]]
    ],                                                      // t10
    [
      [0,1,2],[1,2,3],[2,3,4],[3,4,5],[4,5,6],[5,6,7],
      [6,7,8],[7,8,9],[8,9,10],[9,10,11],[10,11,12],
      [11,12,13],[12,13,14],[13,14,15]
    ]                                                       // t11
  ],
  ["eappend_T0",
    undef,                                                  // t01
    [[0]],                                                  // t02
    undef,                                                  // t03
    [
      ["A",0],[" ",0],["s",0],["t",0],
      ["r",0],["i",0],["n",0],["g",0]
    ],                                                      // t04
    [
      ["orange",0],["apple",0],
      ["grape",0],["banana",0]
    ],                                                      // t05
    [
      ["b",0],["a",0],["n",0],["a",0],
      ["n",0],["a",0],["s",0]
    ],                                                      // t06
    [[undef,0]],                                            // t07
    [[1,2,0],[2,3,0]],                                      // t08
    [["ab",0],[1,2,0],[2,3,0],[4,5,0]],                     // t09
    [[1,2,3,0],[4,5,6,0],[7,8,9,0],["a","b","c",0]],        // t10
    [
      [0,0],[1,0],[2,0],[3,0],[4,0],[5,0],
      [6,0],[7,0],[8,0],[9,0],[10,0],[11,0],
      [12,0],[13,0],[14,0],[15,0]
    ]                                                       // t11
  ],
  ["insert_T0",
    undef,                                                  // t01
    undef,                                                  // t02
    undef,                                                  // t03
    undef,                                                  // t04
    ["orange",0,"apple","grape","banana"],                  // t05
    ["b","a","n","a","n","a",0,"s"],                        // t06
    undef,                                                  // t07
    [[1,2],0,[2,3]],                                        // t08
    ["ab",[1,2],0,[2,3],[4,5]],                             // t09
    undef,                                                  // t10
    [0,1,2,3,4,0,5,6,7,8,9,10,11,12,13,14,15]               // t11
  ],
  ["delete_T0",
    undef,                                                  // t01
    empty_lst,                                              // t02
    undef,                                                  // t03
    ["A"," ","s","t","r","i","n","g"],                      // t04
    ["orange","grape","banana"],                            // t05
    ["b","a","n","a","n","a"],                              // t06
    [undef],                                                // t07
    [[1,2]],                                                // t08
    ["ab",[1,2],[4,5]],                                     // t09
    [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],                // t10
    [0,1,2,3,4,6,7,8,9,10,11,12,13,14,15]                   // t11
  ],
  ["strip",
    undef,                                                  // t01
    empty_lst,                                              // t02
    undef,                                                  // t03
    ["A"," ","s","t","r","i","n","g"],                      // t04
    ["orange","apple","grape","banana"],                    // t05
    ["b","a","n","a","n","a","s"],                          // t06
    [undef],                                                // t07
    [[1,2],[2,3]],                                          // t08
    ["ab",[1,2],[2,3],[4,5]],                               // t09
    [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],                // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]                 // t11
  ],
  ["unique",
    undef,                                                  // t01
```

```
      empty_lst,                                        // t02
      undef,                                            // t03
      ["A"," ","s","t","r","i","n","g"],                // t04
      ["orange","apple","grape","banana"],              // t05
      ["b","a","n","s"],                                // t06
      [undef],                                          // t07
      [[1,2],[2,3]],                                    // t08
      ["ab",[1,2],[2,3],[4,5]],                         // t09
      [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]],          // t10
      [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]           // t11
    ]
  ];

  // sanity-test tables
  table_check( test_r, test_c, false );
  table_check( good_r, good_c, false );

  // validate helper function and module
  function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
  module run_test( fname, fresult, vid )
  {
    value_text = get_table_v(test_r, test_c, vid, "td");
    pass_value = get_table_v(good_r, good_c, fname, vid);

    test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
    test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals",
    pass_value );

    if ( pass_value != skip )
    {
      if ( !test_pass )
        log_warn( str(vid, "(", value_text, ") ", test_text) );
      else if ( show_passing )
        log_info( str(vid, " ", test_text) );
    }
    else if ( show_skipped )
      log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
  }

  // Indirect function calls would be very useful here!!!
  for (vid=test_ids) run_test( "edefined_or_DE3", edefined_or(get_value(vid),3,"default"), vid
    );
  for (vid=test_ids) run_test( "find_12", find([1,2],get_value(vid)), vid );
  for (vid=test_ids) run_test( "count_S1", count(1,get_value(vid),true), vid );
  for (vid=test_ids) run_test( "exists_S1", exists(1,get_value(vid),true), vid );
  for (vid=test_ids) run_test( "first", first(get_value(vid)), vid );
  for (vid=test_ids) run_test( "second", second(get_value(vid)), vid );
  for (vid=test_ids) run_test( "third", third(get_value(vid)), vid );
  for (vid=test_ids) run_test( "last", last(get_value(vid)), vid );
  for (vid=test_ids) run_test( "nfirst_1", nfirst(get_value(vid),n=1), vid );
  for (vid=test_ids) run_test( "nlast_1", nlast(get_value(vid),n=1), vid );
  for (vid=test_ids) run_test( "nhead_1", nhead(get_value(vid),n=1), vid );
  for (vid=test_ids) run_test( "ntail_1", ntail(get_value(vid),n=1), vid );
  for (vid=test_ids) run_test( "reverse", reverse(get_value(vid)), vid );
  for (vid=test_ids) run_test( "rselect_02", rselect(get_value(vid),i=[0:2]), vid );
  for (vid=test_ids) run_test( "nssequence_31", nssequence(get_value(vid),n=3,s=1), vid );
  for (vid=test_ids) run_test( "eappend_T0", eappend(0,get_value(vid)), vid );
  for (vid=test_ids) run_test( "insert_T0", insert(0,get_value(vid),mv=["x","r","apple","s",[2,3],5
    ]), vid );
  for (vid=test_ids) run_test( "delete_T0", delete(get_value(vid),mv=["x","r","apple","s",[2,3],5]), vid
    );
  for (vid=test_ids) run_test( "strip", strip(get_value(vid)), vid );
  for (vid=test_ids) run_test( "unique", unique(get_value(vid)), vid );

  // end-of-tests
```

3.1.2.2.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run_test(); t01 passed: 'edefined_or_DE3(undef)=default'"
ECHO: "[ INFO ] run_test(); t02 passed: 'edefined_or_DE3([])=default'"
ECHO: "[ INFO ] run_test(); t03 passed: 'edefined_or_DE3([0 : 0.5 : 9])=default'"
ECHO: "[ INFO ] run_test(); t04 passed: 'edefined_or_DE3(A string)=t'"
ECHO: "[ INFO ] run_test(); t05 passed: 'edefined_or_DE3(["orange", "apple", "grape", "banana"])=banana'"
ECHO: "[ INFO ] run_test(); t06 passed: 'edefined_or_DE3(["b", "a", "n", "a", "n", "a", "s"])=a'"
ECHO: "[ INFO ] run_test(); t07 passed: 'edefined_or_DE3([undef])=default'"
ECHO: "[ INFO ] run_test(); t08 passed: 'edefined_or_DE3([[1, 2], [2, 3]])=default'"
ECHO: "[ INFO ] run_test(); t09 passed: 'edefined_or_DE3(["ab", [1, 2], [2, 3], [4, 5]])=[4, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'edefined_or_DE3([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
```

```
    "c"]])=["a", "b", "c"]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’edefined_or_DE3([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=3’"
ECHO: "[ INFO ] run_test(); t01 passed: ’find_12(undef)=[]’"
ECHO: "[ INFO ] run_test(); t02 passed: ’find_12([])=[]’"
ECHO: "[ INFO ] run_test(); t03 passed: ’find_12([0 : 0.5 : 9])=[]’"
ECHO: "[ INFO ] run_test(); t04 passed: ’find_12(A string)=[]’"
ECHO: "[ INFO ] run_test(); t05 passed: ’find_12(["orange", "apple", "grape", "banana"])=[]’"
ECHO: "[ INFO ] run_test(); t06 passed: ’find_12(["b", "a", "n", "a", "n", "a", "s"])=[]’"
ECHO: "[ INFO ] run_test(); t07 passed: ’find_12([undef])=[]’"
ECHO: "[ INFO ] run_test(); t08 passed: ’find_12([[1, 2], [2, 3]])=[0]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’find_12(["ab", [1, 2], [2, 3], [4, 5]])=[1]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’find_12([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’find_12([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=[]’"
ECHO: "[ INFO ] run_test(); t01 passed: ’count_S1(undef)=0’"
ECHO: "[ INFO ] run_test(); t02 passed: ’count_S1([])=0’"
ECHO: "[ INFO ] run_test(); t03 passed: ’count_S1([0 : 0.5 : 9])=0’"
ECHO: "[ INFO ] run_test(); t04 passed: ’count_S1(A string)=0’"
ECHO: "[ INFO ] run_test(); t05 passed: ’count_S1(["orange", "apple", "grape", "banana"])=0’"
ECHO: "[ INFO ] run_test(); t06 passed: ’count_S1(["b", "a", "n", "a", "n", "a", "s"])=0’"
ECHO: "[ INFO ] run_test(); t07 passed: ’count_S1([undef])=0’"
ECHO: "[ INFO ] run_test(); t08 passed: ’count_S1([[1, 2], [2, 3]])=1’"
ECHO: "[ INFO ] run_test(); t09 passed: ’count_S1(["ab", [1, 2], [2, 3], [4, 5]])=1’"
ECHO: "[ INFO ] run_test(); t10 passed: ’count_S1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=1’"
ECHO: "[ INFO ] run_test(); t11 passed: ’count_S1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=1’"
ECHO: "[ INFO ] run_test(); t01 passed: ’exists_S1(undef)=false’"
ECHO: "[ INFO ] run_test(); t02 passed: ’exists_S1([])=false’"
ECHO: "[ INFO ] run_test(); t03 passed: ’exists_S1([0 : 0.5 : 9])=false’"
ECHO: "[ INFO ] run_test(); t04 passed: ’exists_S1(A string)=false’"
ECHO: "[ INFO ] run_test(); t05 passed: ’exists_S1(["orange", "apple", "grape", "banana"])=false’"
ECHO: "[ INFO ] run_test(); t06 passed: ’exists_S1(["b", "a", "n", "a", "n", "a", "s"])=false’"
ECHO: "[ INFO ] run_test(); t07 passed: ’exists_S1([undef])=false’"
ECHO: "[ INFO ] run_test(); t08 passed: ’exists_S1([[1, 2], [2, 3]])=true’"
ECHO: "[ INFO ] run_test(); t09 passed: ’exists_S1(["ab", [1, 2], [2, 3], [4, 5]])=true’"
ECHO: "[ INFO ] run_test(); t10 passed: ’exists_S1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=true’"
ECHO: "[ INFO ] run_test(); t11 passed: ’exists_S1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=true’"
ECHO: "[ INFO ] run_test(); t01 passed: ’first(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’first([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’first([0 : 0.5 : 9])=0’"
ECHO: "[ INFO ] run_test(); t04 passed: ’first(A string)=A’"
ECHO: "[ INFO ] run_test(); t05 passed: ’first(["orange", "apple", "grape", "banana"])=orange’"
ECHO: "[ INFO ] run_test(); t06 passed: ’first(["b", "a", "n", "a", "n", "a", "s"])=b’"
ECHO: "[ INFO ] run_test(); t07 passed: ’first([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’first([[1, 2], [2, 3]])=[1, 2]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’first(["ab", [1, 2], [2, 3], [4, 5]])=ab’"
ECHO: "[ INFO ] run_test(); t10 passed: ’first([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 2,
    3]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’first([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=0’"
ECHO: "[ INFO ] run_test(); t01 passed: ’second(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’second([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’second([0 : 0.5 : 9])=0.5’"
ECHO: "[ INFO ] run_test(); t04 passed: ’second(A string)= ’"
ECHO: "[ INFO ] run_test(); t05 passed: ’second(["orange", "apple", "grape", "banana"])=apple’"
ECHO: "[ INFO ] run_test(); t06 passed: ’second(["b", "a", "n", "a", "n", "a", "s"])=a’"
ECHO: "[ INFO ] run_test(); t07 passed: ’second([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’second([[1, 2], [2, 3]])=[2, 3]’"
ECHO: "[ INFO ] run_test(); t09 passed: ’second(["ab", [1, 2], [2, 3], [4, 5]])=[1, 2]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’second([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[4, 5,
    6]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’second([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=1’"
ECHO: "[ INFO ] run_test(); t01 passed: ’third(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’third([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’third([0 : 0.5 : 9])=9’"
ECHO: "[ INFO ] run_test(); t04 passed: ’third(A string)=s’"
ECHO: "[ INFO ] run_test(); t05 passed: ’third(["orange", "apple", "grape", "banana"])=grape’"
ECHO: "[ INFO ] run_test(); t06 passed: ’third(["b", "a", "n", "a", "n", "a", "s"])=n’"
ECHO: "[ INFO ] run_test(); t07 passed: ’third([undef])=undef’"
ECHO: "[ INFO ] run_test(); t08 passed: ’third([[1, 2], [2, 3]])=undef’"
ECHO: "[ INFO ] run_test(); t09 passed: ’third(["ab", [1, 2], [2, 3], [4, 5]])=[2, 3]’"
ECHO: "[ INFO ] run_test(); t10 passed: ’third([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[7, 8,
    9]’"
ECHO: "[ INFO ] run_test(); t11 passed: ’third([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=2’"
ECHO: "[ INFO ] run_test(); t01 passed: ’last(undef)=undef’"
ECHO: "[ INFO ] run_test(); t02 passed: ’last([])=undef’"
ECHO: "[ INFO ] run_test(); t03 passed: ’last([0 : 0.5 : 9])=undef’"
ECHO: "[ INFO ] run_test(); t04 passed: ’last(A string)=g’"
ECHO: "[ INFO ] run_test(); t05 passed: ’last(["orange", "apple", "grape", "banana"])=banana’"
```

```
ECHO: "[ INFO ] run_test(); t06 passed: 'last(["b", "a", "n", "a", "n", "a", "s"])=s'"
ECHO: "[ INFO ] run_test(); t07 passed: 'last([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'last([[1, 2], [2, 3]])=[2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'last(["ab", [1, 2], [2, 3], [4, 5]])=[4, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'last([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=["a", "b", "c"]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'last([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=15'"
ECHO: "[ INFO ] run_test(); t01 passed: 'nfirst_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'nfirst_1([])=undef'"
ECHO: "[ INFO ] run_test(); t03 passed: 'nfirst_1([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'nfirst_1(A string)=["A"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'nfirst_1(["orange", "apple", "grape", "banana"])=["orange"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'nfirst_1(["b", "a", "n", "a", "n", "a", "s"])=["b"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'nfirst_1([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'nfirst_1([[1, 2], [2, 3]])=[[1, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'nfirst_1(["ab", [1, 2], [2, 3], [4, 5]])=["ab"]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'nfirst_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'nfirst_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'nlast_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'nlast_1([])=undef'"
ECHO: "[ INFO ] run_test(); t03 passed: 'nlast_1([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'nlast_1(A string)=["g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'nlast_1(["orange", "apple", "grape", "banana"])=["banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'nlast_1(["b", "a", "n", "a", "n", "a", "s"])=["s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'nlast_1([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'nlast_1([[1, 2], [2, 3]])=[[2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'nlast_1(["ab", [1, 2], [2, 3], [4, 5]])=[[4, 5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'nlast_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'nlast_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'nhead_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'nhead_1([])=undef'"
ECHO: "[ INFO ] run_test(); t03 passed: 'nhead_1([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'nhead_1(A string)=["A", " ", "s", "t", "r", "i", "n"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'nhead_1(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'nhead_1(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'nhead_1([undef])=[]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'nhead_1([[1, 2], [2, 3]])=[[1, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'nhead_1(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'nhead_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'nhead_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'ntail_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'ntail_1([])=undef'"
ECHO: "[ INFO ] run_test(); t03 passed: 'ntail_1([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'ntail_1(A string)=[" ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'ntail_1(["orange", "apple", "grape", "banana"])=["apple", "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'ntail_1(["b", "a", "n", "a", "n", "a", "s"])=["a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'ntail_1([undef])=[]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'ntail_1([[1, 2], [2, 3]])=[[2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'ntail_1(["ab", [1, 2], [2, 3], [4, 5]])=[[1, 2], [2, 3], [4, 5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'ntail_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[4, 5, 6], [7, 8, 9], ["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'ntail_1([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'reverse(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'reverse([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'reverse([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'reverse(A string)=["g", "n", "i", "r", "t", "s", " ", "A"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'reverse(["orange", "apple", "grape", "banana"])=["banana", "grape", "apple", "orange"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'reverse(["b", "a", "n", "a", "n", "a", "s"])=["s", "a", "n", "a", "n", "a", "b"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'reverse([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'reverse([[1, 2], [2, 3]])=[[2, 3], [1, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'reverse(["ab", [1, 2], [2, 3], [4, 5]])=[[4, 5], [2, 3], [1, 2], "ab"]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'reverse([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[["a", "b", "c"], [7, 8, 9], [4, 5, 6], [1, 2, 3]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'reverse([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'rselect_02(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'rselect_02([])=[]'"
```

```
ECHO: "[ INFO ] run_test(); t03 passed: 'rselect_02([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'rselect_02(A string)=["A", " ", "s"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'rselect_02(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'rselect_02(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'rselect_02([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'rselect_02([[1, 2], [2, 3]])=undef'"
ECHO: "[ INFO ] run_test(); t09 passed: 'rselect_02(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'rselect_02([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'rselect_02([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'nssequence_31(undef)=[]'"
ECHO: "[ INFO ] run_test(); t02 passed: 'nssequence_31([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'nssequence_31([0 : 0.5 : 9])=[]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'nssequence_31(A string)=[["A", " ", "s"], [" ", "s", "t"], ["s", "t", "r"], ["t", "r", "i"], ["r", "i", "n"], ["i", "n", "g"]]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'nssequence_31(["orange", "apple", "grape", "banana"])=[["orange", "apple", "grape"], ["apple", "grape", "banana"]]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'nssequence_31(["b", "a", "n", "a", "n", "a", "s"])=[["b", "a", "n"], ["a", "n", "a"], ["n", "a", "n"], ["a", "n", "a"], ["n", "a", "s"]]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'nssequence_31([undef])=[]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'nssequence_31([[1, 2], [2, 3]])=[]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'nssequence_31(["ab", [1, 2], [2, 3], [4, 5]])=[["ab", [1, 2], [2, 3]], [[1, 2], [2, 3], [4, 5]]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'nssequence_31([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[4, 5, 6], [7, 8, 9], ["a", "b", "c"]]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'nssequence_31([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[[0, 1, 2], [1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 7], [6, 7, 8], [7, 8, 9], [8, 9, 10], [9, 10, 11], [10, 11, 12], [11, 12, 13], [12, 13, 14], [13, 14, 15]]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'eappend_T0(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eappend_T0([])=[[0]]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eappend_T0([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'eappend_T0(A string)=[["A", 0], [" ", 0], ["s", 0], ["t", 0], ["r", 0], ["i", 0], ["n", 0], ["g", 0]]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eappend_T0(["orange", "apple", "grape", "banana"])=[["orange", 0], ["apple", 0], ["grape", 0], ["banana", 0]]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'eappend_T0(["b", "a", "n", "a", "n", "a", "s"])=[["b", 0], ["a", 0], ["n", 0], ["a", 0], ["n", 0], ["a", 0], ["s", 0]]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eappend_T0([undef])=[[undef, 0]]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eappend_T0([[1, 2], [2, 3]])=[[1, 2, 0], [2, 3, 0]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eappend_T0(["ab", [1, 2], [2, 3], [4, 5]])=[["ab", 0], [1, 2, 0], [2, 3, 0], [4, 5, 0]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eappend_T0([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3, 0], [4, 5, 6, 0], [7, 8, 9, 0], ["a", "b", "c", 0]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'eappend_T0([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[[0, 0], [1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0], [7, 0], [8, 0], [9, 0], [10, 0], [11, 0], [12, 0], [13, 0], [14, 0], [15, 0]]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'insert_T0(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'insert_T0([])=undef'"
ECHO: "[ INFO ] run_test(); t03 passed: 'insert_T0([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'insert_T0(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'insert_T0(["orange", "apple", "grape", "banana"])=["orange", 0, "apple", "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'insert_T0(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", 0, "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'insert_T0([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'insert_T0([[1, 2], [2, 3]])=[[1, 2], 0, [2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'insert_T0(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], 0, [2, 3], [4, 5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'insert_T0([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=undef'"
ECHO: "[ INFO ] run_test(); t11 passed: 'insert_T0([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 0, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'delete_T0(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'delete_T0([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'delete_T0([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'delete_T0(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'delete_T0(["orange", "apple", "grape", "banana"])=["orange", "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'delete_T0(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'delete_T0([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'delete_T0([[1, 2], [2, 3]])=[[1, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'delete_T0(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [4, 5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'delete_T0([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'delete_T0([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'strip(undef)=undef'"
```

```
ECHO: "[ INFO ] run_test(); t02 passed: 'strip([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'strip([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'strip(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'strip(["orange", "apple", "grape", "banana"])=["orange", "apple",
    "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'strip(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a",
    "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'strip([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'strip([[1, 2], [2, 3]])=[[1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'strip(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3], [4,
    5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'strip([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2,
    3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'strip([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'unique(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'unique([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'unique([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'unique(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'unique(["orange", "apple", "grape", "banana"])=["orange", "apple",
    "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'unique(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'unique([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'unique([[1, 2], [2, 3]])=[[1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'unique(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3], [4,
    5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'unique([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2,
    3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'unique([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
```

3.1.2.3 Lists

- Script
- Results

3.1.2.3.1 Script

```
include <datatypes.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["t01", "The undefined value",        undef],
  ["t02", "The empty list",             empty_lst],
  ["t03", "A range",                    [0:0.5:9]],
  ["t04", "A string",                   "A string"],
  ["t05", "Test list 01",               ["orange","apple","grape","banana"]],
  ["t06", "Test list 02",               ["b","a","n","a","n","a","s"]],
  ["t07", "Test list 03",               [undef]],
  ["t08", "Test list 04",               [[1,2],[2,3]]],
  ["t09", "Test list 05",               ["ab",[1,2],[2,3],[4,5]]],
  ["t10", "Test list 06",               [[1,2,3],[4,5,6],[7,8,9],["a","b","c"]]],
  ["t11", "Vector of integers 0 to 15", [for (i=[0:15]) i]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);
```

```
// expected rows: ("golden" test results), use 's' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["lstr",
    undef,                                          // t01
    empty_str,                                      // t02
    "[0 : 0.5 : 9]",                                // t03
    "A string",                                     // t04
    "orangeapplegrapebanana",                       // t05
    "bananas",                                      // t06
    "undef",                                        // t07
    "[1, 2][2, 3]",                                 // t08
    "ab[1, 2][2, 3][4, 5]",                         // t09
    "[1, 2, 3][4, 5, 6][7, 8, 9][\"a\", \"b\", \"c\"]", // t10
    "0123456789101112131415"                        // t11
  ],
  ["lstr_html_B",
    "<b>undef</b>",                                 // t01
    empty_str,                                      // t02
    "<b>[0 : 0.5 : 9]</b>",                         // t03
    "<b>A string</b>",                              // t04
    "<b>orange</b><b>apple</b><b>grape</b><b>banana</b>",
    "<b>b</b><b>a</b><b>n</b><b>a</b><b>n</b><b>a</b><b>s</b>",
    "<b>undef</b>",                                 // t07
    "<b>[1, 2]</b><b>[2, 3]</b>",                   // t08
    "<b>ab</b><b>[1, 2]</b><b>[2, 3]</b><b>[4, 5]</b>", // t09
    "<b>[1, 2, 3]</b><b>[4, 5, 6]</b><b>[7, 8, 9]</b><b>[\"a\", \"b\", \"c\"]</b>",
    "
  <b>0</b><b>1</b><b>2</b><b>3</b><b>4</b><b>5</b><b>6</b><b>7</b><b>8</b><b>9</b><b>10</b><b>11</b><b>12</b><b>13</b><b>14</
  ],
  ["consts",
    empty_lst,                                      // t01
    empty_lst,                                      // t02
    empty_lst,                                      // t03
    empty_lst,                                      // t04
    empty_lst,                                      // t05
    empty_lst,                                      // t06
    empty_lst,                                      // t07
    empty_lst,                                      // t08
    empty_lst,                                      // t09
    empty_lst,                                      // t10
    empty_lst                                       // t11
  ],
  ["get_index",
    empty_lst,                                      // t01
    empty_lst,                                      // t02
    empty_lst,                                      // t03
    [0,1,2,3,4,5,6,7],                              // t04
    [0,1,2,3],                                      // t05
    [0,1,2,3,4,5,6],                                // t06
    [0],                                            // t07
    [0,1],                                          // t08
    [0,1,2,3],                                      // t09
    [0,1,2,3],                                      // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]         // t11
  ],
  ["pad_9",
    [undef,0,0,0,0,0,0,0,0],                        // t01
    [0,0,0,0,0,0,0,0,0],                            // t02
    [[0:0.5:9],0,0,0,0,0,0,0,0],                    // t03
    ["A string",0],                                 // t04
    ["orange","apple","grape","banana",0,0,0,0,0],  // t05
    ["b","a","n","a","n","a","s",0,0],              // t06
    [undef,0,0,0,0,0,0,0,0],                        // t07
    [[1,2],[2,3],0,0,0,0,0,0,0],                    // t08
    ["ab",[1,2],[2,3],[4,5],0,0,0,0,0],             // t09
    [[1,2,3],[4,5,6],[7,8,9],["a","b","c"],0,0,0,0,0], // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]         // t11
  ],
  ["limit_12",
    undef,                                          // t01
    empty_lst,                                      // t02
    [0:0.5:9],                                      // t03
    "A string",                                     // t04
    ["orange","apple","grape","banana"],            // t05
    ["b","a","n","a","n","a","s"],                  // t06
    [undef],                                        // t07
    [[1,2],[2,2]],                                  // t08
```

```
    ["ab",[1,2],[2,2],[2,2]],                              // t09
    [[1,2,2],[2,2,2],[2,2,2],["a","b","c"]],               // t10
    [1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2]                      // t11
  ],
  ["sum",
    undef,                                                 // t01
    0,                                                     // t02
    85.5,                                                  // t03
    undef,                                                 // t04
    undef,                                                 // t05
    undef,                                                 // t06
    undef,                                                 // t07
    [3,5],                                                 // t08
    undef,                                                 // t09
    [undef,undef,undef],                                   // t10
    120                                                    // t11
  ],
  ["mean",
    undef,                                                 // t01
    0,                                                     // t02
    4.5,                                                   // t03
    undef,                                                 // t04
    undef,                                                 // t05
    undef,                                                 // t06
    undef,                                                 // t07
    [1.5,2.5],                                             // t08
    undef,                                                 // t09
    [undef,undef,undef],                                   // t10
    7.5                                                    // t11
  ],
  ["eselect_F",
    undef,                                                 // t01
    empty_lst,                                             // t02
    undef,                                                 // t03
    ["A"," ","s","t","r","i","n","g"],                     // t04
    ["o","a","g","b"],                                     // t05
    ["b","a","n","a","n","a","s"],                         // t06
    [undef],                                               // t07
    [1,2],                                                 // t08
    ["a",1,2,4],                                           // t09
    [1,4,7,"a"],                                           // t10
    skip                                                   // t11
  ],
  ["eselect_L",
    undef,                                                 // t01
    empty_lst,                                             // t02
    undef,                                                 // t03
    ["A"," ","s","t","r","i","n","g"],                     // t04
    ["e","e","e","a"],                                     // t05
    ["b","a","n","a","n","a","s"],                         // t06
    [undef],                                               // t07
    [2,3],                                                 // t08
    ["b",2,3,5],                                           // t09
    [3,6,9,"c"],                                           // t10
    skip                                                   // t11
  ],
  ["eselect_1",
    undef,                                                 // t01
    empty_lst,                                             // t02
    undef,                                                 // t03
    skip,                                                  // t04
    ["r","p","r","a"],                                     // t05
    skip,                                                  // t06
    [undef],                                               // t07
    [2,3],                                                 // t08
    ["b",2,3,5],                                           // t09
    [2,5,8,"b"],                                           // t10
    skip                                                   // t11
  ],
  ["smerge",
    undef,                                                 // t01
    empty_lst,                                             // t02
    [[0:0.5:9]],                                           // t03
    ["A string"],                                          // t04
    ["orange","apple","grape","banana"],                   // t05
    ["b","a","n","a","n","a","s"],                         // t06
    [undef],                                               // t07
    [1,2,2,3],                                             // t08
    ["ab",1,2,2,3,4,5],                                    // t09
    [1,2,3,4,5,6,7,8,9,"a","b","c"],                       // t10
    [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]                // t11
```

```
    ],
    ["pmerge",
      undef,                                          // t01
      empty_lst,                                      // t02
      undef,                                          // t03
      ["A string"],                                   // t04
      [
        ["o","a","g","b"],["r","p","r","a"],
        ["a","p","a","n"],["n","l","p","a"],
        ["g","e","e","n"]
      ],                                              // t05
      [["b","a","n","a","n","a","s"]],                // t06
      undef,                                          // t07
      [[1,2],[2,3]],                                  // t08
      [["a",1,2,4],["b",2,3,5]],                      // t09
      [[1,4,7,"a"],[2,5,8,"b"],[3,6,9,"c"]],          // t10
      undef                                           // t11
    ],
    ["qsort",
      undef,                                          // t01
      empty_lst,                                      // t02
      undef,                                          // t03
      undef,                                          // t04
      ["apple","banana","grape","orange"],            // t05
      ["a","a","a","b","n","n","s"],                  // t06
      [undef],                                        // t07
      skip,                                           // t08
      skip,                                           // t09
      skip,                                           // t10
      [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]         // t11
    ],
    ["qsort_1R",
      undef,                                          // t01
      empty_lst,                                      // t02
      undef,                                          // t03
      undef,                                          // t04
      ["orange","grape","apple","banana"],            // t05
      skip,                                           // t06
      skip,                                           // t07
      [[2,3],[1,2]],                                  // t08
      [[4,5],[2,3],[1,2],"ab"],                       // t09
      [[7,8,9],[4,5,6],[1,2,3],["a","b","c"]],        // t10
      skip                                            // t11
    ],
    ["qsort2_1R",
      undef,                                          // t01
      empty_lst,                                      // t02
      undef,                                          // t03
      undef,                                          // t04
      ["orange","grape","apple","banana"],            // t05
      skip,                                           // t06
      skip,                                           // t07
      [[2,3],[1,2]],                                  // t08
      ["ab",[4,5],[2,3],[1,2]],                       // t09
      [["a","b","c"],[7,8,9],[4,5,6],[1,2,3]],        // t10
      skip                                            // t11
    ],
    ["qsort2_HR",
      undef,                                          // t01
      empty_lst,                                      // t02
      undef,                                          // t03
      undef,                                          // t04
      ["orange","grape","banana","apple"],            // t05
      ["s","n","n","b","a","a","a"],                  // t06
      [undef],                                        // t07
      [[3,2],[2,1]],                                  // t08
      [[5,4],[3,2],[2,1],"ab"],                       // t09
      [["c","b","a"],[9,8,7],[6,5,4],[3,2,1]],        // t10
      [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]         // t11
    ]
  ];

  // sanity-test tables
  table_check( test_r, test_c, false );
  table_check( good_r, good_c, false );

  // validate helper function and module
  function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
  module run_test( fname, fresult, vid )
  {
    value_text = get_table_v(test_r, test_c, vid, "td");
```

```
    pass_value = get_table_v(good_r, good_c, fname, vid);

    test_pass = validate( cv=fresult, t="equals", ev=pass_value, pf=true );
    test_text = validate( str(fname, "(", get_value(vid), ")=", pass_value), fresult, "equals",
    pass_value );

    if ( pass_value != skip )
    {
      if ( !test_pass )
        log_warn( str(vid, "(", value_text, ") ", test_text) );
      else if ( show_passing )
        log_info( str(vid, " ", test_text) );
    }
    else if ( show_skipped )
      log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
  }

  // Indirect function calls would be very useful here!!!
  for (vid=test_ids) run_test( "lstr", lstr(get_value(vid)), vid );
  for (vid=test_ids) run_test( "lstr_html_B", lstr_html(get_value(vid),p="b"), vid );
  for (vid=test_ids) run_test( "consts", consts(get_value(vid)), vid );
  for (vid=test_ids) run_test( "get_index", get_index(get_value(vid)), vid );
  for (vid=test_ids) run_test( "pad_9", pad(get_value(vid), w=9), vid );
  log_info( "not testing: dround()" );
  log_info( "not testing: sround()" );
  for (vid=test_ids) run_test( "limit_12", limit(get_value(vid),1,2), vid );
  for (vid=test_ids) run_test( "sum", sum(get_value(vid)), vid );
  for (vid=test_ids) run_test( "mean", mean(get_value(vid)), vid );
  log_info( "not testing: ciselect()" );
  log_info( "not testing: cmvselect()" );
  for (vid=test_ids) run_test( "eselect_F", eselect(get_value(vid),f=true), vid );
  for (vid=test_ids) run_test( "eselect_L", eselect(get_value(vid),l=true), vid );
  for (vid=test_ids) run_test( "eselect_1", eselect(get_value(vid),i=1), vid );
  for (vid=test_ids) run_test( "smerge", smerge(get_value(vid)), vid );
  for (vid=test_ids) run_test( "pmerge", pmerge(get_value(vid)), vid );
  for (vid=test_ids) run_test( "qsort", qsort(get_value(vid)), vid );
  for (vid=test_ids) run_test( "qsort_1R", qsort(get_value(vid), i=1, r=true), vid );
  for (vid=test_ids) run_test( "qsort2_1R", qsort2(get_value(vid), i=1, r=true), vid );
  for (vid=test_ids) run_test( "qsort2_HR", qsort2(get_value(vid), d=5, r=true), vid );

  // end-of-tests
```

3.1.2.3.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run_test(); t01 passed: 'lstr(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'lstr([])='"
ECHO: "[ INFO ] run_test(); t03 passed: 'lstr([0 : 0.5 : 9])=[0 : 0.5 : 9]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'lstr(A string)=A string'"
ECHO: "[ INFO ] run_test(); t05 passed: 'lstr(["orange", "apple", "grape",
      "banana"])=orangeapplegrapebanana'"
ECHO: "[ INFO ] run_test(); t06 passed: 'lstr(["b", "a", "n", "a", "n", "a", "s"])=bananas'"
ECHO: "[ INFO ] run_test(); t07 passed: 'lstr([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'lstr([[1, 2], [2, 3]])=[1, 2][2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'lstr(["ab", [1, 2], [2, 3], [4, 5]])=ab[1, 2][2, 3][4, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'lstr([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 2,
      3][4, 5, 6][7, 8, 9]["a", "b", "c"]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'lstr([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
      15])=0123456789101112131415'"
ECHO: "[ INFO ] run_test(); t01 passed: 'lstr_html_B(undef)=<b>undef</b>'"
ECHO: "[ INFO ] run_test(); t02 passed: 'lstr_html_B([])='"
ECHO: "[ INFO ] run_test(); t03 passed: 'lstr_html_B([0 : 0.5 : 9])=<b>[0 : 0.5 : 9]</b>'"
ECHO: "[ INFO ] run_test(); t04 passed: 'lstr_html_B(A string)=<b>A string</b>'"
ECHO: "[ INFO ] run_test(); t05 passed: 'lstr_html_B(["orange", "apple", "grape",
      "banana"])=<b>orange</b><b>apple</b><b>grape</b><b>banana</b>'"
ECHO: "[ INFO ] run_test(); t06 passed: 'lstr_html_B(["b", "a", "n", "a", "n", "a",
      "s"])=<b>b</b><b>a</b><b>n</b><b>a</b><b>n</b><b>a</b><b>s</b>'"
ECHO: "[ INFO ] run_test(); t07 passed: 'lstr_html_B([undef])=<b>undef</b>'"
ECHO: "[ INFO ] run_test(); t08 passed: 'lstr_html_B([[1, 2], [2, 3]])=<b>[1, 2]</b><b>[2, 3]</b>'"
ECHO: "[ INFO ] run_test(); t09 passed: 'lstr_html_B(["ab", [1, 2], [2, 3], [4, 5]])=<b>ab</b><b>[1,
      2]</b><b>[2, 3]</b><b>[4, 5]</b>'"
ECHO: "[ INFO ] run_test(); t10 passed: 'lstr_html_B([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
      "c"]])=<b>[1, 2, 3]</b><b>[4, 5, 6]</b><b>[7, 8, 9]</b><b>["a", "b", "c"]</b>'"
ECHO: "[ INFO ] run_test(); t11 passed: 'lstr_html_B([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
      15])
     =<b>0</b><b>1</b><b>2</b><b>3</b><b>4</b><b>5</b><b>6</b><b>7</b><b>8</b><b>9</b><b>10</b><b>11</b><b>12</b><b>13</b><b>14</
ECHO: "[ INFO ] run_test(); t01 passed: 'consts(undef)=[]'"
ECHO: "[ INFO ] run_test(); t02 passed: 'consts([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'consts([0 : 0.5 : 9])=[]'"
```

```
ECHO: "[ INFO ] run_test(); t04 passed: 'consts(A string)=[]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'consts(["orange", "apple", "grape", "banana"])=[]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'consts(["b", "a", "n", "a", "n", "a", "s"])=[]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'consts([undef])=[]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'consts([[1, 2], [2, 3]])=[]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'consts(["ab", [1, 2], [2, 3], [4, 5]])=[]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'consts([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'consts([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'get_index(undef)=[]'"
ECHO: "[ INFO ] run_test(); t02 passed: 'get_index([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'get_index([0 : 0.5 : 9])=[]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'get_index(A string)=[0, 1, 2, 3, 4, 5, 6, 7]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'get_index(["orange", "apple", "grape", "banana"])=[0, 1, 2, 3]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'get_index(["b", "a", "n", "a", "n", "a", "s"])=[0, 1, 2, 3, 4, 5, 6]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'get_index([undef])=[0]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'get_index([[1, 2], [2, 3]])=[0, 1]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'get_index(["ab", [1, 2], [2, 3], [4, 5]])=[0, 1, 2, 3]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'get_index([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[0, 1, 2, 3]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'get_index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'pad_9(undef)=[undef, 0, 0, 0, 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t02 passed: 'pad_9([])=[0, 0, 0, 0, 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'pad_9([0 : 0.5 : 9])=[[0 : 0.5 : 9], 0, 0, 0, 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'pad_9(A string)=["A string", 0]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'pad_9(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape", "banana", 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'pad_9(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s", 0, 0]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'pad_9([undef])=[undef, 0, 0, 0, 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'pad_9([[1, 2], [2, 3]])=[[1, 2], [2, 3], 0, 0, 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'pad_9(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 3], [4, 5], 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'pad_9([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"], 0, 0, 0, 0, 0]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'pad_9([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] root(); not testing: dround()"
ECHO: "[ INFO ] root(); not testing: sround()"
ECHO: "[ INFO ] run_test(); t01 passed: 'limit_12(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'limit_12([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'limit_12([0 : 0.5 : 9])=[0 : 0.5 : 9]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'limit_12(A string)=A string'"
ECHO: "[ INFO ] run_test(); t05 passed: 'limit_12(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'limit_12(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'limit_12([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'limit_12([[1, 2], [2, 3]])=[[1, 2], [2, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'limit_12(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [1, 2], [2, 2], [2, 2]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'limit_12([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 2, 2], [2, 2, 2], [2, 2, 2], ["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'limit_12([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'sum(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'sum([])=0'"
ECHO: "[ INFO ] run_test(); t03 passed: 'sum([0 : 0.5 : 9])=85.5'"
ECHO: "[ INFO ] run_test(); t04 passed: 'sum(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'sum(["orange", "apple", "grape", "banana"])=undef'"
ECHO: "[ INFO ] run_test(); t06 passed: 'sum(["b", "a", "n", "a", "n", "a", "s"])=undef'"
ECHO: "[ INFO ] run_test(); t07 passed: 'sum([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'sum([[1, 2], [2, 3]])=[3, 5]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'sum(["ab", [1, 2], [2, 3], [4, 5]])=undef'"
ECHO: "[ INFO ] run_test(); t10 passed: 'sum([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[undef, undef, undef]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'sum([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=120'"
ECHO: "[ INFO ] run_test(); t01 passed: 'mean(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'mean([])=0'"
ECHO: "[ INFO ] run_test(); t03 passed: 'mean([0 : 0.5 : 9])=4.5'"
ECHO: "[ INFO ] run_test(); t04 passed: 'mean(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'mean(["orange", "apple", "grape", "banana"])=undef'"
ECHO: "[ INFO ] run_test(); t06 passed: 'mean(["b", "a", "n", "a", "n", "a", "s"])=undef'"
ECHO: "[ INFO ] run_test(); t07 passed: 'mean([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'mean([[1, 2], [2, 3]])=[1.5, 2.5]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'mean(["ab", [1, 2], [2, 3], [4, 5]])=undef'"
ECHO: "[ INFO ] run_test(); t10 passed: 'mean([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[undef, undef, undef]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'mean([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=7.5'"
```

```
ECHO: "[ INFO ] root(); not testing: ciselect()"
ECHO: "[ INFO ] root(); not testing: cmvselect()"
ECHO: "[ INFO ] run_test(); t01 passed: 'eselect_F(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eselect_F([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eselect_F([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'eselect_F(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eselect_F(["orange", "apple", "grape", "banana"])=["o", "a", "g", "b"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'eselect_F(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eselect_F([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eselect_F([[1, 2], [2, 3]])=[1, 2]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eselect_F(["ab", [1, 2], [2, 3], [4, 5]])=["a", 1, 2, 4]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eselect_F([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 4, 7, "a"]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'eselect_F(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'eselect_L(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eselect_L([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eselect_L([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'eselect_L(A string)=["A", " ", "s", "t", "r", "i", "n", "g"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eselect_L(["orange", "apple", "grape", "banana"])=["e", "e", "e", "a"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'eselect_L(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eselect_L([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eselect_L([[1, 2], [2, 3]])=[2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eselect_L(["ab", [1, 2], [2, 3], [4, 5]])=["b", 2, 3, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eselect_L([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[3, 6, 9, "c"]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'eselect_L(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'eselect_1(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'eselect_1([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'eselect_1([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 *skip*: 'eselect_1(A string)'"
ECHO: "[ INFO ] run_test(); t05 passed: 'eselect_1(["orange", "apple", "grape", "banana"])=["r", "p", "r", "a"]'"
ECHO: "[ INFO ] run_test(); t06 *skip*: 'eselect_1(Test list 02)'"
ECHO: "[ INFO ] run_test(); t07 passed: 'eselect_1([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'eselect_1([[1, 2], [2, 3]])=[2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'eselect_1(["ab", [1, 2], [2, 3], [4, 5]])=["b", 2, 3, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'eselect_1([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[2, 5, 8, "b"]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'eselect_1(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'smerge(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'smerge([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'smerge([0 : 0.5 : 9])=[[0 : 0.5 : 9]]'"
ECHO: "[ INFO ] run_test(); t04 passed: 'smerge(A string)=["A string"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'smerge(["orange", "apple", "grape", "banana"])=["orange", "apple", "grape", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'smerge(["b", "a", "n", "a", "n", "a", "s"])=["b", "a", "n", "a", "n", "a", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'smerge([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'smerge([[1, 2], [2, 3]])=[1, 2, 2, 3]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'smerge(["ab", [1, 2], [2, 3], [4, 5]])=["ab", 1, 2, 2, 3, 4, 5]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'smerge([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[1, 2, 3, 4, 5, 6, 7, 8, 9, "a", "b", "c"]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'smerge([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'pmerge(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'pmerge([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'pmerge([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'pmerge(A string)=["A string"]'"
ECHO: "[ INFO ] run_test(); t05 passed: 'pmerge(["orange", "apple", "grape", "banana"])=[["o", "a", "g", "b"], ["r", "p", "r", "a"], ["a", "p", "a", "n"], ["n", "l", "p", "a"], ["g", "e", "e", "n"]]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'pmerge(["b", "a", "n", "a", "n", "a", "s"])=[["b", "a", "n", "a", "n", "a", "s"]]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'pmerge([undef])=undef'"
ECHO: "[ INFO ] run_test(); t08 passed: 'pmerge([[1, 2], [2, 3]])=[[1, 2], [2, 3]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'pmerge(["ab", [1, 2], [2, 3], [4, 5]])=[["a", 1, 2, 4], ["b", 2, 3, 5]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'pmerge([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[1, 4, 7, "a"], [2, 5, 8, "b"], [3, 6, 9, "c"]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'pmerge([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=undef'"
ECHO: "[ INFO ] run_test(); t01 passed: 'qsort(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'qsort([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'qsort([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'qsort(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'qsort(["orange", "apple", "grape", "banana"])=["apple", "banana", "grape", "orange"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'qsort(["b", "a", "n", "a", "n", "a", "s"])=["a", "a", "a", "b",
```

```
    "n", "n", "s"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'qsort([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 *skip*: 'qsort(Test list 04)'"
ECHO: "[ INFO ] run_test(); t09 *skip*: 'qsort(Test list 05)'"
ECHO: "[ INFO ] run_test(); t10 *skip*: 'qsort(Test list 06)'"
ECHO: "[ INFO ] run_test(); t11 passed: 'qsort([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])=[0,
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]'"
ECHO: "[ INFO ] run_test(); t01 passed: 'qsort_1R(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'qsort_1R([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'qsort_1R([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'qsort_1R(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'qsort_1R(["orange", "apple", "grape", "banana"])=["orange",
    "grape", "apple", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 *skip*: 'qsort_1R(Test list 02)'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'qsort_1R(Test list 03)'"
ECHO: "[ INFO ] run_test(); t08 passed: 'qsort_1R([[1, 2], [2, 3]])=[[2, 3], [1, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'qsort_1R(["ab", [1, 2], [2, 3], [4, 5]])=[[4, 5], [2, 3], [1, 2],
    "ab"]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'qsort_1R([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b", "c"]])=[[7,
    8, 9], [4, 5, 6], [1, 2, 3], ["a", "b", "c"]]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'qsort_1R(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'qsort2_1R(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'qsort2_1R([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'qsort2_1R([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'qsort2_1R(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'qsort2_1R(["orange", "apple", "grape", "banana"])=["orange",
    "grape", "apple", "banana"]'"
ECHO: "[ INFO ] run_test(); t06 *skip*: 'qsort2_1R(Test list 02)'"
ECHO: "[ INFO ] run_test(); t07 *skip*: 'qsort2_1R(Test list 03)'"
ECHO: "[ INFO ] run_test(); t08 passed: 'qsort2_1R([[1, 2], [2, 3]])=[[2, 3], [1, 2]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'qsort2_1R(["ab", [1, 2], [2, 3], [4, 5]])=["ab", [4, 5], [2, 3],
    [1, 2]]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'qsort2_1R([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=[["a", "b", "c"], [7, 8, 9], [4, 5, 6], [1, 2, 3]]'"
ECHO: "[ INFO ] run_test(); t11 *skip*: 'qsort2_1R(Vector of integers 0 to 15)'"
ECHO: "[ INFO ] run_test(); t01 passed: 'qsort2_HR(undef)=undef'"
ECHO: "[ INFO ] run_test(); t02 passed: 'qsort2_HR([])=[]'"
ECHO: "[ INFO ] run_test(); t03 passed: 'qsort2_HR([0 : 0.5 : 9])=undef'"
ECHO: "[ INFO ] run_test(); t04 passed: 'qsort2_HR(A string)=undef'"
ECHO: "[ INFO ] run_test(); t05 passed: 'qsort2_HR(["orange", "apple", "grape", "banana"])=["orange",
    "grape", "banana", "apple"]'"
ECHO: "[ INFO ] run_test(); t06 passed: 'qsort2_HR(["b", "a", "n", "a", "n", "a", "s"])=["s", "n", "n",
    "b", "a", "a", "a"]'"
ECHO: "[ INFO ] run_test(); t07 passed: 'qsort2_HR([undef])=[undef]'"
ECHO: "[ INFO ] run_test(); t08 passed: 'qsort2_HR([[1, 2], [2, 3]])=[[3, 2], [2, 1]]'"
ECHO: "[ INFO ] run_test(); t09 passed: 'qsort2_HR(["ab", [1, 2], [2, 3], [4, 5]])=[[5, 4], [3, 2], [2, 1],
    "ab"]'"
ECHO: "[ INFO ] run_test(); t10 passed: 'qsort2_HR([[1, 2, 3], [4, 5, 6], [7, 8, 9], ["a", "b",
    "c"]])=[["c", "b", "a"], [9, 8, 7], [6, 5, 4], [3, 2, 1]]'"
ECHO: "[ INFO ] run_test(); t11 passed: 'qsort2_HR([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15])=[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]'"
```

3.2 Math

- Vector Algebra
- Bitwise

3.2.1 Vector Algebra

- Script
- Results

3.2.1.1 Script

```
include <math.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
```

```
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["fac", "Function argument count",    undef],
  ["crp", "Result precision",           undef],
  ["t01", "All undefined",              [undef,undef,undef,undef,undef,undef]],
  ["t02", "All empty lists",            [empty_lst,empty_lst,
  empty_lst,empty_lst,empty_lst,empty_lst]],
  ["t03", "All scalars",                [60, 50, 40, 30, 20, 10]],
  ["t04", "All 1d vectors",             [[99], [58], [12], [42], [15], [1]]],
  ["t05", "All 2d vectors",             [
                                          [99,2], [58,16], [12,43],
                                          [42,13], [15,59], [1,85]
                                        ]],
  ["t06", "All 3d vectors",             [
                                          [199,20,55], [158,116,75], [12,43,90],
                                          [42,13,34], [15,59,45], [62,33,69]
                                        ]],
  ["t07", "All 4d vectors",             [
                                          [169,27,35,10], [178,016,25,20], [12,43,90,30],
                                          [42,13,34,60], [15,059,45,50], [62,33,69,40]
                                        ]],
  ["t08", "Orthogonal vectors",         [
                                          +x_axis3d_uv, +
  y_axis3d_uv, +z_axis3d_uv,
                                          -x_axis3d_uv, -
  y_axis3d_uv, -z_axis3d_uv,
                                        ]],
  ["t09", "Coplanar vectors",           [
                                          +x_axis3d_uv, +
  y_axis3d_uv, [2,2,0],
                                          origin3d, origin3d,
  origin3d,
                                        ]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 'skip' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["distance_pp",
    2,                                              // fac
    4,                                              // crp
    undef,                                          // t01
    undef,                                          // t02
    10,                                             // t03
    41,                                             // t04
    43.3244,                                        // t05
    106.2873,                                       // t06
    20.0499,                                        // t07
    1.4142,                                         // t08
    1.4142                                          // t09
  ],
  ["is_left_ppp",
    3,                                              // fac
    4,                                              // crp
    undef,                                          // t01
    undef,                                          // t02
    undef,                                          // t03
    undef,                                          // t04
    -463,                                           // t05
    17009,                                          // t06
    -1583,                                          // t07
```

```
    1,                                          // t08
    -3                                          // t09
  ],
  ["dimension_2to3_v",
    1,                                          // fac
    4,                                          // crp
    [undef,undef,0],                            // t01
    [undef,undef,0],                            // t02
    [undef,undef,0],                            // t03
    [99,undef,0],                               // t04
    [99,2,0],                                   // t05
    [199,20,55],                                // t06
    [169,27,0],                                 // t07
    x_axis3d_uv,                                // t08
    x_axis3d_uv                                 // t09
  ],
  ["get_line_dim",
    2,                                          // fac
    4,                                          // crp
    2,                                          // t01
    0,                                          // t02
    2,                                          // t03
    1,                                          // t04
    2,                                          // t05
    3,                                          // t06
    4,                                          // t07
    3,                                          // t08
    3                                           // t09
  ],
  ["get_line_tp",
    2,                                          // fac
    4,                                          // crp
    [undef,undef],                              // t01
    empty_lst,                                  // t02
    [60,50],                                    // t03
    [58],                                       // t04
    [58,16],                                    // t05
    [158,116,75],                               // t06
    [178,16,25,20],                             // t07
    y_axis3d_uv,                                // t08
    y_axis3d_uv                                 // t09
  ],
  ["get_line_ip",
    2,                                          // fac
    4,                                          // crp
    origin2d,                                   // t01
    empty_lst,                                  // t02
    origin2d,                                   // t03
    [99],                                       // t04
    [99,2],                                     // t05
    [199,20,55],                                // t06
    [169,27,35,10],                             // t07
    x_axis3d_uv,                                // t08
    x_axis3d_uv                                 // t09
  ],
  ["get_line2origin",
    2,                                          // fac
    4,                                          // crp
    [undef, undef],                             // t01
    empty_lst,                                  // t02
    [60,50],                                    // t03
    [-41],                                      // t04
    [-41,14],                                   // t05
    [-41,96,20],                                // t06
    [9,-11,-10,10],                             // t07
    [-1,1,0],                                   // t08
    [-1,1,0]                                    // t09
  ],
  ["dot_ll",
    4,                                          // fac
    4,                                          // crp
    undef,                                      // t01
    undef,                                      // t02
    3900,                                       // t03
    -1230,                                      // t04
    -1650,                                      // t05
    -5230,                                      // t06
    1460,                                       // t07
    1,                                          // t08
    0                                           // t09
  ],
```

```
["cross_ll",
  4,                                          // fac
  4,                                          // crp
  skip,                                       // t01
  skip,                                       // t02
  skip,                                       // t03
  skip,                                       // t04
  810,                                        // t05
  [-4776,-1696,-1650],                        // t06
  skip,                                       // t07
  [-1,-1,1],                                  // t08
  [0,0,4]                                     // t09
],
["striple_lll",
  6,                                          // fac
  4,                                          // crp
  skip,                                       // t01
  skip,                                       // t02
  skip,                                       // t03
  skip,                                       // t04
  [-14760,5040],                              // t05
  -219976,                                    // t06
  skip,                                       // t07
  -2,                                         // t08
  0                                           // t09
],
["angle_ll",
  4,                                          // fac
  4,                                          // crp
  undef,                                      // t01
  undef,                                      // t02
  -2.9357,                                    // t03
  undef,                                      // t04
  153.8532,                                   // t05
  134.4573,                                   // t06
  undef,                                      // t07
  60,                                         // t08
  90                                          // t09
],
["angle_lll",
  6,                                          // fac
  4,                                          // crp
  skip,                                       // t01
  skip,                                       // t02
  skip,                                       // t03
  skip,                                       // t04
  skip,                                       // t05
  -91.362,                                    // t06
  skip,                                       // t07
  -63.4349,                                   // t08
  0                                           // t09
],
["unit_l",
  2,                                          // fac
  4,                                          // crp
  undef,                                      // t01
  undef,                                      // t02
  [.7682,0.6402],                             // t03
  [-1],                                       // t04
  [-0.9464,0.3231],                           // t05
  [-0.3857,0.9032,0.1882],                    // t06
  [0.44888,-0.5486,-0.4988,0.4988],           // t07
  [-0.7071,0.7071,0],                         // t08
  [-0.7071,0.7071,0]                          // t09
],
["are_coplanar_lll",
  6,                                          // fac
  4,                                          // crp
  skip,                                       // t01
  skip,                                       // t02
  skip,                                       // t03
  skip,                                       // t04
  skip,                                       // t05
  false,                                      // t06
  skip,                                       // t07
  false,                                      // t08
  true                                        // t09
],
["get_pnorm2nv",
  2,                                          // fac
  4,                                          // crp
```

```
    skip,                                                   // t01
    skip,                                                   // t02
    [60,50,0],                                              // t03
    skip,                                                   // t04
    [0,0,1468],                                             // t05
    [-4880,-6235,19924],                                    // t06
    skip,                                                   // t07
    z_axis3d_uv,                                            // t08
    z_axis3d_uv                                             // t09
  ]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
function gv( vid, e ) = get_value( vid )[e];
module run( fname, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");

  if ( get_table_v(good_r, good_c, fname, vid) != skip )
    children();
  else if ( show_skipped )
    log_info( str("*skip*: ", vid, " '", fname, "(", value_text, ")'") );
}
module test( fname, fresult, vid, pair )
{
  value_text = get_table_v(test_r, test_c, vid, "td");
  fname_argc = get_table_v(good_r, good_c, fname, "fac");
  comp_prcsn = get_table_v(good_r, good_c, fname, "crp");
  pass_value = get_table_v(good_r, good_c, fname, vid);

  test_pass = validate(cv=fresult, t="almost", ev=pass_value, p=comp_prcsn, pf=true);
  farg_text = (pair == true)
            ? lstr(eappend(", ", nssequence(rselect(get_value(vid), [0:
  fname_argc-1]), n=2, s=2), r=false, j=false, l=false))
            : lstr(eappend(", ", rselect(get_value(vid), [0:fname_argc-1]), r=false,
  j=false, l=false));
  test_text = validate(str(fname, "(", farg_text, ")=~", pass_value), fresult, "almost",
  pass_value, comp_prcsn);

  if ( pass_value != skip )
  {
    if ( !test_pass )
      log_warn( str(vid, "(", value_text, ") ", test_text) );
    else if ( show_passing )
      log_info( str(vid, " ", test_text) );
  }
  else if ( show_skipped )
    log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
}

// Indirect function calls would be very useful here!!!
run_ids = delete( test_ids, mv=["fac", "crp"] );

// group 1: point
for (vid=run_ids) run("distance_pp",vid) test( "distance_pp", distance_pp(gv(vid,0),gv(vid,1
  )), vid, false );
for (vid=run_ids) run("is_left_ppp",vid) test( "is_left_ppp", is_left_ppp(gv(vid,0),gv(vid,1
  ),gv(vid,2)), vid, false );

// group 2: vector
for (vid=run_ids) run("dimension_2to3_v",vid) test( "dimension_2to3_v",
  dimension_2to3_v(gv(vid,0)), vid, false );

// group 3: line (or vector)
for (vid=run_ids) run("get_line_dim",vid) test( "get_line_dim", get_line_dim([gv(vid,0),gv(
  vid,1)]), vid, true );
for (vid=run_ids) run("get_line_tp",vid) test( "get_line_tp", get_line_tp([gv(vid,0),gv(vid,
  1)]), vid, true );
for (vid=run_ids) run("get_line_ip",vid) test( "get_line_ip", get_line_ip([gv(vid,0),gv(vid,
  1)]), vid, true );
for (vid=run_ids) run("get_line2origin",vid) test( "get_line2origin", get_line2origin([
  gv(vid,0),gv(vid,1)]), vid, true );
for (vid=run_ids) run("dot_ll",vid) test( "dot_ll", dot_ll([gv(vid,0),gv(vid,1)],[gv(vid,2),gv(
  vid,3)]), vid, true );
for (vid=run_ids) run("cross_ll",vid) test( "cross_ll", cross_ll([gv(vid,0),gv(vid,1)],[gv(vid,
  2),gv(vid,3)]), vid, true );
```

```
for (vid=run_ids) run("striple_lll",vid) test( "striple_lll", striple_lll([gv(vid,0),gv(vid,1)],[gv(vid,2),gv(vid,3)],[gv(vid,4),gv(vid,5)]), vid, true );
for (vid=run_ids) run("angle_ll",vid) test( "angle_ll", angle_ll([gv(vid,0),gv(vid,1)],[gv(vid,2),gv(vid,3)]), vid, true );
for (vid=run_ids) run("angle_lll",vid) test( "angle_lll", angle_lll([gv(vid,0),gv(vid,1)],[gv(vid,2),gv(vid,3)],[gv(vid,4),gv(vid,5)]), vid, true );
for (vid=run_ids) run("unit_l",vid) test( "unit_l", unit_l([gv(vid,0),gv(vid,1)]), vid, true );
for (vid=run_ids) run("are_coplanar_lll",vid) test( "are_coplanar_lll", are_coplanar_lll([gv(vid,0),gv(vid,1)],[gv(vid,2),gv(vid,3)],[gv(vid,4),gv(vid,5)]), vid, true );

// group 4: plane and pnorm
for (vid=run_ids) run("get_pnorm2nv",vid) test( "get_pnorm2nv", get_pnorm2nv([gv(vid,0),gv(vid,1)]), vid, true );

// end-of-tests
```

3.2.1.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run(): test(); t01 passed: 'distance_pp(undef, undef)=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'distance_pp([], [])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'distance_pp(60, 50)=~10'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'distance_pp([99], [58])=~41'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'distance_pp([99, 2], [58, 16])=~43.3244'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'distance_pp([199, 20, 55], [158, 116, 75])=~106.287'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'distance_pp([169, 27, 35, 10], [178, 16, 25, 20])=~20.0499'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'distance_pp([1, 0, 0], [0, 1, 0])=~1.4142'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'distance_pp([1, 0, 0], [0, 1, 0])=~1.4142'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'is_left_ppp(undef, undef, undef)=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'is_left_ppp([], [], [])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'is_left_ppp(60, 50, 40)=~undef'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'is_left_ppp([99], [58], [12])=~undef'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'is_left_ppp([99, 2], [58, 16], [12, 43])=~-463'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'is_left_ppp([199, 20, 55], [158, 116, 75], [12, 43, 90])=~17009'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'is_left_ppp([169, 27, 35, 10], [178, 16, 25, 20], [12, 43, 90, 30])=~-1583'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'is_left_ppp([1, 0, 0], [0, 1, 0], [0, 0, 1])=~1'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'is_left_ppp([1, 0, 0], [0, 1, 0], [2, 2, 0])=~-3'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'dimension_2to3_v(undef)=~[undef, undef, 0]'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'dimension_2to3_v([])=~[undef, undef, 0]'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'dimension_2to3_v(60)=~[undef, undef, 0]'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'dimension_2to3_v([99])=~[99, undef, 0]'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'dimension_2to3_v([99, 2])=~[99, 2, 0]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'dimension_2to3_v([199, 20, 55])=~[199, 20, 55]'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'dimension_2to3_v([169, 27, 35, 10])=~[169, 27, 0]'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'dimension_2to3_v([1, 0, 0])=~[1, 0, 0]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'dimension_2to3_v([1, 0, 0])=~[1, 0, 0]'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'get_line_dim([undef, undef])=~2'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'get_line_dim([[], []])=~0'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'get_line_dim([60, 50])=~2'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'get_line_dim([[99], [58]])=~1'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'get_line_dim([[99, 2], [58, 16]])=~2'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'get_line_dim([[199, 20, 55], [158, 116, 75]])=~3'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'get_line_dim([[169, 27, 35, 10], [178, 16, 25, 20]])=~4'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'get_line_dim([[1, 0, 0], [0, 1, 0]])=~3'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'get_line_dim([[1, 0, 0], [0, 1, 0]])=~3'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'get_line_tp([undef, undef])=~[undef, undef]'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'get_line_tp([[], []])=~[]'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'get_line_tp([60, 50])=~[60, 50]'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'get_line_tp([[99], [58]])=~[58]'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'get_line_tp([[99, 2], [58, 16]])=~[58, 16]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'get_line_tp([[199, 20, 55], [158, 116, 75]])=~[158, 116, 75]'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'get_line_tp([[169, 27, 35, 10], [178, 16, 25, 20]])=~[178, 16, 25, 20]'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'get_line_tp([[1, 0, 0], [0, 1, 0]])=~[0, 1, 0]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'get_line_tp([[1, 0, 0], [0, 1, 0]])=~[0, 1, 0]'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'get_line_ip([undef, undef])=~[0, 0]'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'get_line_ip([[], []])=~[]'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'get_line_ip([60, 50])=~[0, 0]'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'get_line_ip([[99], [58]])=~[99]'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'get_line_ip([[99, 2], [58, 16]])=~[99, 2]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'get_line_ip([[199, 20, 55], [158, 116, 75]])=~[199, 20, 55]'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'get_line_ip([[169, 27, 35, 10], [178, 16, 25, 20]])=~[169, 27, 35, 10]'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'get_line_ip([[1, 0, 0], [0, 1, 0]])=~[1, 0, 0]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'get_line_ip([[1, 0, 0], [0, 1, 0]])=~[1, 0, 0]'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'get_line2origin([undef, undef])=~[undef, undef]'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'get_line2origin([[], []])=~[]'"
```

```
ECHO: "[ INFO ] run(): test(); t03 passed: 'get_line2origin([60, 50])=~[60, 50]'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'get_line2origin([[99], [58]])=~[-41]'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'get_line2origin([[99, 2], [58, 16]])=~[-41, 14]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'get_line2origin([[199, 20, 55], [158, 116, 75]])=~[-41, 96, 20]'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'get_line2origin([[169, 27, 35, 10], [178, 16, 25, 20]])=~[9, -11, -10, 10]'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'get_line2origin([[1, 0, 0], [0, 1, 0]])=~[-1, 1, 0]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'get_line2origin([[1, 0, 0], [0, 1, 0]])=~[-1, 1, 0]'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'dot_ll([undef, undef], [undef, undef])=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'dot_ll([[], []], [[], []])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'dot_ll([60, 50], [40, 30])=~3900'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'dot_ll([[99], [58]], [[12], [42]])=~-1230'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'dot_ll([[99, 2], [58, 16]], [[12, 43], [42, 13]])=~-1650'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'dot_ll([[199, 20, 55], [158, 116, 75]], [[12, 43, 90], [42, 13, 34]])=~-5230'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'dot_ll([[169, 27, 35, 10], [178, 16, 25, 20]], [[12, 43, 90, 30], [42, 13, 34, 60]])=~1460'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'dot_ll([[1, 0, 0], [0, 1, 0]], [[0, 0, 1], [-1, 0, 0]])=~1'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'dot_ll([[1, 0, 0], [0, 1, 0]], [[2, 2, 0], [0, 0, 0]])=~0'"
ECHO: "[ INFO ] run(); *skip*: t01 'cross_ll(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'cross_ll(All empty lists)'"
ECHO: "[ INFO ] run(); *skip*: t03 'cross_ll(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'cross_ll(All 1d vectors)'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'cross_ll([[99, 2], [58, 16]], [[12, 43], [42, 13]])=~810'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'cross_ll([[199, 20, 55], [158, 116, 75]], [[12, 43, 90], [42, 13, 34]])=~[-4776, -1696, -1650]'"
ECHO: "[ INFO ] run(); *skip*: t07 'cross_ll(All 4d vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'cross_ll([[1, 0, 0], [0, 1, 0]], [[0, 0, 1], [-1, 0, 0]])=~[-1, -1, 1]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'cross_ll([[1, 0, 0], [0, 1, 0]], [[2, 2, 0], [0, 0, 0]])=~[0, 0, 4]'"
ECHO: "[ INFO ] run(); *skip*: t01 'striple_lll(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'striple_lll(All empty lists)'"
ECHO: "[ INFO ] run(); *skip*: t03 'striple_lll(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'striple_lll(All 1d vectors)'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'striple_lll([[99, 2], [58, 16]], [[12, 43], [42, 13]], [[15, 59], [1, 85]])=~[-14760, 5040]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'striple_lll([[199, 20, 55], [158, 116, 75]], [[12, 43, 90], [42, 13, 34]], [[15, 59, 45], [62, 33, 69]])=~-219976'"
ECHO: "[ INFO ] run(); *skip*: t07 'striple_lll(All 4d vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'striple_lll([[1, 0, 0], [0, 1, 0]], [[0, 0, 1], [-1, 0, 0]], [[0, -1, 0], [0, 0, -1]])=~-2'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'striple_lll([[1, 0, 0], [0, 1, 0]], [[2, 2, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0]])=~0'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'angle_ll([undef, undef], [undef, undef])=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'angle_ll([[], []], [[], []])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'angle_ll([60, 50], [40, 30])=~-2.9357'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'angle_ll([[99], [58]], [[12], [42]])=~undef'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'angle_ll([[99, 2], [58, 16]], [[12, 43], [42, 13]])=~153.853'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'angle_ll([[199, 20, 55], [158, 116, 75]], [[12, 43, 90], [42, 13, 34]])=~134.457'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'angle_ll([[169, 27, 35, 10], [178, 16, 25, 20]], [[12, 43, 90, 30], [42, 13, 34, 60]])=~undef'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'angle_ll([[1, 0, 0], [0, 1, 0]], [[0, 0, 1], [-1, 0, 0]])=~60'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'angle_ll([[1, 0, 0], [0, 1, 0]], [[2, 2, 0], [0, 0, 0]])=~90'"
ECHO: "[ INFO ] run(); *skip*: t01 'angle_lll(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'angle_lll(All empty lists)'"
ECHO: "[ INFO ] run(); *skip*: t03 'angle_lll(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'angle_lll(All 1d vectors)'"
ECHO: "[ INFO ] run(); *skip*: t05 'angle_lll(All 2d vectors)'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'angle_lll([[199, 20, 55], [158, 116, 75]], [[12, 43, 90], [42, 13, 34]], [[15, 59, 45], [62, 33, 69]])=~-91.362'"
ECHO: "[ INFO ] run(); *skip*: t07 'angle_lll(All 4d vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'angle_lll([[1, 0, 0], [0, 1, 0]], [[0, 0, 1], [-1, 0, 0]], [[0, -1, 0], [0, 0, -1]])=~-63.4349'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'angle_lll([[1, 0, 0], [0, 1, 0]], [[2, 2, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0]])=~0'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'unit_l([undef, undef])=~undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'unit_l([[], []])=~undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'unit_l([60, 50])=~[0.7682, 0.6402]'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'unit_l([[99], [58]])=~[-1]'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'unit_l([[99, 2], [58, 16]])=~[-0.9464, 0.3231]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'unit_l([[199, 20, 55], [158, 116, 75]])=~[-0.3857, 0.9032, 0.1882]'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'unit_l([[169, 27, 35, 10], [178, 16, 25, 20]])=~[0.44888, -0.5486, -0.4988, 0.4988]'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'unit_l([[1, 0, 0], [0, 1, 0]])=~[-0.7071, 0.7071, 0]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'unit_l([[1, 0, 0], [0, 1, 0]])=~[-0.7071, 0.7071, 0]'"
ECHO: "[ INFO ] run(); *skip*: t01 'are_coplanar_lll(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'are_coplanar_lll(All empty lists)'"
```

```
ECHO: "[ INFO ] run(); *skip*: t03 'are_coplanar_lll(All scalars)'"
ECHO: "[ INFO ] run(); *skip*: t04 'are_coplanar_lll(All 1d vectors)'"
ECHO: "[ INFO ] run(); *skip*: t05 'are_coplanar_lll(All 2d vectors)'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'are_coplanar_lll([[199, 20, 55], [158, 116, 75]], [[12, 43,
    90], [42, 13, 34]], [[15, 59, 45], [62, 33, 69]])=~false'"
ECHO: "[ INFO ] run(); *skip*: t07 'are_coplanar_lll(All 4d vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'are_coplanar_lll([[1, 0, 0], [0, 1, 0]], [[0, 0, 1], [-1, 0,
    0]], [[0, -1, 0], [0, 0, -1]])=~false'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'are_coplanar_lll([[1, 0, 0], [0, 1, 0]], [[2, 2, 0], [0, 0,
    0]], [[0, 0, 0], [0, 0, 0]])=~true'"
ECHO: "[ INFO ] run(); *skip*: t01 'get_pnorm2nv(All undefined)'"
ECHO: "[ INFO ] run(); *skip*: t02 'get_pnorm2nv(All empty lists)'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'get_pnorm2nv([60, 50])=~[60, 50, 0]'"
ECHO: "[ INFO ] run(); *skip*: t04 'get_pnorm2nv(All 1d vectors)'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'get_pnorm2nv([[99, 2], [58, 16]])=~[0, 0, 1468]'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'get_pnorm2nv([[199, 20, 55], [158, 116, 75]])=~[-4880, -6235,
    19924]'"
ECHO: "[ INFO ] run(); *skip*: t07 'get_pnorm2nv(All 4d vectors)'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'get_pnorm2nv([[1, 0, 0], [0, 1, 0]])=~[0, 0, 1]'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'get_pnorm2nv([[1, 0, 0], [0, 1, 0]])=~[0, 0, 1]'"
```

3.2.2 Bitwise

- Script
- Results

3.2.2.1 Script

```
include <math/math_bitwise.scad>;
use <datatypes/datatypes_table.scad>;
use <console.scad>;
use <validation.scad>;

show_passing = true;    // show passing tests
show_skipped = true;    // show skipped tests

echo( str("OpenSCAD Version ", version()) );

// test-values columns
test_c =
[
  ["id", "identifier"],
  ["td", "description"],
  ["tv", "test value"]
];

// test-values rows
test_r =
[
  ["fac", "Function argument count",    undef],
  ["t01", "All undefined",              [undef,undef]],
  ["t02", "All empty lists",            [empty_lst,empty_lst]],
  ["t03", "test value 1",               [254, 0]],
  ["t04", "test value 2",               [254, 1]],
  ["t05", "test value 3",               [255, 0]],
  ["t06", "test value 4",               [0, 255]],
  ["t07", "test value 5",               [126, 63]],
  ["t08", "test value 6",               [25, 10]],
  ["t09", "test value 7",               [1024, 512]],
  ["t10", "test value 8",               [4253, 315]],
  ["t11", "test value 9",               [835, 769]],
  ["t12", "test value 10",              [856, 625]]
];

test_ids = get_table_ridl( test_r );

// expected columns: ("id" + one column for each test)
good_c = pmerge([concat("id", test_ids), concat("identifier", test_ids)]);

// expected rows: ("golden" test results), use 'skip' to skip test
skip = -1;  // skip test

good_r =
[ // function
  ["bitwise_is_equal_0", 2,
```

```
    false,                          // t01
    false,                          // t02
    true,                           // t03
    false,                          // t04
    false,                          // t05
    true,                           // t06
    true,                           // t07
    true,                           // t08
    true,                           // t09
    true,                           // t10
    true,                           // t11
    true                            // t12
  ],
  ["bitwise_is_equal_1", 2,
    false,                          // t01
    false,                          // t02
    false,                          // t03
    true,                           // t04
    true,                           // t05
    false,                          // t06
    false,                          // t07
    false,                          // t08
    false,                          // t09
    false,                          // t10
    false,                          // t11
    false                           // t12
  ],
  ["bitwise_i2v", 1,
    undef,                          // t01
    undef,                          // t02
    [1,1,1,1,1,1,1,0],              // t03
    [1,1,1,1,1,1,1,0],              // t04
    [1,1,1,1,1,1,1,1],              // t05
    [0],                            // t06
    [1,1,1,1,1,1,0],                // t07
    [1,1,0,0,1],                    // t08
    [1,0,0,0,0,0,0,0,0,0,0],        // t09
    [1,0,0,0,0,1,0,0,1,1,1,0,1],    // t10
    [1,1,0,1,0,0,0,0,1,1],          // t11
    [1,1,0,1,0,1,1,0,0,0]           // t12
  ],
  ["bitwise_i2v_v2i", 1,
    undef,                          // t01
    undef,                          // t02
    254,                            // t03
    254,                            // t04
    255,                            // t05
    0,                              // t06
    126,                            // t07
    25,                             // t08
    1024,                           // t09
    4253,                           // t10
    835,                            // t11
    856                             // t12
  ],
  ["bitwise_i2s", 1,
    undef,                          // t01
    undef,                          // t02
    "11111110",                     // t03
    "11111110",                     // t04
    "11111111",                     // t05
    "0",                            // t06
    "1111110",                      // t07
    "11001",                        // t08
    "10000000000",                  // t09
    "1000010011101",                // t10
    "1101000011",                   // t11
    "1101011000"                    // t12
  ],
  ["bitwise_i2s_s2i", 1,
    undef,                          // t01
    undef,                          // t02
    254,                            // t03
    254,                            // t04
    255,                            // t05
    0,                              // t06
    126,                            // t07
    25,                             // t08
    1024,                           // t09
    4253,                           // t10
    835,                            // t11
```

```
  856                             // t12
],
["bitwise_imi_32", 1,
  undef,                          // t01
  undef,                          // t02
  7,                              // t03
  7,                              // t04
  7,                              // t05
  0,                              // t06
  7,                              // t07
  6,                              // t08
  0,                              // t09
  7,                              // t10
  0,                              // t11
  6                               // t12
],
["bitwise_and", 2,
  undef,                          // t01
  undef,                          // t02
  0,                              // t03
  0,                              // t04
  0,                              // t05
  0,                              // t06
  62,                             // t07
  8,                              // t08
  0,                              // t09
  25,                             // t10
  769,                            // t11
  592                             // t12
],
["bitwise_or", 2,
  undef,                          // t01
  undef,                          // t02
  254,                            // t03
  255,                            // t04
  255,                            // t05
  255,                            // t06
  127,                            // t07
  27,                             // t08
  1536,                           // t09
  4543,                           // t10
  835,                            // t11
  889                             // t12
],
["bitwise_xor", 2,
  undef,                          // t01
  undef,                          // t02
  254,                            // t03
  255,                            // t04
  255,                            // t05
  255,                            // t06
  65,                             // t07
  19,                             // t08
  1536,                           // t09
  4518,                           // t10
  66,                             // t11
  297                             // t12
],
["bitwise_not", 1,
  undef,                          // t01
  undef,                          // t02
  1,                              // t03
  1,                              // t04
  0,                              // t05
  1,                              // t06
  1,                              // t07
  6,                              // t08
  1023,                           // t09
  3938,                           // t10
  188,                            // t11
  167                             // t12
],
["bitwise_lsh", 1,
  undef,                          // t01
  undef,                          // t02
  508,                            // t03
  508,                            // t04
  510,                            // t05
  0,                              // t06
  252,                            // t07
  50,                             // t08
```

```
    2048,                           // t09
    8506,                           // t10
    1670,                           // t11
    1712                            // t12
  ],
  ["bitwise_rsh", 1,
    undef,                          // t01
    undef,                          // t02
    127,                            // t03
    127,                            // t04
    127,                            // t05
    0,                              // t06
    63,                             // t07
    12,                             // t08
    512,                            // t09
    2126,                           // t10
    417,                            // t11
    428                             // t12
  ]
];

// sanity-test tables
table_check( test_r, test_c, false );
table_check( good_r, good_c, false );

// validate helper function and module
function get_value( vid ) = get_table_v(test_r, test_c, vid, "tv");
function gv( vid, e ) = get_value( vid )[e];
module run( fname, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");

  if ( get_table_v(good_r, good_c, fname, vid) != skip )
    children();
  else if ( show_skipped )
    log_info( str("*skip*: ", vid, " '", fname, "(", value_text, ")'") );
}
module test( fname, fresult, vid )
{
  value_text = get_table_v(test_r, test_c, vid, "td");
  fname_argc = get_table_v(good_r, good_c, fname, "fac");
  pass_value = get_table_v(good_r, good_c, fname, vid);

  test_pass = validate(cv=fresult, t="equals", ev=pass_value, pf=true);
  farg_text = lstr(eappend(", ", rselect(get_value(vid), [0:fname_argc-1]), r=false,
  j=false, l=false));
  test_text = validate(str(fname, "(", farg_text, ")=", pass_value), fresult, "equals",
  pass_value);

  if ( pass_value != skip )
  {
    if ( !test_pass )
      log_warn( str(vid, "(", value_text, ") ", test_text) );
    else if ( show_passing )
      log_info( str(vid, " ", test_text) );
  }
  else if ( show_skipped )
    log_info( str(vid, " *skip*: '", fname, "(", value_text, ")'") );
}

// Indirect function calls would be very useful here!!!
run_ids = delete( test_ids, mv=["fac", "crp"] );
for (vid=run_ids) run("bitwise_is_equal_0",vid) test( "bitwise_is_equal_0",
  bitwise_is_equal(gv(vid,0),gv(vid,1),0), vid );
for (vid=run_ids) run("bitwise_is_equal_1",vid) test( "bitwise_is_equal_1",
  bitwise_is_equal(gv(vid,0),gv(vid,1),1), vid );
for (vid=run_ids) run("bitwise_i2v",vid) test( "bitwise_i2v", bitwise_i2v(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_i2v_v2i",vid) test( "bitwise_i2v_v2i",
  bitwise_v2i(bitwise_i2v(gv(vid,0))), vid );
for (vid=run_ids) run("bitwise_i2s",vid) test( "bitwise_i2s", bitwise_i2s(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_i2s_s2i",vid) test( "bitwise_i2s_s2i",
  bitwise_s2i(bitwise_i2s(gv(vid,0))), vid );
for (vid=run_ids) run("bitwise_imi_32",vid) test( "bitwise_imi_32",
  bitwise_imi(gv(vid,0),3,2), vid );
for (vid=run_ids) run("bitwise_and",vid) test( "bitwise_and", bitwise_and(gv(vid,0),gv(vid,1
  )), vid );
for (vid=run_ids) run("bitwise_or",vid) test( "bitwise_or", bitwise_or(gv(vid,0),gv(vid,1)),
  vid );
for (vid=run_ids) run("bitwise_xor",vid) test( "bitwise_xor", bitwise_xor(gv(vid,0),gv(vid,1
  )), vid );
for (vid=run_ids) run("bitwise_not",vid) test( "bitwise_not", bitwise_not(gv(vid,0)), vid );
```

```
for (vid=run_ids) run("bitwise_lsh",vid) test( "bitwise_lsh", bitwise_lsh(gv(vid,0)), vid );
for (vid=run_ids) run("bitwise_rsh",vid) test( "bitwise_rsh", bitwise_rsh(gv(vid,0)), vid );

// end-of-tests
```

3.2.2.2 Results

```
ECHO: "OpenSCAD Version [2017, 2, 19]"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_is_equal_0(undef, undef)=false’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_is_equal_0([], [])=false’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_is_equal_0(254, 0)=true’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_is_equal_0(254, 1)=false’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_is_equal_0(255, 0)=false’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_is_equal_0(0, 255)=true’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_is_equal_0(126, 63)=true’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_is_equal_0(25, 10)=true’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_is_equal_0(1024, 512)=true’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_is_equal_0(4253, 315)=true’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_is_equal_0(835, 769)=true’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_is_equal_0(856, 625)=true’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_is_equal_1(undef, undef)=false’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_is_equal_1([], [])=false’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_is_equal_1(254, 0)=false’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_is_equal_1(254, 1)=true’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_is_equal_1(255, 0)=true’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_is_equal_1(0, 255)=false’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_is_equal_1(126, 63)=false’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_is_equal_1(25, 10)=false’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_is_equal_1(1024, 512)=false’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_is_equal_1(4253, 315)=false’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_is_equal_1(835, 769)=false’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_is_equal_1(856, 625)=false’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2v(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2v([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2v(254)=[1, 1, 1, 1, 1, 1, 1, 0]’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2v(254)=[1, 1, 1, 1, 1, 1, 1, 0]’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2v(255)=[1, 1, 1, 1, 1, 1, 1, 1]’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2v(0)=[0]’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2v(126)=[1, 1, 1, 1, 1, 1, 0]’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2v(25)=[1, 1, 0, 0, 1]’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2v(1024)=[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2v(4253)=[1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1]’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2v(835)=[1, 1, 0, 1, 0, 0, 0, 0, 1, 1]’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2v(856)=[1, 1, 0, 1, 0, 1, 1, 0, 0, 0]’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2v_v2i(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2v_v2i([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2v_v2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2v_v2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2v_v2i(255)=255’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2v_v2i(0)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2v_v2i(126)=126’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2v_v2i(25)=25’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2v_v2i(1024)=1024’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2v_v2i(4253)=4253’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2v_v2i(835)=835’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2v_v2i(856)=856’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2s(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2s([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2s(254)=11111110’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2s(254)=11111110’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2s(255)=11111111’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2s(0)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2s(126)=1111110’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2s(25)=11001’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2s(1024)=10000000000’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2s(4253)=1000010011101’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2s(835)=1101000011’"
ECHO: "[ INFO ] run(): test(); t12 passed: ’bitwise_i2s(856)=1101011000’"
ECHO: "[ INFO ] run(): test(); t01 passed: ’bitwise_i2s_s2i(undef)=undef’"
ECHO: "[ INFO ] run(): test(); t02 passed: ’bitwise_i2s_s2i([])=undef’"
ECHO: "[ INFO ] run(): test(); t03 passed: ’bitwise_i2s_s2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t04 passed: ’bitwise_i2s_s2i(254)=254’"
ECHO: "[ INFO ] run(): test(); t05 passed: ’bitwise_i2s_s2i(255)=255’"
ECHO: "[ INFO ] run(): test(); t06 passed: ’bitwise_i2s_s2i(0)=0’"
ECHO: "[ INFO ] run(): test(); t07 passed: ’bitwise_i2s_s2i(126)=126’"
ECHO: "[ INFO ] run(): test(); t08 passed: ’bitwise_i2s_s2i(25)=25’"
ECHO: "[ INFO ] run(): test(); t09 passed: ’bitwise_i2s_s2i(1024)=1024’"
ECHO: "[ INFO ] run(): test(); t10 passed: ’bitwise_i2s_s2i(4253)=4253’"
ECHO: "[ INFO ] run(): test(); t11 passed: ’bitwise_i2s_s2i(835)=835’"
```

```
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_i2s_s2i(856)=856'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_imi_32(undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_imi_32([])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_imi_32(254)=7'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_imi_32(254)=7'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_imi_32(255)=7'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_imi_32(0)=0'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_imi_32(126)=7'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_imi_32(25)=6'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_imi_32(1024)=0'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_imi_32(4253)=7'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_imi_32(835)=0'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_imi_32(856)=6'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_and(undef, undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_and([], [])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_and(254, 0)=0'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_and(254, 1)=0'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_and(255, 0)=0'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_and(0, 255)=0'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_and(126, 63)=62'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_and(25, 10)=8'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_and(1024, 512)=0'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_and(4253, 315)=25'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_and(835, 769)=769'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_and(856, 625)=592'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_or(undef, undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_or([], [])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_or(254, 0)=254'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_or(254, 1)=255'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_or(255, 0)=255'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_or(0, 255)=255'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_or(126, 63)=127'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_or(25, 10)=27'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_or(1024, 512)=1536'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_or(4253, 315)=4543'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_or(835, 769)=835'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_or(856, 625)=889'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_xor(undef, undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_xor([], [])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_xor(254, 0)=254'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_xor(254, 1)=255'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_xor(255, 0)=255'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_xor(0, 255)=255'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_xor(126, 63)=65'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_xor(25, 10)=19'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_xor(1024, 512)=1536'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_xor(4253, 315)=4518'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_xor(835, 769)=66'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_xor(856, 625)=297'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_not(undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_not([])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_not(254)=1'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_not(254)=1'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_not(255)=0'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_not(0)=1'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_not(126)=1'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_not(25)=6'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_not(1024)=1023'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_not(4253)=3938'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_not(835)=188'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_not(856)=167'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_lsh(undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_lsh([])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_lsh(254)=508'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_lsh(254)=508'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_lsh(255)=510'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_lsh(0)=0'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_lsh(126)=252'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_lsh(25)=50'"
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_lsh(1024)=2048'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_lsh(4253)=8506'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_lsh(835)=1670'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_lsh(856)=1712'"
ECHO: "[ INFO ] run(): test(); t01 passed: 'bitwise_rsh(undef)=undef'"
ECHO: "[ INFO ] run(): test(); t02 passed: 'bitwise_rsh([])=undef'"
ECHO: "[ INFO ] run(): test(); t03 passed: 'bitwise_rsh(254)=127'"
ECHO: "[ INFO ] run(): test(); t04 passed: 'bitwise_rsh(254)=127'"
ECHO: "[ INFO ] run(): test(); t05 passed: 'bitwise_rsh(255)=127'"
ECHO: "[ INFO ] run(): test(); t06 passed: 'bitwise_rsh(0)=0'"
ECHO: "[ INFO ] run(): test(); t07 passed: 'bitwise_rsh(126)=63'"
ECHO: "[ INFO ] run(): test(); t08 passed: 'bitwise_rsh(25)=12'"
```

```
ECHO: "[ INFO ] run(): test(); t09 passed: 'bitwise_rsh(1024)=512'"
ECHO: "[ INFO ] run(): test(); t10 passed: 'bitwise_rsh(4253)=2126'"
ECHO: "[ INFO ] run(): test(); t11 passed: 'bitwise_rsh(835)=417'"
ECHO: "[ INFO ] run(): test(); t12 passed: 'bitwise_rsh(856)=428'"
```

4 Installing

First install `openscad-amu`. More information can be found at `amu on Thingiverse` and in the GitHib `amu repository` where the source is maintained.

A build script exists for *Linux* and *Cygwin* (pull requests for *macos* are welcome). If `wget` is not available, here is a downloadable link to the `bootstrap` script.

```
$ mkdir tmp && cd tmp

$ wget https://raw.githubusercontent.com/royasutton/openscad-amu/master/snapshots/bootstrap.{bash,conf} .
$ chmod +x bootstrap.bash

$ ./bootstrap.bash --yes --install

$ openscad-seam -v -V
```

If the last step reports the tool version, then the install most likely completed successfully and the temporary directory may be removed as desired.

Now `omdl` can be compiled, verified, and installed. First download the source from `omdl on Thingiverse` or clone the GitHub `omdl repository` and install as follows:

```
$ git clone https://github.com/royasutton/omdl.git
$ cd omdl

$ make install
```

The library and documentation should now have been installed to the OpenSCAD *built-in* library location along with the `omdl` reference documentation that can be views with a web browser.

Have a look in:

- **Linux:** $HOME/.local/share/OpenSCAD/libraries
- **Windows:** My Documents\OpenSCAD\libraries

You may include the desired library component from your project as follows, replacing the version number as needed:

```
include <omdl-v0.4/shapes2de.scad>;
include <omdl-v0.4/shapes3d.scad>;
...
```

5 Test List

File units_resolution.scad

Review model for accuracy.

6 Todo List

globalScope> Global linear_extrude_uls (h, center=false)

This function should be rewritten to use the built-in scaling provided by linear_extrude() in the upper and lower scaling zones.

globalScope> Global pyramid_q (size, center=false)

Support vertex rounding radius.

globalScope> Global pyramid_t (size, center=false)

Support vertex rounding radius.

File shapes3d.scad

Complete rounded cylinder.

globalScope> Global triangle_ppp (v1, v2, v3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

Replace the hull() operation with calculated tangential intersection of the rounded vertexes.

Remove the all or nothing requirement for vertex rounding.

7 Module Index

7.1 Modules

Here is a list of all modules:

8 File Index

8.1 File List

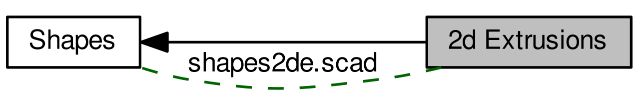
Here is a list of all documented files with brief descriptions:

9 Module Documentation

9.1 2d Extrusions

Extruded two-dimensional geometric shapes.

Collaboration diagram for 2d Extrusions:



Files

- file shapes2de.scad

  Linearly extruded two-dimensional basic shapes.

Functions

- module erectangle (size, h, vr, vrm=0, center=false)
  An extruded rectangle with edge, fillet, and/or chamfer corners.
- module erectangle_c (size, core, h, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)
  An extruded rectangle with a removed rectangular core.
- module erhombus (size, h, vr, center=false)
  An extruded rhombus.
- module etriangle_ppp (v1, v2, v3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by three vertices.
- module etriangle_lp (v, h, vr, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by a list of its three vertices.
- module etriangle_sss (s1, s2, s3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by its three side lengths.
- module etriangle_ls (v, h, vr, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by a list of its three side lengths.
- module etriangle_ls_c (vs, vc, h, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false, center=false)
  A general triangle specified by its sides with a removed triangular core.
- module etriangle_sas (s1, a, s2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by two sides and the included angle.
- module etriangle_asa (a1, s, a2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by a side and two adjacent angles.
- module etriangle_aas (a1, a2, s, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded general triangle specified by a side, one adjacent angle and the opposite angle.
- module etriangle_ss (x, y, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded right-angled triangle specified by its opposite and adjacent side lengths.
- module etriangle_sa (x, y, aa, oa, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)
  An extruded right-angled triangle specified by a side length and an angle.
- module engon (n, r, h, vr, center=false)
  An extruded n-sided equiangular/equilateral regular polygon.
- module eellipse (size, h, center=false)
  An extruded ellipse.
- module eellipse_c (size, core, h, t, co, cr=0, center=false)
  An extruded ellipse with a removed elliptical core.
- module eellipse_s (size, h, a1=0, a2=0, center=false)
  An extruded ellipse sector.
- module eellipse_cs (size, core, h, t, a1=0, a2=0, co, cr=0, center=false)
  An extruded sector of an ellipse with a removed elliptical core.
- module estar2d (size, h, n=5, vr, center=false)
  An extruded two-dimensional star.

9.1.1 Detailed Description

Extruded two-dimensional geometric shapes.

9.1.2 Function Documentation

9.1.2.1 module eellipse (size , h , center = false)

An extruded ellipse.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

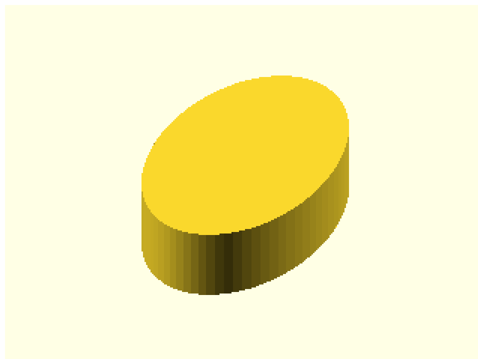


Figure 1: eellipse

```
eellipse( size=[25, 40], h=20, center=true );
```

Definition at line 777 of file shapes2de.scad.

9.1.2.2 module eellipse_c (size , core , h , t , co , cr = 0, center = false)

An extruded ellipse with a removed elliptical core.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
core	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
t	<decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).
co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.

- `size = core + t`; when `t` and `core` are given.

Example

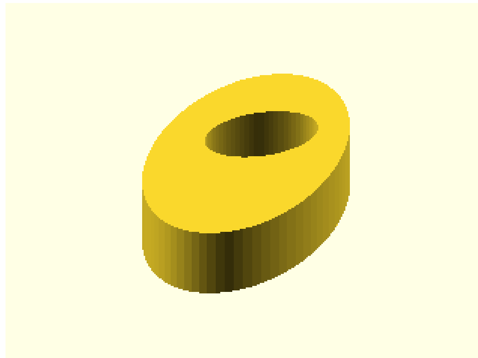


Figure 2: eellipse_c

```
eellipse_c( size=[25,40], core=[16,10], co=[0,10], cr=45, h=20, center=true );
```

Definition at line 817 of file shapes2de.scad.

9.1.2.3 module eellipse_cs (size , core , h , t , a1 = 0, a2 = 0, co , cr = 0, center = false)

An extruded sector of an ellipse with a removed elliptical core.

Parameters

size	$\langle$decimal-list-2$\vert$decimal$\rangle$ A list [rx, ry] of decimals or a single decimal for (rx=ry).
core	$\langle$decimal-list-2$\vert$decimal$\rangle$ A list [rx, ry] of decimals or a single decimal for (rx=ry).
h	$\langle$decimal-list-3:9$\vert$decimal$\rangle$ A list of decimals or a single decimal to specify simple extrusion height.
t	$\langle$decimal-list-2$\vert$decimal$\rangle$ A list [x, y] of decimals or a single decimal for (x=y).
a1	$\langle$decimal$\rangle$ The start angle in degrees.
a2	$\langle$decimal$\rangle$ The stop angle in degrees.
co	$\langle$decimal-list-2$\rangle$ Core offset. A list [x, y] of decimals.
cr	$\langle$decimal$\rangle$ Core z-rotation.
center	$\langle$boolean$\rangle$ Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

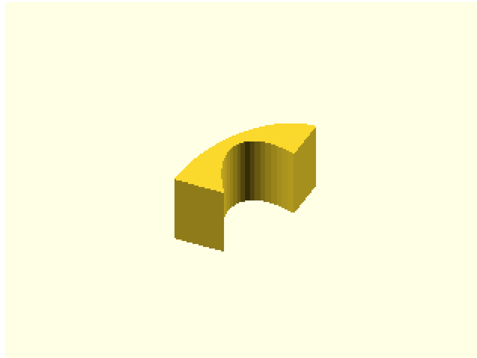


Figure 3: eellipse_cs

```
eellipse_cs( size=[25,40], t=[10,5], a1=90, a2=180, co=[10,0], cr=45, h=20, center=true );
```

Definition at line 897 of file shapes2de.scad.

9.1.2.4 module eellipse_s (size , h , a1 = 0, a2 = 0, center = false)

An extruded ellipse sector.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

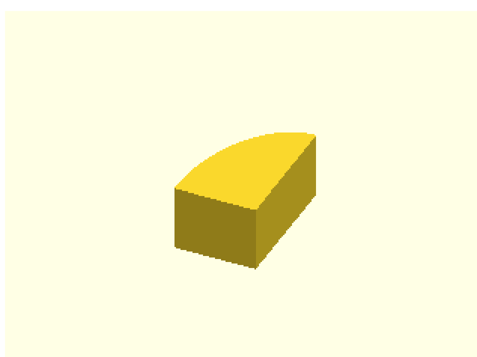


Figure 4: eellipse_s

```
eellipse_s( size=[25,40], h=20, a1=90, a2=180, center=true );
```

Definition at line 852 of file shapes2de.scad.

9.1.2.5 module engon (n , r , h , vr , center = false)

An extruded n-sided equiangular/equilateral regular polygon.

Parameters

n	<decimal> The number of sides.
r	<decimal> The ngon vertex radius.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The vertex rounding radius.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

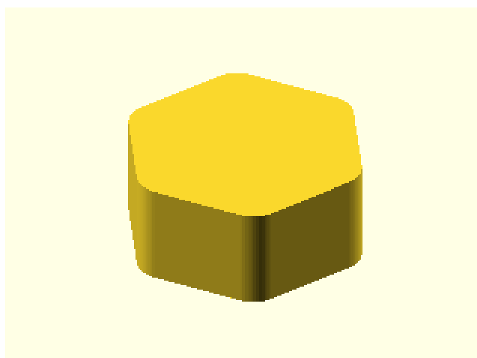


Figure 5: engon

```
engon( n=6, r=25, h=20, vr=6, center=true );
```

See `Wikipedia` for more information.

Definition at line 747 of file shapes2de.scad.

9.1.2.6 module erectangle (size , h , vr , vrm = 0, center = false)

An extruded rectangle with edge, fillet, and/or chamfer corners.

Parameters

size	<decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.

<table>
<tr><td>vr</td><td><decimal-list-4|decimal> The corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.</td></tr>
<tr><td>vrm</td><td><integer> The corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value 0 for fillet and 1 for chamfer.</td></tr>
<tr><td>center</td><td><boolean> Center about origin.</td></tr>
</table>

See also

linear_extrude_uls for a description on specifying `h`.

Example

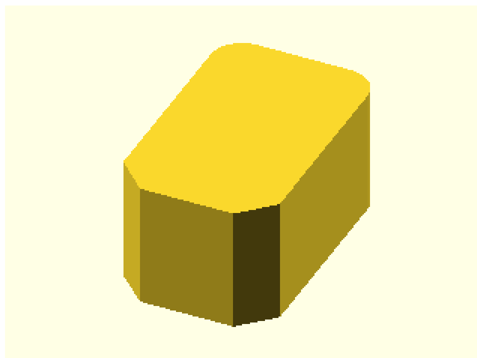


Figure 6: erectangle

```
erectangle( size=[25,40], vr=5, vrm=3, h=20, center=true );
```

Definition at line 114 of file shapes2de.scad.

9.1.2.7 module erectangle_c (size , core , h , t , co , cr = 0, vr , vr1 , vr2 , vrm = 0, vrm1 , vrm2 , center = false)

An extruded rectangle with a removed rectangular core.

Parameters

<table>
<tr><td>size</td><td><decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>core</td><td><decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>h</td><td><decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.</td></tr>
<tr><td>t</td><td><decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>co</td><td><decimal-list-2> Core offset. A list [x, y] of decimals.</td></tr>
<tr><td>cr</td><td><decimal> Core z-rotation.</td></tr>
<tr><td>vr</td><td><decimal-list-4|decimal> The default corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.</td></tr>
<tr><td>vr1</td><td><decimal-list-4|decimal> The outer corner rounding radius.</td></tr>
<tr><td>vr2</td><td><decimal-list-4|decimal> The core corner rounding radius.</td></tr>
<tr><td>vrm</td><td><integer> The default corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value 0 for fillet and 1 for chamfer.</td></tr>
<tr><td>vrm1</td><td><integer> The outer corner radius mode.</td></tr>
<tr><td>vrm2</td><td><integer> The core corner radius mode.</td></tr>
<tr><td>center</td><td><boolean> Center about origin.</td></tr>
</table>

See also

linear_extrude_uls for a description on specifying `h`.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

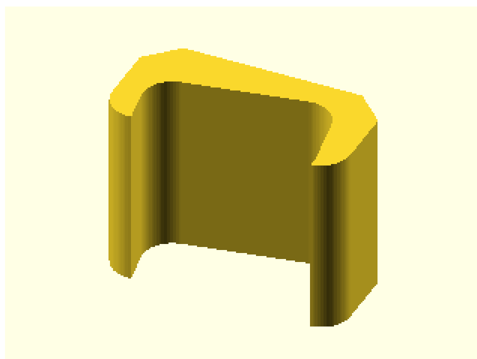


Figure 7: erectangle_c

```
erectangle_c( size=[40,20], t=[10,1], co=[0,-6], cr=10, vr=5, vrm1=12, h=30, center=true );
```

Definition at line 168 of file shapes2de.scad.

9.1.2.8 module erhombus (size , h , vr , center = `false`)

An extruded rhombus.

Parameters

<table>
<tr><td>size</td><td><decimal-list-2|decimal> A list [w, h] of decimals or a single decimal for (w=h).</td></tr>
<tr><td>h</td><td><decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.</td></tr>
<tr><td>vr</td><td><decimal-list-4|decimal> The corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.</td></tr>
<tr><td>center</td><td><boolean> Center about origin.</td></tr>
</table>

See also

linear_extrude_uls for a description on specifying `h`.

Example

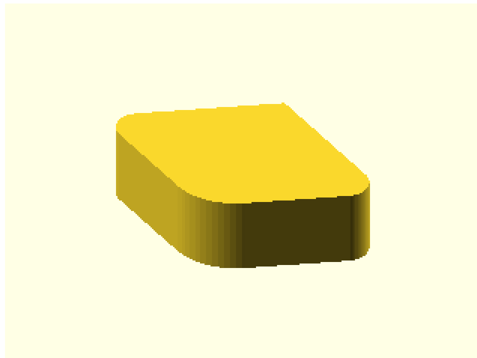


Figure 8: erhombus

```
erhombus( size=[40,25], h=10, vr=[3,0,3,9], center=true );
```

Definition at line 217 of file shapes2de.scad.

9.1.2.9 module estar2d (size , h , n = 5, vr , center = false)

An extruded two-dimensional star.

Parameters

<table>
<tr><td>size</td><td><decimal-list-2|decimal> A list [l, w] of decimals or a single decimal for (size=l=2*w).</td></tr>
<tr><td>h</td><td><decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.</td></tr>
<tr><td>n</td><td><decimal> The number of points.</td></tr>
<tr><td>vr</td><td><decimal-list-3|decimal> The vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).</td></tr>
<tr><td>center</td><td><boolean> Center about origin.</td></tr>
</table>

See also

linear_extrude_uls for a description on specifying h.

Example

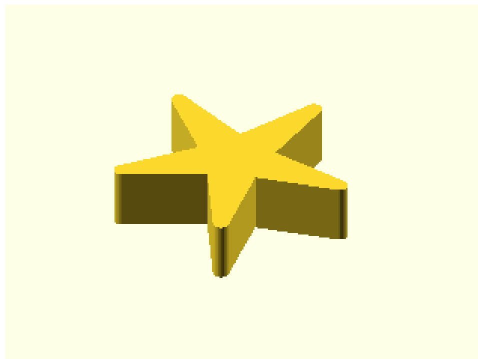


Figure 9: estar2d

```
estar2d( size=[40, 15], h=15, n=5, vr=2, center=true );
```

Definition at line 937 of file shapes2de.scad.

9.1.2.10 module etriangle_aas (a1 , a2 , s , h , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false, center = false)

An extruded general triangle specified by a side, one adjacent angle and the opposite angle.

Parameters

a1	<decimal> The opposite angle 1 in degrees.
a2	<decimal> The adjacent angle 2 in degrees.
s	<decimal> The side length.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
x	<decimal> The side to draw on the positive x-axis (x=1 for s).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying h.

Example

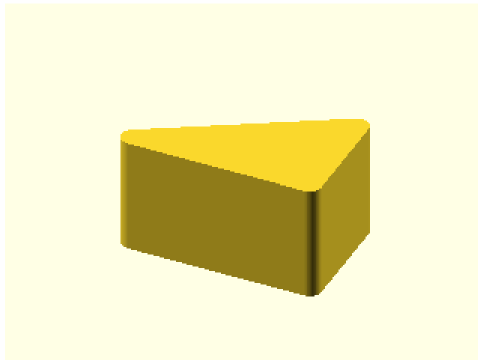


Figure 10: etriangle_aas

```
1       etriangle_aas( a1=60, a2=30, s=40, h=20, vr=2, centroid=true, center=true );
```

Definition at line 599 of file shapes2de.scad.

9.1.2.11 module etriangle_asa (a1 , s , a2 , h , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false, center = false)

An extruded general triangle specified by a side and two adjacent angles.

Parameters

a1	<decimal> The adjacent angle 1 in degrees.
s	<decimal> The side length adjacent to the angles.
a2	<decimal> The adjacent angle 2 in degrees.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
x	<decimal> The side to draw on the positive x-axis (x=1 for s).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

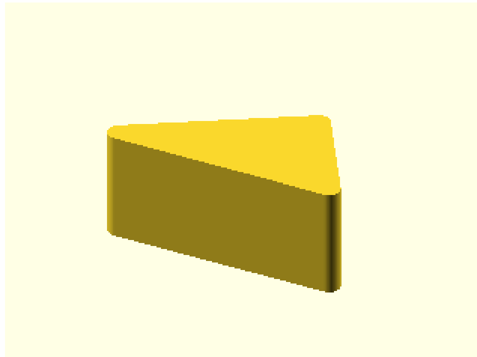


Figure 11: etriangle_asa

```
etriangle_asa( a1=30, s=50, a2=60, h=20, vr=2, centroid=true, center=true );
```

Definition at line 546 of file shapes2de.scad.

9.1.2.12 module etriangle_lp (v , h , vr , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by a list of its three vertices.

Parameters

v	<point-2d-list-3> A list [v1, v2, v3] of points [x, y].
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal-list-3|decimal> The vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

```
t = triangle_sss2lp( 30, 40, 50 );
r = [2, 4, 6];
etriangle_lp( v=t, h=5, vr=r );
```

Definition at line 304 of file shapes2de.scad.

9.1.2.13 module etriangle_ls (v , h , vr , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by a list of its three side lengths.

Parameters

v	<decimal-list-3> A list [s1, s2, s3] of decimals.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal-list-3|decimal> The vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

```
t = triangle_sss2lp( 3, 4, 5 );
s = triangle_lp2ls( t );
etriangle_ls( v=s, h=5, vr=2 );
```

Definition at line 393 of file shapes2de.scad.

9.1.2.14 module etriangle_ls_c (vs , vc , h , co , cr = 0, vr , vr1 , vr2 , centroid = false, incenter = false, center = false)

A general triangle specified by its sides with a removed triangular core.

Parameters

vs	<decimal-list-3|decimal> The size. A list [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
vc	<decimal-list-3|decimal> The core. A list [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<decimal-list-3|decimal> The default vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
vr1	<decimal-list-3|decimal> The outer vertex rounding radius.
vr2	<decimal-list-3|decimal> The core vertex rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

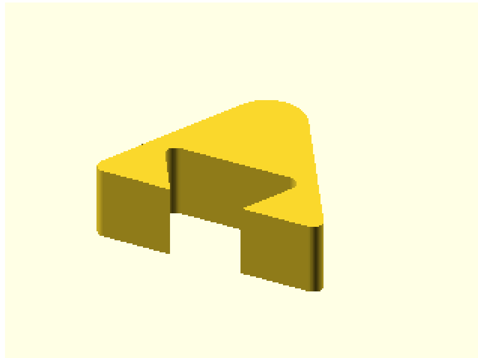


Figure 12: etriangle_ls_c

```
1       etriangle_ls_c(vs=50, vc=30, h=15, co=[0,-10], cr=180, vr=[2,2,8], centroid=true, center=true);
```

Note

The outer and inner triangles centroids are aligned prior to the core removal.

Definition at line 440 of file shapes2de.scad.

9.1.2.15 module etriangle_ppp (v1 , v2 , v3 , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by three vertices.

Parameters

v1	<point-2d> A point [x, y] for vertex 1.
v2	<point-2d> A point [x, y] for vertex 2.
v3	<point-2d> A point [x, y] for vertex 3.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

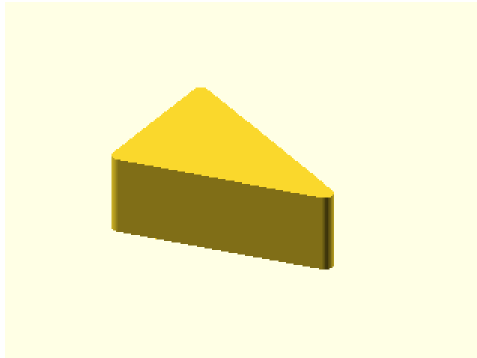


Figure 13: etriangle_ppp

```
etriangle_ppp( v1=[0,0], v2=[5,25], v3=[40,5], h=20, vr=2, centroid=true, center=true );
```

Definition at line 254 of file shapes2de.scad.

9.1.2.16 module etriangle_sa (x , y , aa , oa , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded right-angled triangle specified by a side length and an angle.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
aa	<decimal> The adjacent angle in degrees.
oa	<decimal> The opposite angle in degrees.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

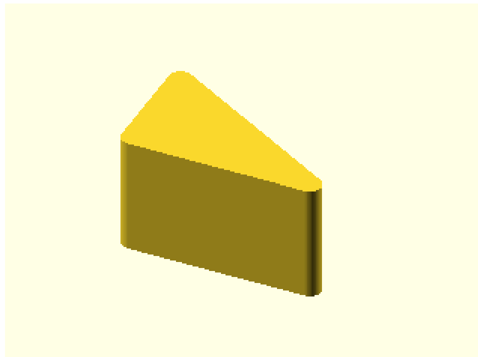


Figure 14: etriangle_sa

```
etriangle_sa( x=40, aa=30, h=20, vr=2, centroid=true, center=true );
```

Note

When both `x` and `y` are given, both triangles are rendered.
When both `aa` and `oa` are given, `aa` is used.

Definition at line 700 of file shapes2de.scad.

9.1.2.17 module etriangle_sas (s1 , a , s2 , h , x = `1`, vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded general triangle specified by two sides and the included angle.

Parameters

s1	<decimal> The length of the side 1.
a	<decimal> The included angle in degrees.
s2	<decimal> The length of the side 2.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
x	<decimal> The side to draw on the positive x-axis (`x=1` for `s1`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.

incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

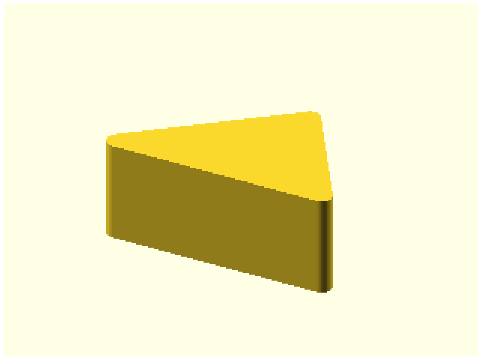


Figure 15: etriangle_sas

```
etriangle_sas( s1=50, a=60, s2=30, h=20, vr=2, centroid=true, center=true );
```

Definition at line 493 of file shapes2de.scad.

9.1.2.18 module etriangle_ss (x , y , h , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`, center = `false`)

An extruded right-angled triangle specified by its opposite and adjacent side lengths.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

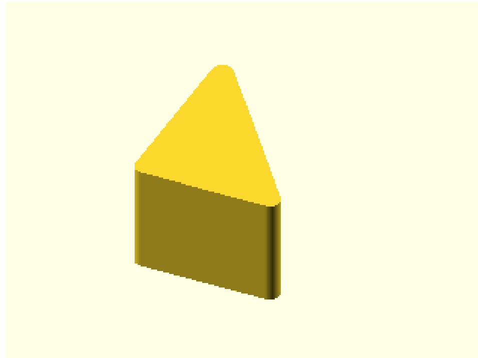


Figure 16: etriangle_ss

```
etriangle_ss( x=30, y=40, h=20, vr=2, centroid=true, center=true );
```

Definition at line 648 of file shapes2de.scad.

9.1.2.19 module etriangle_sss (s1 , s2 , s3 , h , vr , v1r , v2r , v3r , centroid = false, incenter = false, center = false)

An extruded general triangle specified by its three side lengths.

Parameters

s1	<decimal> The length of the side 1 (along the x-axis).
s2	<decimal> The length of the side 2.
s3	<decimal> The length of the side 3.
h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.
center	<boolean> Center about origin.

See also

linear_extrude_uls for a description on specifying `h`.

Example

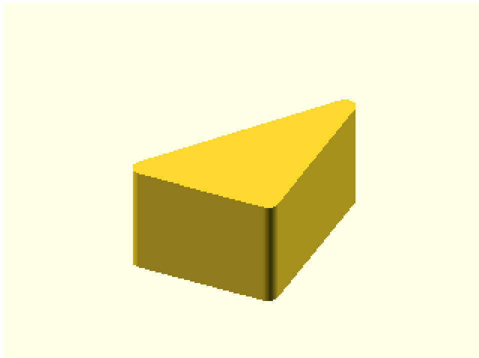


Figure 17: etriangle_sss

```
etriangle_sss( s1=30, s2=40, s3=50, h=20, vr=2, centroid=true, center=true );
```

Definition at line 343 of file shapes2de.scad.

9.2 2d Shapes

Two-dimensional geometric shapes.

Collaboration diagram for 2d Shapes:

Files

- file shapes2d.scad

  Two-dimensional basic shapes.

Functions

- module rectangle (size, vr, vrm=0, center=false)

  A rectangle with edge, fillet, and/or chamfer corners.
- module rectangle_c (size, core, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)

  A rectangle with a removed rectangular core.
- module rhombus (size, vr, center=false)

  A rhombus.
- module triangle_ppp (v1, v2, v3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by three vertices.
- module triangle_lp (v, vr, centroid=false, incenter=false)

  A general triangle specified by a list of its three vertices.
- module triangle_sss (s1, s2, s3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by its three side lengths.
- module triangle_ls (v, vr, centroid=false, incenter=false)

  A general triangle specified by a list of its three side lengths.
- module triangle_ls_c (vs, vc, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false)

  A general triangle specified by its sides with a removed triangular core.
- module triangle_sas (s1, a, s2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by two sides and the included angle.
- module triangle_asa (a1, s, a2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by a side and two adjacent angles.
- module triangle_aas (a1, a2, s, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by a side, one adjacent angle and the opposite angle.
- module triangle_ss (x, y, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A right-angled triangle specified by its opposite and adjacent side lengths.
- module triangle_sa (x, y, aa, oa, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A right-angled triangle specified by a side length and an angle.

- module ngon (n, r, vr)
  An n-sided equiangular/equilateral regular polygon.
- module ellipse (size)
  An ellipse.
- module ellipse_c (size, core, t, co, cr=0)
  An ellipse with a removed elliptical core.
- module ellipse_s (size, a1=0, a2=0)
  An ellipse sector.
- module ellipse_cs (size, core, t, a1=0, a2=0, co, cr=0)
  A sector of an ellipse with a removed elliptical core.
- module star2d (size, n=5, vr)
  A two-dimensional star.

9.2.1 Detailed Description

Two-dimensional geometric shapes.

9.2.2 Function Documentation

9.2.2.1 module ellipse (size)

An ellipse.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).

Example

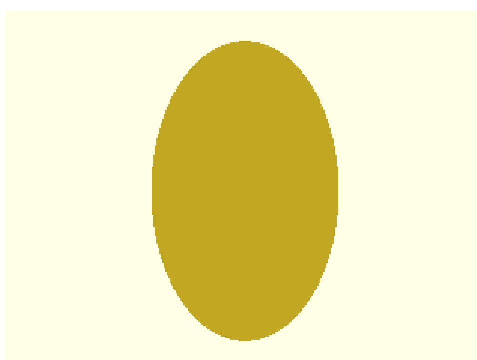


Figure 18: ellipse

```
ellipse( size=[25, 40] );
```

Definition at line 1102 of file shapes2d.scad.

9.2.2.2 module ellipse_c (size , core , t , co , cr = 0)

An ellipse with a removed elliptical core.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
core	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
t	<decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).
co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

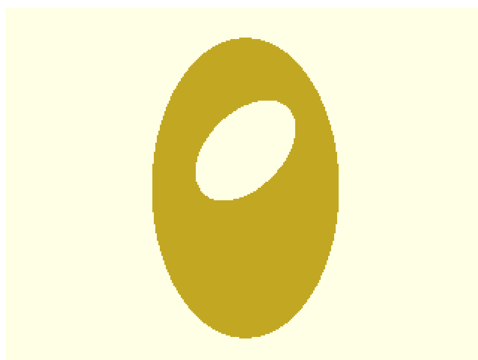


Figure 19: ellipse_c

```
ellipse_c( size=[25,40], core=[16,10], co=[0,10], cr=45 );
```

Definition at line 1143 of file shapes2d.scad.

9.2.2.3 module ellipse_cs (size , core , t , a1 = 0, a2 = 0, co , cr = 0)

A sector of an ellipse with a removed elliptical core.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
core	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
t	<decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.
co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

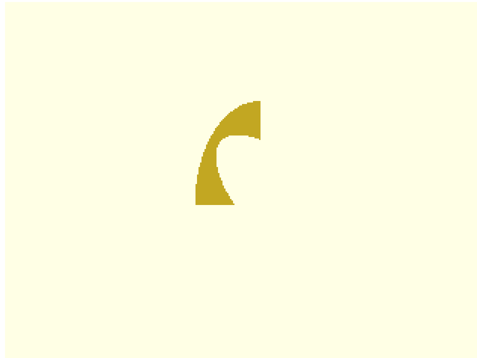


Figure 20: ellipse_cs

```
ellipse_cs( size=[25,40], t=[10,5], a1=90, a2=180, co=[10,0], cr=45);
```

Definition at line 1253 of file shapes2d.scad.

9.2.2.4 module ellipse_s (size , a1 = 0, a2 = 0)

An ellipse sector.

Parameters

size	<decimal-list-2|decimal> A list [rx, ry] of decimals or a single decimal for (rx=ry).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.

Example

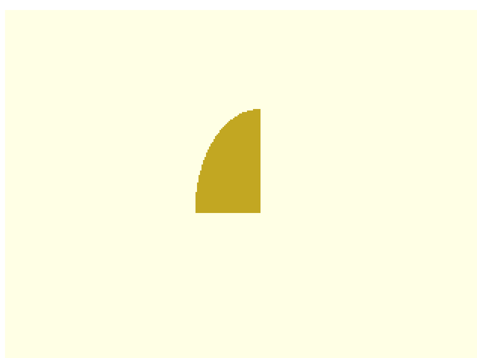


Figure 21: ellipse_s

```
ellipse_s( size=[25,40], a1=90, a2=180 );
```

Definition at line 1185 of file shapes2d.scad.

9.2.2.5 module ngon (n , r , vr)

An n-sided equiangular/equilateral regular polygon.

Parameters

n	<integer> The number of sides.
r	<decimal> The ngon vertex radius.
vr	<decimal> The vertex rounding radius.

Example

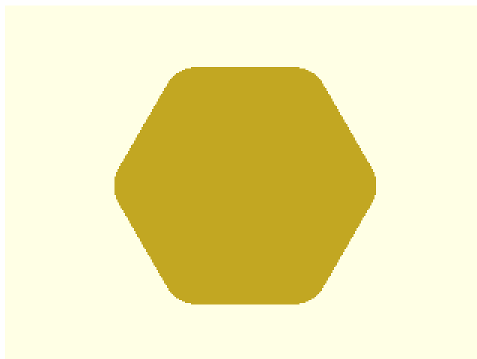


Figure 22: ngon

```
ngon( n=6, r=25, vr=6 );
```

See Wikipedia for more information.

Definition at line 1068 of file shapes2d.scad.

9.2.2.6 module rectangle (size , vr , vrm = 0, center = false)

A rectangle with edge, fillet, and/or chamfer corners.

Parameters

size	<decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).
vr	<decimal-list-4|decimal> The corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
vrm	<integer> The corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value **0** for *fillet* and **1** for *chamfer*.
center	<boolean> Center about origin.

Example

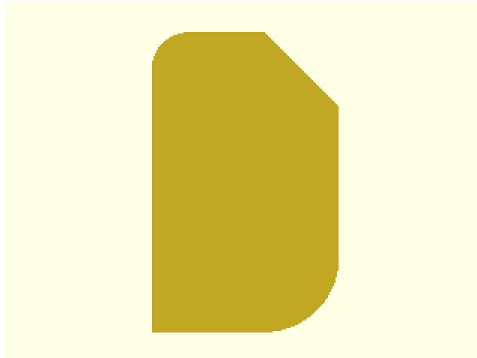


Figure 23: rectangle

```
rectangle( size=[25,40], vr=[0,10,10,5], vrm=4, center=true );
```

Note

A corner *fillet* replaces an edge with a quarter circle of radius `vr`, inset `[vr, vr]` from the corner vertex.
A corner *chamfer* replaces an edge with an isosceles right triangle with side lengths equal to the corresponding corner rounding radius `vr`. Therefore the chamfer length will be `vr*sqrt(2)` at 45 degree angles.

Definition at line 116 of file shapes2d.scad.

9.2.2.7 module rectangle_c (size , core , t , co , cr = `0`, vr , vr1 , vr2 , vrm = `0`, vrm1 , vrm2 , center = `false`)

A rectangle with a removed rectangular core.

Parameters

<table>
<tr><td>size</td><td><decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>core</td><td><decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>t</td><td><decimal-list-2|decimal> A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>co</td><td><decimal-list-2> Core offset. A list [x, y] of decimals.</td></tr>
<tr><td>cr</td><td><decimal> Core z-rotation.</td></tr>
<tr><td>vr</td><td><decimal-list-4|decimal> The default corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.</td></tr>
<tr><td>vr1</td><td><decimal-list-4|decimal> The outer corner rounding radius.</td></tr>
<tr><td>vr2</td><td><decimal-list-4|decimal> The core corner rounding radius.</td></tr>
<tr><td>vrm</td><td><integer> The default corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value 0 for fillet and 1 for chamfer.</td></tr>
<tr><td>vrm1</td><td><integer> The outer corner radius mode.</td></tr>
<tr><td>vrm2</td><td><integer> The core corner radius mode.</td></tr>
<tr><td>center</td><td><boolean> Center about origin.</td></tr>
</table>

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

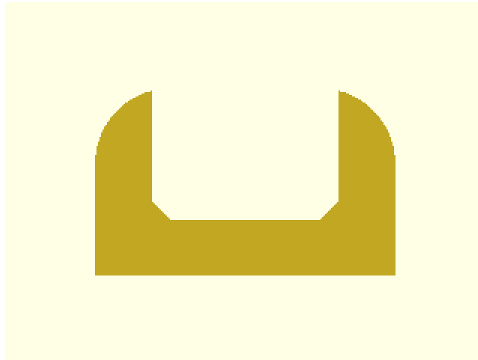


Figure 24: rectangle_c

```
rectangle_c( size=[40,25], t=[15,5], vr1=[0,0,10,10], vr2=2.5, vrm2=3, co=[0,5], center=true );
```

Definition at line 236 of file shapes2d.scad.

9.2.2.8 module rhombus (size , vr , center = `false`)

A rhombus.

Parameters

size	<decimal-list-2|decimal> A list [w, h] of decimals or a single decimal for (w=h).
vr	<decimal-list-4|decimal> The corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.
center	<boolean> Center about origin.

Example

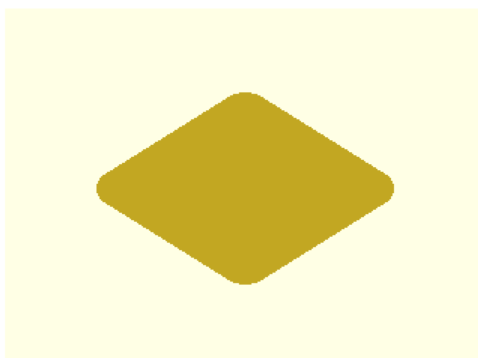


Figure 25: rhombus

```
rhombus( size=[40,25], vr=[2,4,2,4], center=true );
```

See `Wikipedia` for more information.

Definition at line 301 of file shapes2d.scad.

9.2.2.9 module star2d (size , n = `5`, vr)

A two-dimensional star.

Parameters

size	<decimal-list-2|decimal> A list [l, w] of decimals or a single decimal for (size=l=2∗w).
n	<decimal> The number of points.
vr	<decimal-list-3|decimal> The vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).

Example

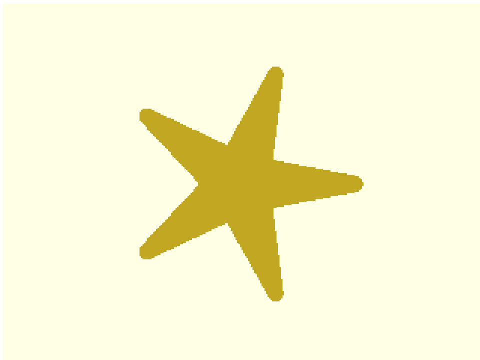


Figure 26: star2d

```
star2d( size=[40, 15], n=5, vr=2 );
```

Definition at line 1300 of file shapes2d.scad.

9.2.2.10 module triangle_aas (a1 , a2 , s , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false)

A general triangle specified by a side, one adjacent angle and the opposite angle.

Parameters

a1	<decimal> The opposite angle 1 in degrees.
a2	<decimal> The adjacent angle 2 in degrees.
s	<decimal> The side length.
x	<integer> The side to draw on the positive x-axis (x=1 for s).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

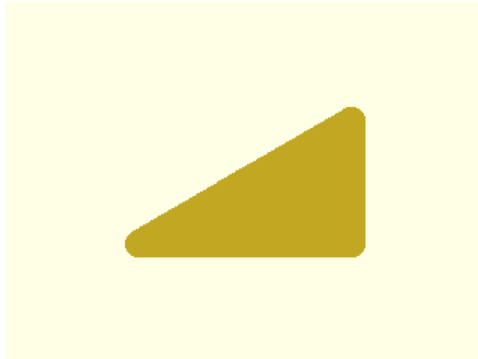


Figure 27: triangle_aas

```
1       triangle_aas( a1=60, a2=30, s=40, vr=2, centroid=true );
```

See `Wikipedia` for more information.

Definition at line 895 of file shapes2d.scad.

9.2.2.11 module triangle_asa (a1 , s , a2 , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false)

A general triangle specified by a side and two adjacent angles.

Parameters

a1	<decimal> The adjacent angle 1 in degrees.
s	<decimal> The side length adjacent to the angles.
a2	<decimal> The adjacent angle 2 in degrees.
x	<integer> The side to draw on the positive x-axis (`x=1` for `s`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

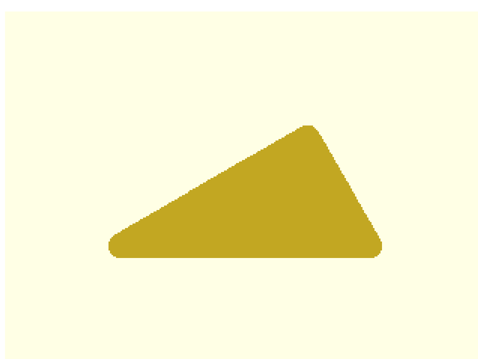


Figure 28: triangle_asa

```
1       triangle_asa( a1=30, s=50, a2=60, vr=2, centroid=true );
```

See Wikipedia for more information.

Definition at line 810 of file shapes2d.scad.

9.2.2.12 module triangle_lp (v , vr , centroid = false, incenter = false)

A general triangle specified by a list of its three vertices.

Parameters

v	<point-2d-list-3> A list [v1, v2, v3] of points [x, y].
vr	<decimal-list-3|decimal> The vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

```
t = triangle_sss2lp( 30, 40, 50 );
r = [2, 4, 6];
triangle_lp( v=t, vr=r  );
```

Definition at line 479 of file shapes2d.scad.

9.2.2.13 module triangle_ls (v , vr , centroid = false, incenter = false)

A general triangle specified by a list of its three side lengths.

Parameters

v	<decimal-list-3> A list [s1, s2, s3] of decimals.
vr	<decimal-list-3|decimal> The vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

```
t = triangle_sss2lp( 3, 4, 5 );
s = triangle_lp2ls( t );
triangle_ls( v=s, vr=2, centroid=true );
```

Definition at line 590 of file shapes2d.scad.

9.2.2.14 module triangle_ls_c (vs , vc , co , cr = 0, vr , vr1 , vr2 , centroid = false, incenter = false)

A general triangle specified by its sides with a removed triangular core.

Parameters

vs	<decimal-list-3|decimal> The size. A list [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
vc	<decimal-list-3|decimal> The core. A list [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).

co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<decimal-list-3|decimal> The default vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
vr1	<decimal-list-3|decimal> The outer vertex rounding radius.
vr2	<decimal-list-3|decimal> The core vertex rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

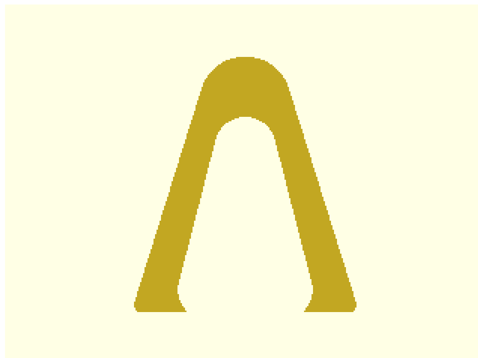


Figure 29: triangle_ls_c

```
triangle_ls_c( vs=[30,50,50], vc=[20,40,40], co=[0,-4], vr1=[1,1,6], vr2=4, centroid=true );
```

Note

The outer and inner triangles centroids are aligned prior to the core removal.

Definition at line 645 of file shapes2d.scad.

9.2.2.15 module triangle_ppp (v1 , v2 , v3 , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A general triangle specified by three vertices.

Parameters

v1	<point-2d> A point [x, y] for vertex 1.
v2	<point-2d> A point [x, y] for vertex 2.
v3	<point-2d> A point [x, y] for vertex 3.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

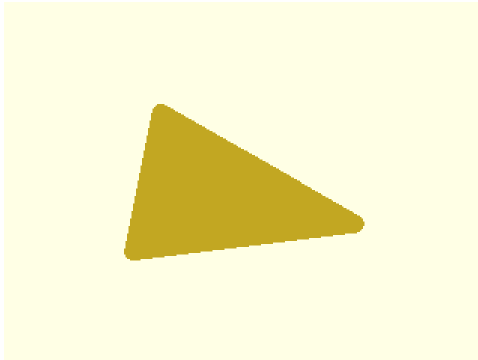


Figure 30: triangle_ppp

```
triangle_ppp( v1=[0,0], v2=[5,25], v3=[40,5], vr=2, centroid=true );
```

Warning

Currently, in order to round any vertex, all must be given a rounding radius, either via `vr` or individually.

Todo Replace the hull() operation with calculated tangential intersection of the rounded vertexes.

Remove the all or nothing requirement for vertex rounding.

Definition at line 402 of file shapes2d.scad.

9.2.2.16 module triangle_sa (x , y , aa , oa , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A right-angled triangle specified by a side length and an angle.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
aa	<decimal> The adjacent angle in degrees.
oa	<decimal> The opposite angle in degrees.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

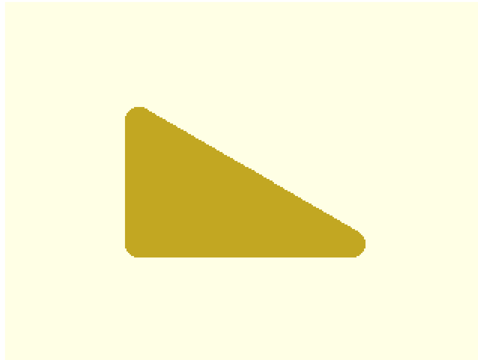


Figure 31: triangle_sa

```
1       triangle_sa( x=40, aa=30, vr=2, centroid=true );
```

Note

When both `x` and `y` are given, both triangles are rendered.
When both `aa` and `oa` are given, `aa` is used.

Definition at line 1015 of file shapes2d.scad.

9.2.2.17 module triangle_sas (s1 , a , s2 , x = 1, vr , v1r , v2r , v3r , centroid = false, incenter = false)

A general triangle specified by two sides and the included angle.

Parameters

s1	<decimal> The length of the side 1.
a	<decimal> The included angle in degrees.
s2	<decimal> The length of the side 2.
x	<integer> The side to draw on the positive x-axis (`x=1` for `s1`).
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

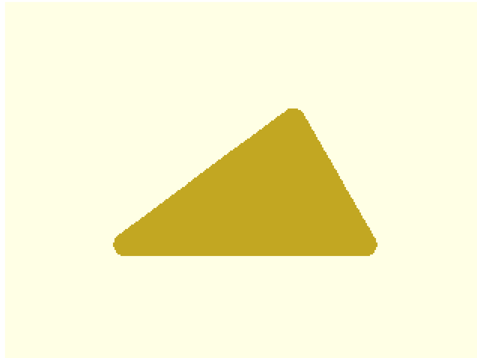


Figure 32: triangle_sas

```
triangle_sas( s1=50, a=60, s2=30, vr=2, centroid=true );
```

See `Wikipedia` for more information.

Definition at line 739 of file shapes2d.scad.

9.2.2.18 module triangle_ss (x , y , vr , v1r , v2r , v3r , centroid = `false`, incenter = `false`)

A right-angled triangle specified by its opposite and adjacent side lengths.

Parameters

x	<decimal> The length of the side along the x-axis.
y	<decimal> The length of the side along the y-axis.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

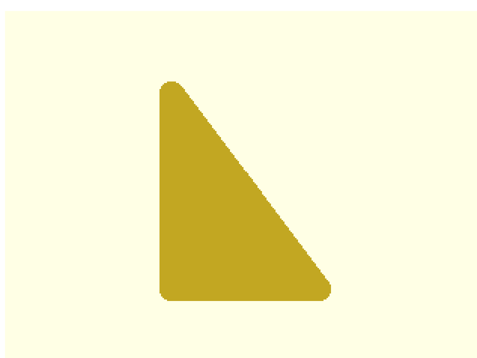


Figure 33: triangle_ss

```
triangle_ss( x=30, y=40, vr=2, centroid=true );
```

Definition at line 972 of file shapes2d.scad.

9.2.2.19 module triangle_sss (s1 , s2 , s3 , vr , v1r , v2r , v3r , centroid = false, incenter = false)

A general triangle specified by its three side lengths.

Parameters

s1	<decimal> The length of the side 1 (along the x-axis).
s2	<decimal> The length of the side 2.
s3	<decimal> The length of the side 3.
vr	<decimal> The default vertex rounding radius.
v1r	<decimal> Vertex 1 rounding radius.
v2r	<decimal> Vertex 2 rounding radius.
v3r	<decimal> Vertex 3 rounding radius.
centroid	<boolean> Center centroid at origin.
incenter	<boolean> Center incenter at origin.

Example

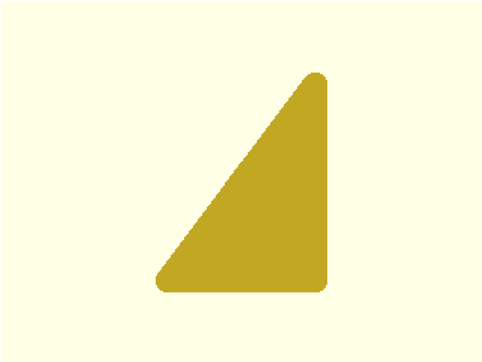


Figure 34: triangle_sss

```
triangle_sss( s1=30, s2=40, s3=50, vr=2, centroid=true );
```

See Wikipedia for more information.

Definition at line 529 of file shapes2d.scad.

9.3 3d Shapes

Three-dimensional geometric shapes.

Collaboration diagram for 3d Shapes:

Files

- file shapes3d.scad

    Three-dimensional basic shapes.

Functions

- module cone (r, h, d, vr, vr1, vr2)

    A cone.
- module cuboid (size, vr, vrm=0, center=false)

    A cuboid with edge, fillet, or chamfer corners.
- module ellipsoid (size)

    An ellipsoid.
- module ellipsoid_s (size, a1=0, a2=0)

    A sector of an ellipsoid.
- module pyramid_t (size, center=false)

    A pyramid with trilateral base formed by four equilateral triangles.
- module pyramid_q (size, center=false)

    A pyramid with quadrilateral base.
- module star3d (size, n=5, half=false)

    A three-dimensional star.
- module torus_rp (size, core, r, l, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, pa=0, ra=360, m=255, center=false, profile=false)

    A rectangular cross-sectional profile revolved about the z-axis.
- module torus_tp (size, core, r, l, co, cr=0, vr, vr1, vr2, pa=0, ra=360, m=255, centroid=false, incenter=false, profile=false,)

    A triangular cross-sectional profile revolved about the z-axis.
- module torus_ep (size, core, r, l, t, a1=0, a2=0, co, cr=0, pa=0, ra=360, m=255, profile=false)

    An elliptical cross-sectional profile revolved about the z-axis.

9.3.1 Detailed Description

Three-dimensional geometric shapes.

9.3.2 Function Documentation

9.3.2.1 module cone (r , h , d , vr , vr1 , vr2)

A cone.

Parameters

r	<decimal> The base radius.
h	<decimal> The height.
d	<decimal> The base diameter.
vr	<decimal> The default corner rounding radius.
vr1	<decimal> The base corner rounding radius.
vr2	<decimal> The point corner rounding radius.

Example

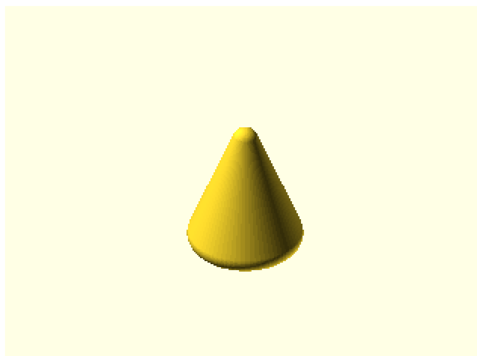


Figure 35: cone

```
cone( h=25, r=10, vr=2 );
```

Definition at line 107 of file shapes3d.scad.

9.3.2.2 module cuboid (size , vr , vrm = 0, center = false)

A cuboid with edge, fillet, or chamfer corners.

Parameters

size	<decimal-list-3|decimal> A list [x, y, z] of decimals or a single decimal for (x=y=z).
vr	<decimal> The corner rounding radius.
vrm	<integer> The radius mode. A 2-bit encoded integer that indicates edge and vertex finish. *B0* controls edge and *B1* controls vertex.
center	<boolean> Center about origin.

Example

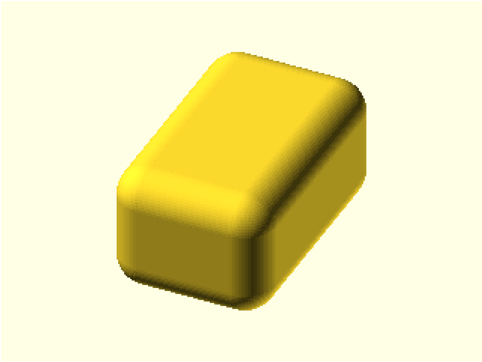


Figure 36: cuboid

```
cuboid( size=[25,40,20], vr=5, center=true );
```

vrm	B1	B0	Description
0	0	0	*fillet* edges with *fillet* vertexes
1	0	1	*chamfer* edges with *sphere* vertexes
2	1	0	*fillet* edges with *chamfer* vertexes
3	1	1	*chamfer* edges with *chamfer* vertexes

Note

Using *fillet* replaces all edges with a quarter circle of radius `vr`, inset `[vr, vr]` from the each edge.
Using *chamfer* replaces all edges with isosceles right triangles with side lengths equal to the corner rounding radius `vr`. Therefore the chamfer length will be `vr*sqrt(2)` at 45 degree angles.

Definition at line 172 of file shapes3d.scad.

9.3.2.3 module ellipsoid (size)

An ellipsoid.

Parameters

size	<decimal-list-2|decimal> A list [w, h] of decimals or a single decimal for (w=h).

Example

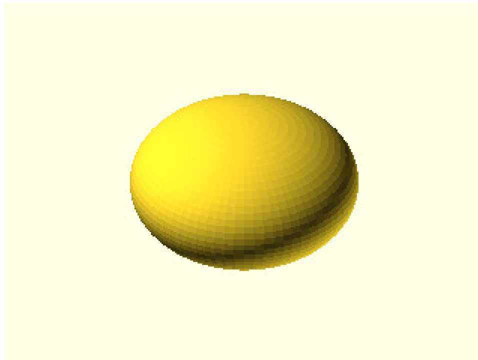


Figure 37: ellipsoid

```
1       ellipsoid( size=[40,25] );
```

Definition at line 257 of file shapes3d.scad.

9.3.2.4 module ellipsoid_s (size , a1 = 0, a2 = 0)

A sector of an ellipsoid.

Parameters

size	<decimal-list-2|decimal> A list [w, h] of decimals or a single decimal for (w=h).
a1	<decimal> The start angle in degrees.
a2	<decimal> The stop angle in degrees.

Example

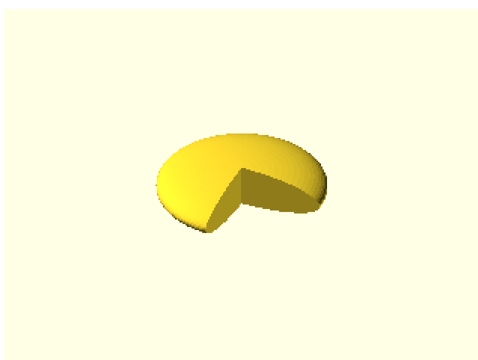


Figure 38: ellipsoid_s

```
1       ellipsoid_s( size=[60,15], a1=0, a2=270 );
```

Definition at line 289 of file shapes3d.scad.

9.3.2.5 module pyramid_q (size , center = false)

A pyramid with quadrilateral base.

Parameters

size	<decimal-list-3|decimal> A list [x, y, z] of decimals or a single decimal for (x=y=z).
center	<boolean> Center about origin.

Example

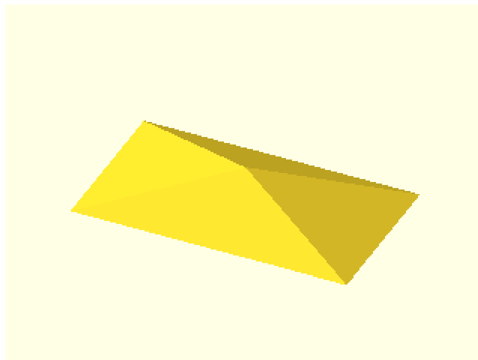


Figure 39: pyramid_q

```
pyramid_q( size=[35,20,5], center=true );
```

Todo Support vertex rounding radius.

Definition at line 391 of file shapes3d.scad.

9.3.2.6 module pyramid_t (size , center = false)

A pyramid with trilateral base formed by four equilateral triangles.

Parameters

size	<decimal> The face radius.
center	<boolean> Center about origin.

Example

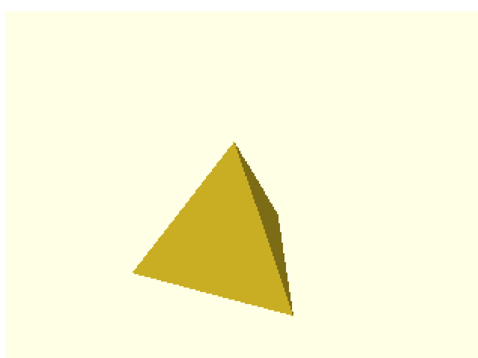


Figure 40: pyramid_t

```
pyramid_t( size=20, center=true );
```

Todo Support vertex rounding radius.

Definition at line 347 of file shapes3d.scad.

9.3.2.7 module star3d (size , n = 5, half = false)

A three-dimensional star.

Parameters

size	<decimal-list-3|decimal> A list [l, w, h] of decimals or a single decimal for (size=l=2∗w=4∗h).
n	<decimal> The number of points.
half	<boolean> Render upper half only.

Example

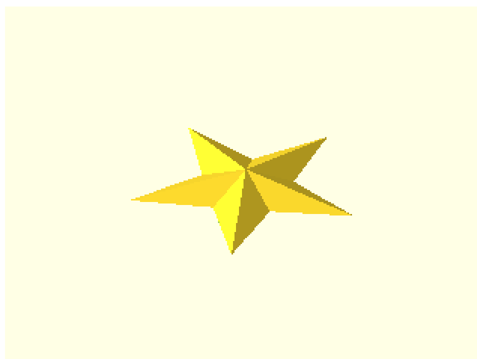


Figure 41: star3d

```
star3d( size=40, n=5, half=true );
```

Definition at line 437 of file shapes3d.scad.

9.3.2.8 module torus_ep (size , core , r , l , t , a1 = 0, a2 = 0, co , cr = 0, pa = 0, ra = 360, m = 255, profile = false)

An elliptical cross-sectional profile revolved about the z-axis.

Parameters

size	<decimal-list-2|decimal> The profile size. A list [x, y] of decimals or a single decimal for (x=y).
core	<decimal-list-2|decimal> The profile core. A list [x, y] of decimals or a single decimal for (x=y).
r	<decimal> The rotation radius.
l	<decimal-list-2|decimal> The elongation length. A list [x, y] of decimals or a single decimal for (x=y)
t	<decimal-list-2|decimal> The profile thickness. A list [x, y] of decimals or a single decimal for (x=y).
a1	<decimal> The profile start angle in degrees.
a2	<decimal> The profile stop angle in degrees.

co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render.
profile	<boolean> Show profile only (do not extrude).

See also

rotate_extrude_tre for description of extrude parameters.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

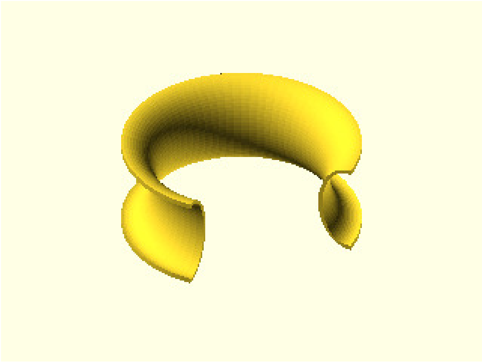


Figure 42: torus_ep

```
torus_ep( size=[20,15], t=[2,4], r=50, a1=0, a2=180, pa=90, ra=270, co=[0,2] );
```

Definition at line 659 of file shapes3d.scad.

9.3.2.9 module torus_rp (size , core , r , l , t , co , cr = 0, vr , vr1 , vr2 , vrm = 0, vrm1 , vrm2 , pa = 0, ra = 360, m = 255, center = false, profile = false)

A rectangular cross-sectional profile revolved about the z-axis.

Parameters

size	<decimal-list-2|decimal> The profile size. A list [x, y] of decimals or a single decimal for (x=y).
core	<decimal-list-2|decimal> The profile core. A list [x, y] of decimals or a single decimal for (x=y).
r	<decimal> The rotation radius.

<table>
<tr><td>l</td><td><decimal-list-2|decimal> The elongation length. A list [x, y] of decimals or a single decimal for (x=y)</td></tr>
<tr><td>t</td><td><decimal-list-2|decimal> The profile thickness. A list [x, y] of decimals or a single decimal for (x=y).</td></tr>
<tr><td>co</td><td><decimal-list-2> Core offset. A list [x, y] of decimals.</td></tr>
<tr><td>cr</td><td><decimal> Core z-rotation.</td></tr>
<tr><td>vr</td><td><decimal-list-4|decimal> The profile default corner rounding radius. A list [v1r, v2r, v3r, v4r] of decimals or a single decimal for (v1r=v2r=v3r=v4r). Unspecified corners are not rounded.</td></tr>
<tr><td>vr1</td><td><decimal-list-4|decimal> The profile outer corner rounding radius.</td></tr>
<tr><td>vr2</td><td><decimal-list-4|decimal> The profile core corner rounding radius.</td></tr>
<tr><td>vrm</td><td><integer> The default corner radius mode. A 4-bit encoded integer that indicates each corner finish. Use bit value 0 for fillet and 1 for chamfer.</td></tr>
<tr><td>vrm1</td><td><integer> The outer corner radius mode.</td></tr>
<tr><td>vrm2</td><td><integer> The core corner radius mode.</td></tr>
<tr><td>pa</td><td><decimal> The profile pitch angle in degrees.</td></tr>
<tr><td>ra</td><td><decimal> The rotation sweep angle in degrees.</td></tr>
<tr><td>m</td><td><integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render.</td></tr>
<tr><td>center</td><td><boolean> Rotate about profile center.</td></tr>
<tr><td>profile</td><td><boolean> Show profile only (do not extrude).</td></tr>
</table>

See also

rotate_extrude_tre for description of extrude parameters.

Thickness `t`

- `core = size - t`; when `t` and `size` are given.
- `size = core + t`; when `t` and `core` are given.

Example

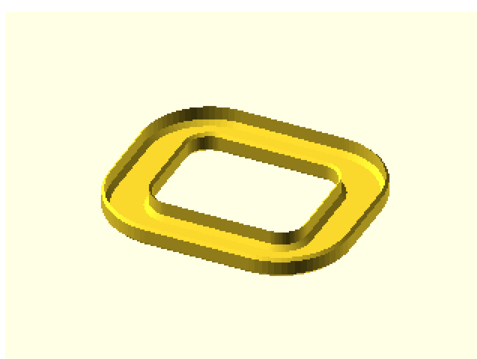


Figure 43: torus_rp

```
torus_rp( size=[40,20], core=[35,20], r=40, l=[90,60], co=[0,2.5], vr=4, vrm=15, center=true );
```

Definition at line 520 of file shapes3d.scad.

9.3.2.10 module torus_tp (size , core , r , l , co , cr = 0, vr , vr1 , vr2 , pa = 0, ra = 360, m = 255, centroid = false, incenter = false, profile = false)

A triangular cross-sectional profile revolved about the z-axis.

Parameters

size	<decimal-list-3|decimal> The size. A list [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
core	<decimal-list-3|decimal> The core. A list [s1, s2, s3] of decimals or a single decimal for (s1=s2=s3).
r	<decimal> The rotation radius.
l	<decimal-list-2|decimal> The elongation length. A list [x, y] of decimals or a single decimal for (x=y)
co	<decimal-list-2> Core offset. A list [x, y] of decimals.
cr	<decimal> Core z-rotation.
vr	<decimal-list-3|decimal> The default vertex rounding radius. A list [v1r, v2r, v3r] of decimals or a single decimal for (v1r=v2r=v3r).
vr1	<decimal-list-3|decimal> The outer vertex rounding radius.
vr2	<decimal-list-3|decimal> The core vertex rounding radius.
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render.
centroid	<boolean> Rotate about profile centroid.
incenter	<boolean> Rotate about profile incenter.
profile	<boolean> Show profile only (do not extrude).

See also

rotate_extrude_tre for description of extrude parameters.

Example

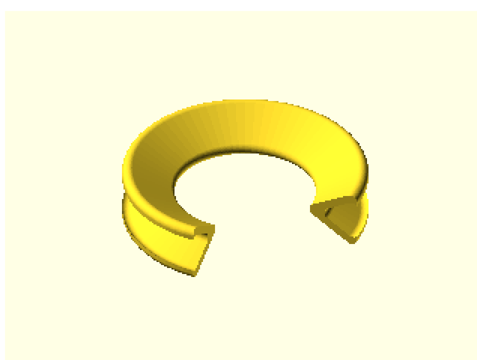


Figure 44: torus_tp

```
torus_tp( size=40, core=30, r=60, co=[0,-4], vr=4, pa=90, ra=270, centroid=true );
```

Note

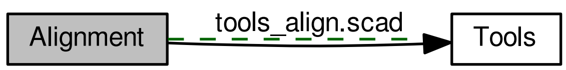
The outer and inner triangles centroids are aligned prior to the core removal.

Definition at line 592 of file shapes3d.scad.

9.4 Alignment

Shape alignment tools.

Collaboration diagram for Alignment:



Files

- file tools_align.scad

    Shape alignment tools.

Functions

- module orient_ll (l=z_axis3d_ul, rl=z_axis3d_ul, r=0)

    Orient a line or vector to a reference line or vector.
- module align_ll (l=z_axis3d_ul, rl=z_axis3d_ul, ap=0, rp=0, r=0, to=origin3d, ro=zero3d)

    Align a line or vector to a reference line or vector.
- module align_l (rl=z_axis3d_ul, rp=0, r=0, to=origin3d, ro=zero3d, d=z_axis_ci)

    Align a shapes' x, y, or z Cartesian axis to reference line or vector.

9.4.1 Detailed Description

Shape alignment tools.

9.4.2 Function Documentation

9.4.2.1 module align_l (rl = z_axis3d_ul, rp = 0, r = 0, to = origin3d, ro = zero3d, d = z_axis_ci)

Align a shapes' x, y, or z Cartesian axis to reference line or vector.

Parameters

rl	<line-3d|line-2d> The reference line or vector.
rp	<integer> The reference-line alignment point (see table).
r	<decimal> Roll about axis `rl` (in degrees).
to	<vector-3d|vector-2d> Translation offset about `rl`.

ro	<decimal-list-1:3|decimal> Rotation offset about `rl` (in degrees).
d	<integer> The Cartesian axis index to align (0, 1, or 2).

The origin will be a translated to the specified alignment point for the reference line `rl`.

ap, rp	alignment point
0	none (no translation)
1	initial
2	median
3	termination
4	initial + termination

See Lines and vectors for argument specification and conventions.

Definition at line 170 of file tools_align.scad.

9.4.2.2 module align_ll (l = z_axis3d_ul, rl = z_axis3d_ul, ap = 0, rp = 0, r = 0, to = origin3d, ro = zero3d)

Align a line or vector to a reference line or vector.

Parameters

l	<line-3d|line-2d> The line or vector to align.
rl	<line-3d|line-2d> The reference line or vector.
ap	<integer> The line alignment point (see table).
rp	<integer> The reference-line alignment point (see table).
r	<decimal> Roll about axis `rl` (in degrees).
to	<vector-3d|vector-2d> Translation offset about `rl`.
ro	<decimal-list-1:3|decimal> Rotation offset about `rl` (in degrees).

The specified alignment point for the line `l` will be a translated to the specified alignment point for the reference line `rl`.

ap, rp	alignment point
0	none (no translation)
1	initial
2	median
3	termination
4	initial + termination

See Lines and vectors for argument specification and conventions.

Definition at line 102 of file tools_align.scad.

9.4.2.3 module orient_ll (l = z_axis3d_ul, rl = z_axis3d_ul, r = 0)

Orient a line or vector to a reference line or vector.

Parameters

l	<line-3d|line-2d> The line or vector to align.
rl	<line-3d|line-2d> The reference line or vector.
r	<decimal> Roll about axis `rl` (in degrees).

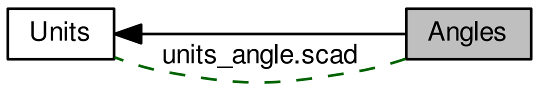
See Lines and vectors for argument specification and conventions.

Definition at line 58 of file tools_align.scad.

9.5 Angles

Angle units and conversions.

Collaboration diagram for Angles:



Files

- file units_angle.scad

  Angle units and conversions.

Functions

- function unit_angle_name (u=base_unit_angle)

  Return the name of an angle unit identifier.
- function convert_angle (a, from=base_unit_angle, to=base_unit_angle)

  Convert an angle from some units to another.

Variables

- base_unit_angle = "d"

  <string> The base units for angle measurements.

9.5.1 Detailed Description

Angle units and conversions.

These functions allow for angles to be specified with units. Angles specified with units are independent of (base_unit_angle). There are also unit conversion functions for converting from one unit to another.

The table below enumerates the supported units.

units id	description	type
r	radian	decimal
d	degree	decimal
dms	degree, minute, second	decimal-list-3

Example

```
include <units/units_angle.scad>;

base_unit_angle = "d";
```

```
// get base unit name
un = unit_angle_name();

// absolute angle measurements in base unit.
c1 = convert_angle(pi/6, "r");
c2 = convert_angle(pi/4, "r");
c3 = convert_angle(180, "d");
c4 = convert_angle([30, 15, 50], "dms");

// convert between units.
c5 = convert_angle([30, 15, 50], from="dms", to="r");
c6 = convert_angle(0.528205, from="r", to="dms");
```

Result (base_angle_length = **r**):

```
ECHO: un = "radian"
ECHO: c1 = 0.523599
ECHO: c2 = 0.785398
ECHO: c3 = 3.14159
ECHO: c4 = 0.528205
ECHO: c5 = 0.528205
ECHO: c6 = [30, 15, 50.102]
```

Result (base_angle_length = **d**):

```
ECHO: un = "degree"
ECHO: c1 = 30
ECHO: c2 = 45
ECHO: c3 = 180
ECHO: c4 = 30.2639
ECHO: c5 = 0.528205
ECHO: c6 = [30, 15, 50.102]
```

Result (base_angle_length = **dms**):

```
ECHO: un = "degree, minute, second"
ECHO: c1 = [29, 59, 60]
ECHO: c2 = [45, 0, 0]
ECHO: c3 = [180, 0, 0]
ECHO: c4 = [30, 15, 50]
ECHO: c5 = 0.528205
ECHO: c6 = [30, 15, 50.102]
```

9.5.2 Function Documentation

9.5.2.1 function convert_angle (a , from = **base_unit_angle**, to = **base_unit_angle**)

Convert an angle from some units to another.

Parameters

a	<decimal|decimal-list-3> An angle to convert.
from	<string> The units of the angle to be converted.
to	<string> A units to which the angle should be converted.

Returns

<decimal|decimal-list-3> The conversion result. Returns **undef** for identifiers that are not defined.

9.5.2.2 function unit_angle_name (u = **base_unit_angle**)

Return the name of an angle unit identifier.

Parameters

u	<string> An angle unit identifier.

Returns

<string> The units name for the given angle unit identifier. Returns **undef** for identifiers that are not defined.

9.6 Bitwise

Base-two bitwise binary operations.

Collaboration diagram for Bitwise:

Files

- file math_bitwise.scad

    Mathematical base-two bitwise binary functions.

Functions

- function bitwise_is_equal (v, b, t=1)

    Test if a base-two bit position of an integer value equals a test bit.
- function bitwise_i2v (v, w=1, bv=1)

    Encode an integer value as a base-two list of bits.
- function bitwise_v2i (v)

    Decode a base-two list of bits to an integer value.
- function bitwise_i2s (v, w=1)

    Encode an integer value as a base-two string of bits.
- function bitwise_s2i (v)

    Decode a base-two string of bits to an integer value.
- function bitwise_imi (v, w, s)

    Decode the integer in a value at a shifted base-two bit mask of width-w.
- function bitwise_and (v1, v2, bv=1)

    Base-two bitwise AND operation for integers.
- function bitwise_or (v1, v2, bv=1)

    Base-two bitwise OR operation for integers.
- function bitwise_xor (v1, v2, bv=1)

    Base-two bitwise XOR operation for integers.
- function bitwise_not (v, w=1, bv=1)

    Base-two bitwise NOT operation for an integer.
- function bitwise_lsh (v, s=1, bm=1, bv=1)

    Base-two bitwise left-shift operation for an integer.
- function bitwise_rsh (v, s=1)

    Base-two bitwise right-shift operation for an integer.

9.6.1 Detailed Description

Base-two bitwise binary operations.

See Wikipedia binary numbers and operations for more information.

See validation results.

9.6.2 Function Documentation

9.6.2.1 function bitwise_and (v1 , v2 , bv = 1)

Base-two bitwise AND operation for integers.

Parameters

v1	<integer> An integer value.
v2	<integer> An integer value.
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise AND of `v1` and `v2`. Returns **undef** when `v1` or `v2` is not an integer.

9.6.2.2 function bitwise_i2s (v , w = 1)

Encode an integer value as a base-two string of bits.

Parameters

v	<integer> An integer value.
w	<integer> The minimum bit width.

Returns

<bit-string> of bits base-two encoding of the integer value. Returns **undef** when `v` or `w` is not an integer.

9.6.2.3 function bitwise_i2v (v , w = 1, bv = 1)

Encode an integer value as a base-two list of bits.

Parameters

v	<integer> An integer value.
w	<integer> The minimum bit width.
bv	(an internal recursion loop variable).

Returns

<bit-list> of bits base-two encoding of the integer value. Returns **undef** when `v` or `w` is not an integer.

9.6.2.4 function bitwise_imi (v , w , s)

Decode the integer in a value at a shifted base-two bit mask of width-w.

Parameters

v	<integer> An integer value.
w	<integer> The bit mask width.
s	<integer> The bit mask shift offset.

Returns

<integer> value of the `w` bits of `v` starting at bit position `s` up to bit `(w+s-1)`.

9.6.2.5 function bitwise_is_equal (v , b , t = 1)

Test if a base-two bit position of an integer value equals a test bit.

Parameters

v	<integer> An integer value.
b	<integer> A base-two bit position.
t	<bit> The bit test value [0|1].

Returns

<boolean> **true** when the base-two bit position of the integer value equals `t`, otherwise returns **false**.

9.6.2.6 function bitwise_lsh (v , s = 1, bm = 1, bv = 1)

Base-two bitwise left-shift operation for an integer.

Parameters

v	<integer> An integer value.
s	<integer> The number of bits to shift.
bm	(an internal recursion loop variable).
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise left-shift of `v` by `s` bits. Returns **undef** when `v` or `s` is not an integer.

9.6.2.7 function bitwise_not (v , w = 1, bv = 1)

Base-two bitwise NOT operation for an integer.

Parameters

v	<integer> An integer value.
w	<integer> The minimum bit width.
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise NOT of `v`. Returns **undef** when `v` is not an integer.

9.6.2.8 function bitwise_or (v1 , v2 , bv = 1)

Base-two bitwise OR operation for integers.

Parameters

v1	<integer> An integer value.
v2	<integer> An integer value.
bv	(an internal recursion loop variable).

Returns

<integer> result of the base-two bitwise OR of `v1` and `v2`. Returns **undef** when `v1` or `v2` is not an integer.

9.6.2.9 function bitwise_rsh (v , s = 1)

Base-two bitwise right-shift operation for an integer.

Parameters

v	<integer> An integer value.
s	<integer> The number of bits to shift.

Returns

<integer> result of the base-two bitwise right-shift of `v` by `s` bits. Returns **undef** when `v` or `s` is not an integer.

9.6.2.10 function bitwise_s2i (v)

Decode a base-two string of bits to an integer value.

Parameters

v	<bit-string> A value encoded as a base-two string of bits.

Returns

<integer> value encoding of the base-two string of bits. Returns **undef** when `v` is not a string of bit values.

9.6.2.11 function bitwise_v2i (v)

Decode a base-two list of bits to an integer value.

Parameters

v	<bit-list> A value encoded as a base-two list of bits.

Returns

<integer> value encoding of the base-two list of bits. Returns **undef** when `v` is not a list of bit values.

9.6.2.12 function bitwise_xor (v1 , v2 , bv = 1)

Base-two bitwise XOR operation for integers.

Parameters

v1	<integer> An integer value.
v2	<integer> An integer value.
bv	(an internal recursion loop variable).

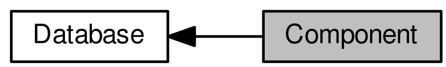
Returns

<integer> result of the base-two bitwise XOR of `v1` and `v2`. Returns **undef** when `v1` or `v2` is not an integer.

9.7 Component

Component specifications.

Collaboration diagram for Component:

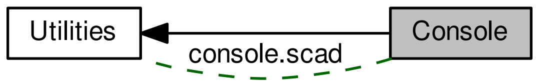


Component specifications.

9.8 Console

Console message logging.

Collaboration diagram for Console:



Files

- file console.scad

  Message logging functions.

Functions

- function stack (b=0, t=0)

  Format the function call stack as a string.
- module log_echo (m)

  Output message to console.
- module log_debug (m)

  Output diagnostic message to console.
- module log_info (m)

  Output information message to console.
- module log_warn (m)

  Output warning message to console.
- module log_error (m)

  Output error message to console.

9.8.1 Detailed Description

Console message logging.

Example

```
use <console.scad>;

$log_debug = true;
message = "console log message";

// general
log_echo( message );

// debugging
log_debug( message );
log_debug( message, $log_debug = false );
```

```
// information
log_info( message );

// warning
log_warn( message );

// error
log_error( message );
```

Result

```
ECHO: "console log message"
ECHO: "[ DEBUG ] root(); console log message"
ECHO: "[ INFO ] root(); console log message"
ECHO:
ECHO: "root()"
ECHO: "###################################"
ECHO: "# [ WARNING ] console log message #"
ECHO: "###################################"
ECHO:
ECHO: "root()"
ECHO: "###################################"
ECHO: "###################################"
ECHO: "##                               ##"
ECHO: "## [ ERROR ] console log message ##"
ECHO: "##                               ##"
ECHO: "###################################"
ECHO: "###################################"
```

9.8.2 Function Documentation

9.8.2.1 module log_debug (m)

Output diagnostic message to console.

Parameters

m	<string> An output message.

Message is written if and only if `$log_debug` is `true`.

Definition at line 105 of file console.scad.

9.8.2.2 module log_echo (m)

Output message to console.

Parameters

m	<string> An output message.

Definition at line 90 of file console.scad.

9.8.2.3 module log_error (m)

Output error message to console.

Parameters

m	<string> An output message.

Output an error message to the console. Ideally, rendering should halt and the script should exit. However, no suitable abort function exists. To alert of the critical error, the error message is also rendered graphically.

Definition at line 166 of file console.scad.

9.8.2.4 module log_info (m)

Output information message to console.

Parameters

m	<string> An output message.

Definition at line 122 of file console.scad.

9.8.2.5 module log_warn (m)

Output warning message to console.

Parameters

m	<string> An output message.

Definition at line 137 of file console.scad.

9.8.2.6 function stack (b = `0`, t = `0`)

Format the function call stack as a string.

Parameters

b	<integer> The stack index bottom offset. Return function names above this offset.
t	<integer> The stack index top offset. Return function names below this offset.

Returns

<string> A string-formatted colon-separated list of functions names for the current function call stack.

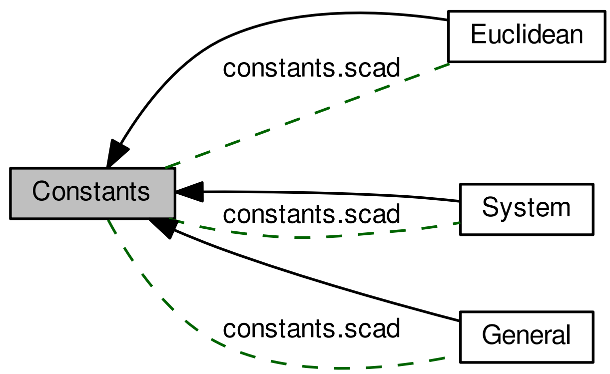
Note

Returns **undef** when `b` is greater than the current number of function instances (ie: `b > $parent_modules-1`).
Returns the string "`root()`" when the function call stack is empty (ie: at the root of a script).

9.9 Constants

Design constant definitions.

Collaboration diagram for Constants:



Modules

- Euclidean

  Euclidean space axis mapping.
- General

  General design constants.
- System

  System and program limits.

Files

- file constants.scad

  Design constant definitions.

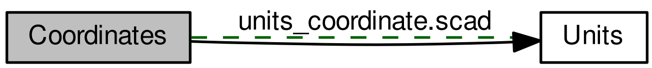
9.9.1 Detailed Description

Design constant definitions.

9.10 Coordinates

Coordinate systems and conversions.

Collaboration diagram for Coordinates:



Files

- file units_coordinate.scad

    Coordinate systems and conversions.

Functions

- function coordinates_name (s=base_coordinates)

    Return the name of the given coordinate system identifier.
- function convert_coordinate (c, from=base_coordinates, to=base_coordinates)

    Convert point from one coordinate system to another.
- function coordinates_cpc (c, r, t=false)

    Radially scale a list of 2d cartesian coordinates.
- function coordinates_pc (p, r, t=false)

    Radially scale and convert a list of 2d polar coordinates to cartesian.
- function coordinates_csc (c, r, t=false)

    Radially scale a list of 3d cartesian coordinates.
- function coordinates_sc (s, r, t=false)

    Radially scale and convert a list of 3d spherical coordinates to cartesian.

Variables

- base_coordinates = "c"

    <string> The base coordinate system.
- coordinates_positive_angles = true

    <boolean> When converting to angular measures add 360 to negative angles.

9.10.1 Detailed Description

Coordinate systems and conversions.

These functions allow for geometric points in space to be specified using multiple coordinate systems. Some geometric calculations are specified more naturally in one or another coordinate system. These conversion functions allow for the movement between the most convenient for a particular application.

For more information see Wikipedia on coordinate system.

The table below enumerates the supported coordinate systems.

system id	description	dimensions	point convention
c	cartesian	2d or 3d	[x, y] or [x, y, z]
p	polar	2d	[r, aa]
y	cylindrical	3d	[r, aa, z]
s	spherical	3d	[r, aa, pa]

The symbols used in the convention column are as follows:

symbol	description	units	reference
x, y, z	coordinate distance	any	xyz-axis
r	radial distance	any	z-axis / xyz-origin
aa	azimuthal angle	degrees	positive x-axis
pa	polar / zenith angle	degrees	positive z-axis

Note

The azimuthal angle is a measure of the radial vector orthogonal projection onto the xy-plane measured from the positive x-axis.
The polar angle is measured from the z-axis (zenith) to the radial vector.

Example

```
include <units/units_coordinate.scad>;

base_coordinates = "c";

// get the base coordinate system name
cs = coordinates_name();

// absolute coordinates in a specified coordinate system.
c1 = convert_coordinate([1, 1, 1], "c");
c2 = convert_coordinate([1, 180], "p");
c3 = convert_coordinate([1, 90, -1], "y");
c4 = convert_coordinate([1, 5, 50], "s");

// convert between system.
c5 = convert_coordinate([10*sqrt(2), 45, 45], from="s", to="y");
c6 = convert_coordinate([sqrt(2), 45], from="p", to="c");
```

Result (base_coordinates = **c**):

```
1 ECHO: cs = "cartesian"
2 ECHO: c1 = [1, 1, 1]
3 ECHO: c2 = [-1, 0]
4 ECHO: c3 = [0, 1, -1]
5 ECHO: c4 = [0.763129, 0.0667652, 0.642788]
6 ECHO: c5 = [10, 45, 10]
7 ECHO: c6 = [1, 1]
```

Result (base_coordinates = **p**):

```
1 ECHO: cs = "polar"
2 ECHO: c1 = [1.41421, 45]
3 ECHO: c2 = [1, 180]
4 ECHO: c3 = [1, 90]
5 ECHO: c4 = [0.766044, 5]
6 ECHO: c5 = [10, 45, 10]
7 ECHO: c6 = [1, 1]
```

Result (base_coordinates = **y**):

```
ECHO: cs = "cylindrical"
ECHO: c1 = [1.41421, 45, 1]
ECHO: c2 = [1, 180, 0]
ECHO: c3 = [1, 90, -1]
ECHO: c4 = [0.766044, 5, 0.642788]
ECHO: c5 = [10, 45, 10]
ECHO: c6 = [1, 1]
```

Result (base_coordinates = **s**):

```
ECHO: cs = "spherical"
ECHO: c1 = [1.73205, 45, 54.7356]
ECHO: c2 = [undef, 180, undef]
ECHO: c3 = [1.41421, 90, 135]
ECHO: c4 = [1, 5, 50]
ECHO: c5 = [10, 45, 10]
ECHO: c6 = [1, 1]
```

9.10.2 Function Documentation

9.10.2.1 function convert_coordinate (c , from = base_coordinates, to = base_coordinates)

Convert point from one coordinate system to another.

Parameters

c	<point> A point to convert.
from	<string> The coordinate system identifier of the point to be converted.
to	<string> The coordinate system identifier to which the point should be converted.

Returns

<point> The converted result. Returns **undef** for identifiers that are not defined.

9.10.2.2 function coordinates_cpc (c , r , t = `false`)

Radially scale a list of 2d cartesian coordinates.

Parameters

c	<coords-2d> A list of cartesian coordinates [[x, y], ...].
r	<decimal> A polar radius.
t	<boolean> Translate or scale radius.

Returns

<coords-2d> A list of scaled cartesian coordinates.

When `t` is **true**, all coordinates will terminate on a circle of radius `r`. When `t` is **false**, the radius of each coordinate is scaled by `r`.

9.10.2.3 function coordinates_csc (c , r , t = `false`)

Radially scale a list of 3d cartesian coordinates.

Parameters

c	<coords-3d> A list of cartesian coordinates [[x, y, z], ...].
r	<decimal> A spherical radius.
t	<boolean> Translate or scale radius.

Returns

<coords-3d> A list of scaled cartesian coordinates.

When `t` is **true**, all coordinates will terminate on a sphere of radius `r`. When `t` is **false**, the radius of each coordinate is scaled by `r`.

9.10.2.4 function coordinates_name (s = base_coordinates)

Return the name of the given coordinate system identifier.

Parameters

s	<string> A coordinate system identifier.

Returns

<string> The system name for the given identifier. Returns **undef** for identifiers that are not defined.

9.10.2.5 function coordinates_pc (p, r, t = `false`)

Radially scale and convert a list of 2d polar coordinates to cartesian.

Parameters

c	<coords-2d> A list of polar coordinates [[r, aa], ...].
r	<decimal> A polar radius.
t	<boolean> Translate or scale radius.

Returns

<coords-2d> A list of scaled cartesian coordinates.

When `t` is **true**, all coordinates will terminate on a circle of radius `r`. When `t` is **false**, the radius of each coordinate is scaled by `r`.

9.10.2.6 function coordinates_sc (s, r, t = `false`)

Radially scale and convert a list of 3d spherical coordinates to cartesian.

Parameters

c	<coords-3d> A list of spherical coordinates [[r, aa, pa], ...].
r	<decimal> A spherical radius.
t	<boolean> Translate or scale radius.

Returns

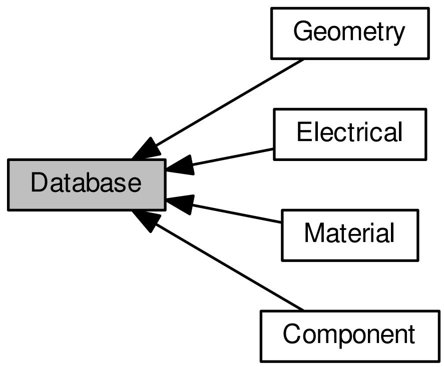
<coords-3d> A list of scaled cartesian coordinates.

When `t` is **true**, all coordinates will terminate on a sphere of radius `r`. When `t` is **false**, the radius of each coordinate is scaled by `r`.

9.11 Database

Design specification data.

Collaboration diagram for Database:



Modules

- Component

  Component specifications.
- Electrical

  Electrical specifications.
- Geometry

  Predefined geometry.
- Material

  Material specifications.

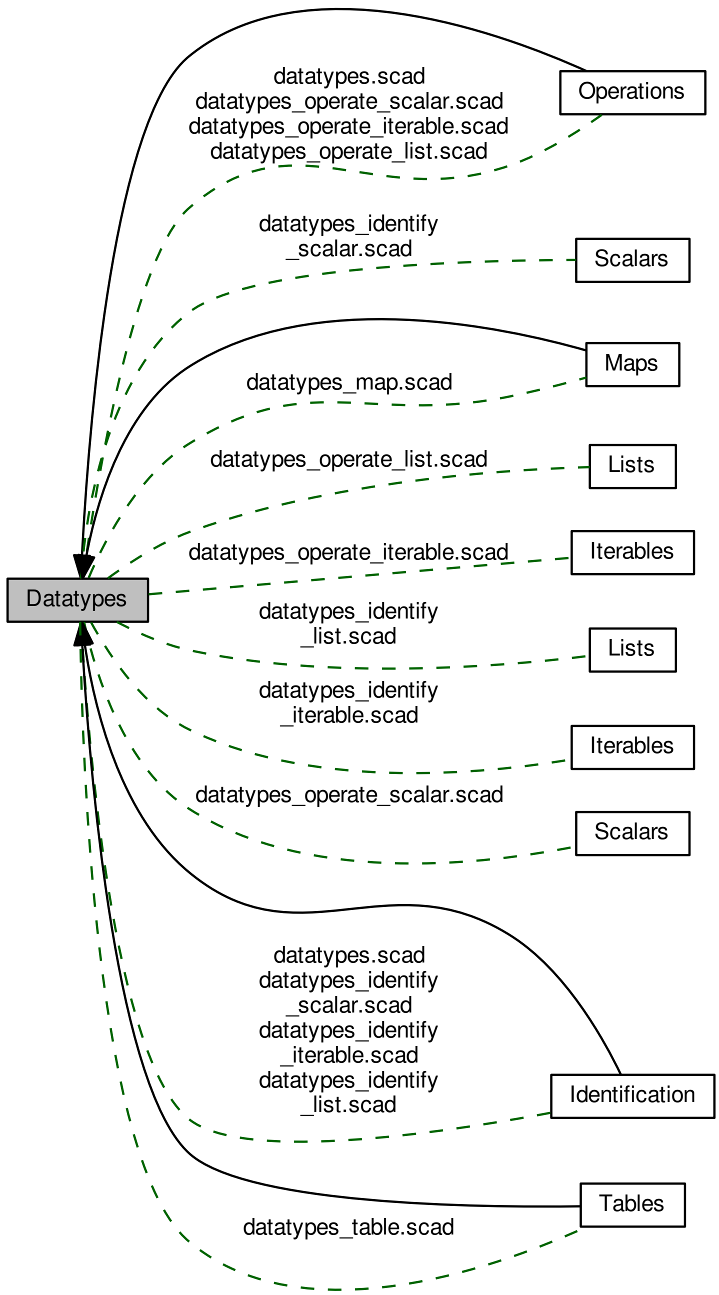
9.11.1 Detailed Description

Design specification data.

9.12 Datatypes

Data type definitions and operators.

Collaboration diagram for Datatypes:



Modules

- Identification

  Compile-time data type identification and tests.
- Maps

  Map data type operations.
- Operations

  Data type operation.
- Tables

  Table data type operations.

Files

- file datatypes.scad

  Data type identification and operations.
- file datatypes_identify_scalar.scad

  Scalar data type identification.
- file datatypes_identify_iterable.scad

  Iterable data type identification.
- file datatypes_identify_list.scad

  List data type identification.
- file datatypes_operate_scalar.scad

  Scalar data type operations.
- file datatypes_operate_iterable.scad

  Iterable data type operations.
- file datatypes_operate_list.scad

  List data type operations.
- file datatypes_map.scad

  Map data type operations.
- file datatypes_table.scad

  Table data type operations.

9.12.1 Detailed Description

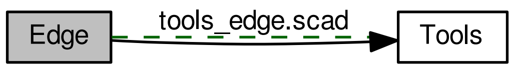
Data type definitions and operators.

See Data types for nomenclature, assumptions, and conventions used to specify values and data types throughout the library.

9.13 Edge

Shape edge finishing tools.

Collaboration diagram for Edge:



Files

- file tools_edge.scad

  Shape edge finishing tools.

Functions

- module edge_profile_r (r, p=0, f=1, a=90,)

  A 2d edge-finish profile specified by intersection radius.
- module edge_add_r (r, l=1, p=0, f=1, m=3, ba=45, a1=0, a2=90, center=false)

  A 3d edge-finish additive shape specified by intersection radius.

9.13.1 Detailed Description

Shape edge finishing tools.

9.13.2 Function Documentation

9.13.2.1 module edge_add_r (r , l = 1, p = 0, f = 1, m = 3, ba = 45, a1 = 0, a2 = 90, center = false)

A 3d edge-finish additive shape specified by intersection radius.

Parameters

r	<decimal> The radius length.
l	<decimal> The edge length.
p	<integer> The profile identifier.
f	<decimal> The mid-point offset factor.
m	<integer> The end finish mode: (B0: bottom, B1: top).
ba	<decimal> The end bevel angle.

a1	<decimal> The edge plane start angle.
a2	<decimal> The edge plane end angle.
center	<boolean> Center length about origin.

Example

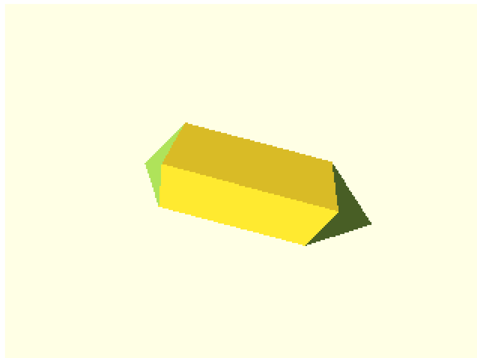


Figure 45: edge_add_r

```
rotate([90,-90,90]) edge_add_r( r=5, l=20, f=5/8, center=true );
```

m	B1	B0	Description
0	0	0	cut bottom (-z) and top (+z)
1	0	1	bevel bottom and cut top
2	1	0	cut bottom and bevel top
3	1	1	bevel bottom and top

See edge_profile_r() for description of available profiles.

Definition at line 208 of file tools_edge.scad.

9.13.2.2 module edge_profile_r (r, p = 0, f = 1, a = 90)

A 2d edge-finish profile specified by intersection radius.

Parameters

r	<decimal> The radius length.
p	<integer> The profile identifier.
f	<decimal> The mid-point offset factor.
a	<decimal> The sweep angle.

Example

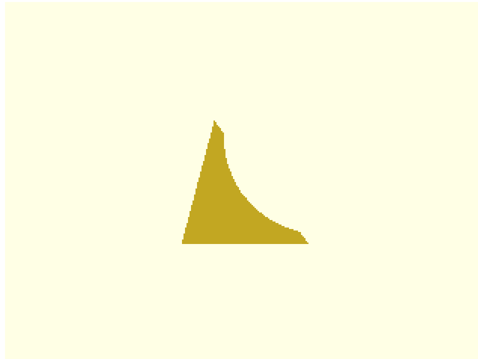


Figure 46: edge_profile_r

```
edge_profile_r( r=5, p=1, f=1+10/100, a=75 );
```

Profiles:

p	Description
0	Two segment bevel with mid inflection
1	A cove with cut-out offset
2	A quarter round with offset

Note

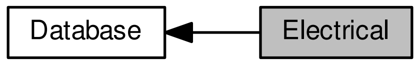
An offset factor greater than 1 moves the mid-point away from the profile edge-vertex. A factor less than 1 move it inwards towards the edge-vertex.

Definition at line 114 of file tools_edge.scad.

9.14 Electrical

Electrical specifications.

Collaboration diagram for Electrical:

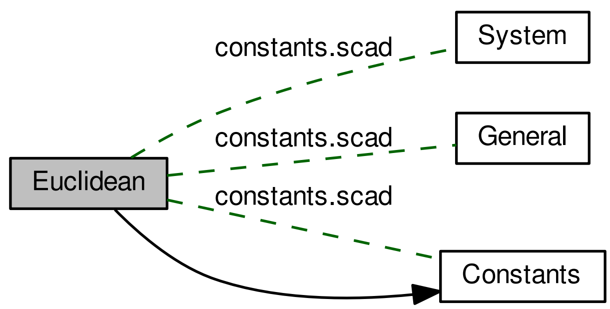


Electrical specifications.

9.15 Euclidean

Euclidean space axis mapping.

Collaboration diagram for Euclidean:



Files

- file constants.scad

  Design constant definitions.

Variables

- x_axis_ci = 0

  <integer> The coordinate axis index for the Euclidean space x-axis.
- y_axis_ci = 1

  <integer> The coordinate axis index for the Euclidean space y-axis.
- z_axis_ci = 2

  <integer> The coordinate axis index for the Euclidean space z-axis.
- zero2d = [0, 0]

  <decimal-list-2> A 2d zero vector (a list with two zeros).
- origin2d = [0, 0]

  <point-2d> The origin point coordinate in 2d Euclidean space.
- x_axis2d_uv = [1, 0]

  <vector-2d> The unit vector of the positive x-axis in 2d Euclidean space.
- y_axis2d_uv = [0, 1]

  <vector-2d> The unit vector of the positive y-axis in 2d Euclidean space.
- x_axis2d_ul = [-x_axis2d_uv, +x_axis2d_uv]

  <line-2d> A positively-directed unit line centered on the x-axis in 2d Euclidean space.
- y_axis2d_ul = [-y_axis2d_uv, +y_axis2d_uv]

  <line-2d> A positively-directed unit line centered on the y-axis in 2d Euclidean space.
- zero3d = [0, 0, 0]

<decimal-list-2> A 3d zero vector (a list with three zeros).

- origin3d = [0, 0, 0]

  <point-3d> The origin point coordinate in 3-dimensional Euclidean space.
- x_axis3d_uv = [1, 0, 0]

  <vector-3d> The unit vector of the positive x-axis in 3d Euclidean space.
- y_axis3d_uv = [0, 1, 0]

  <vector-3d> The unit vector of the positive y-axis in 3d Euclidean space.
- z_axis3d_uv = [0, 0, 1]

  <vector-3d> The unit vector of the positive z-axis in 3d Euclidean space.
- x_axis3d_ul = [-x_axis3d_uv, +x_axis3d_uv]

  <line-3d> A positively-directed unit line centered on the x-axis in 3d Euclidean space.
- y_axis3d_ul = [-y_axis3d_uv, +y_axis3d_uv]

  <line-3d> A positively-directed unit line centered on the y-axis in 3d Euclidean space.
- z_axis3d_ul = [-z_axis3d_uv, +z_axis3d_uv]

  <line-3d> A positively-directed unit line centered on the z-axis in 3d Euclidean space.
- xy_plane_on = [origin3d, z_axis3d_uv]

  <plane> The right-handed xy plane centered at the origin with normal vector.
- yz_plane_on = [origin3d, x_axis3d_uv]

  <plane> The right-handed yz plane centered at the origin with normal vector.
- zx_plane_on = [origin3d, y_axis3d_uv]

  <plane> The right-handed zx plane centered at the origin with normal vector.
- xy_plane_os = [origin3d, [for (r=[[1,1],[1,-1],[-1,-1],[-1,1]]) [r[0],r[1],0]]]

  <plane> The right-handed xy plane centered at the origin with coplanar unit square points.
- yz_plane_os = [origin3d, [for (r=[[1,1],[1,-1],[-1,-1],[-1,1]]) [0,r[0],r[1]]]]

  <plane> The right-handed yz plane centered at the origin with coplanar unit square points.
- zx_plane_os = [origin3d, [for (r=[[1,1],[1,-1],[-1,-1],[-1,1]]) [r[1],0,r[0]]]]

  <plane> The right-handed zx plane centered at the origin with coplanar unit square points.

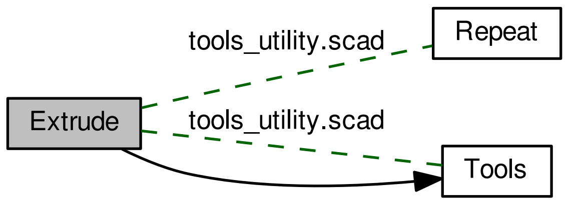
9.15.1 Detailed Description

Euclidean space axis mapping.

9.16 Extrude

Shape extrusion tools.

Collaboration diagram for Extrude:



Files

- file tools_utility.scad

    Shape transformation utility tools.

Functions

- module rotate_extrude_tr (r, pa=0, ra=360, profile=false)

    Translate, rotate, and revolve the 2d shape about the z-axis.
- module rotate_extrude_tre (r, l, pa=0, ra=360, m=255, profile=false)

    Translate, rotate, and revolve the 2d shape about the z-axis with linear elongation.
- module linear_extrude_uls (h, center=false)

    Linearly extrude 2d shape with extrusion upper and lower scaling.

9.16.1 Detailed Description

Shape extrusion tools.

9.16.2 Function Documentation

9.16.2.1 module linear_extrude_uls (h , center = `false`)

Linearly extrude 2d shape with extrusion upper and lower scaling.

Parameters

h	<decimal-list-3:9|decimal> A list of decimals or a single decimal to specify simple extrusion height.

center	<boolean> Center extrusion about origin.

When `h` is a decimal, the shape is extruded linearly as normal. To scale the upper and lower slices of the extrusion, `h` must be assigned a list with a minimum of three decimal values as described in the following table.

sym	h[n]	default	description
h	0		total extrusion height
n1	1		(+z) number of scaled extrusion slices
h1	2		(+z) extrusion scale percentage
x1	3	-h1	(+z) x-dimension scale percentage
y1	4	x1	(+z) y-dimension scale percentage
n2	5	n1	(-z) number of scaled extrusion slices
h2	6	h1	(-z) extrusion scale percentage
x2	7	x1	(-z) x-dimension scale percentage
y2	8	y1	(-z) y-dimension scale percentage

Example

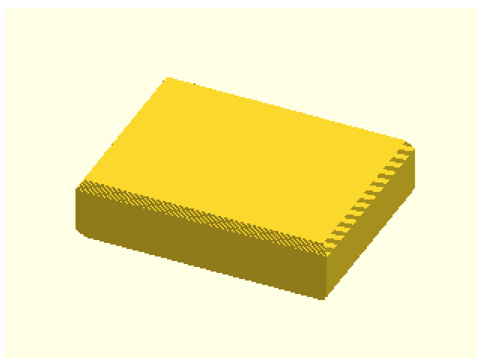


Figure 47: linear_extrude_uls

```
linear_extrude_uls( [5,10,15,-5], center=true ) square( [20,15], center=true );
```

Note

When symmetrical scaling is desired, shape must be centered about origin.

Todo This function should be rewritten to use the built-in scaling provided by linear_extrude() in the upper and lower scaling zones.

Definition at line 233 of file tools_utility.scad.

9.16.2.2 module rotate_extrude_tr (r , pa = `0`, ra = `360`, profile = `false`)

Translate, rotate, and revolve the 2d shape about the z-axis.

Parameters

r	<decimal> The rotation radius.
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
profile	<boolean> Show profile only (do not extrude).

Example

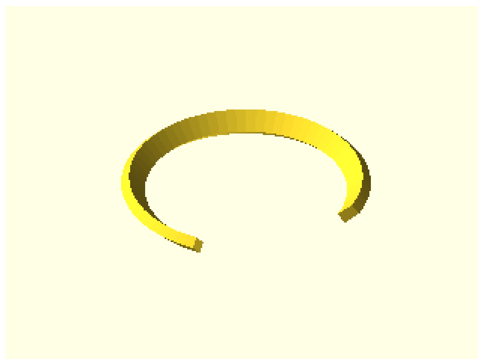


Figure 48: rotate_extrude_tr

```
rotate_extrude_tr( r=50, pa=45, ra=270 ) square( [10,5], center=true );
```

Definition at line 103 of file tools_utility.scad.

9.16.2.3 module rotate_extrude_tre (r , l , pa = 0, ra = 360, m = 255, profile = false)

Translate, rotate, and revolve the 2d shape about the z-axis with linear elongation.

Parameters

r	<decimal> The rotation radius.
l	<decimal-list-2|decimal> The elongation length. A list [x, y] of decimals or a single decimal for (x=y)
pa	<decimal> The profile pitch angle in degrees.
ra	<decimal> The rotation sweep angle in degrees.
m	<integer> The section render mode. An 8-bit encoded integer that indicates the revolution sections to render. Bit values **1** enables the corresponding section and bit values **0** are disabled. Sections are assigned to the bit position in counter-clockwise order.
profile	<boolean> Show profile only (do not extrude).

Example

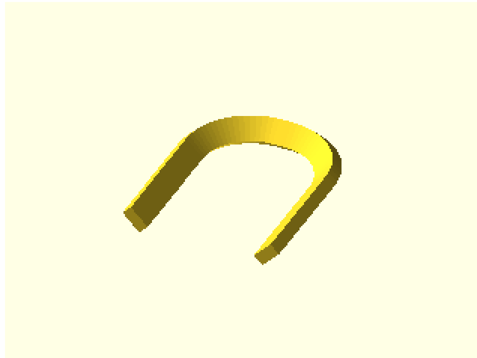


Figure 49: rotate_extrude_tre

```
rotate_extrude_tre( r=25, l=[5, 50], pa=45, m=31 ) square( [10,5], center=true );
```

Note

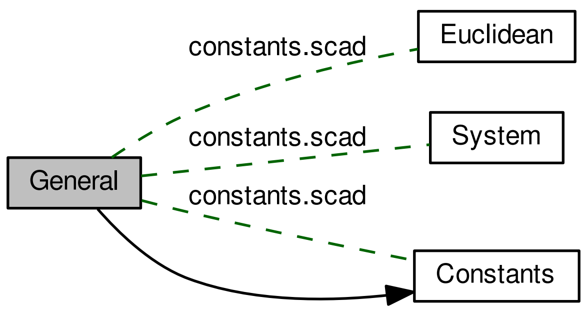
When elongating `(l > 0)`, `ra` is ignored. However, `m` may be used to control which complete revolution section to render.

Definition at line 139 of file tools_utility.scad.

9.17 General

General design constants.

Collaboration diagram for General:



Files

- file constants.scad

  Design constant definitions.

Variables

- aeps = 0.001

  <decimal> Epsilon, small distance to deal with overlapping shapes.
- pi = 3.14159265358979323

  <decimal> The ratio of a circle's circumference to its diameter.
- tau = 2∗pi

  <decimal> The ratio of a circle's circumference to its radius.
- phi = (1 + sqrt(5)) / 2

  <decimal> The golden ratio.

9.17.1 Detailed Description

General design constants.

9.18 Geometry

Predefined geometry.

Collaboration diagram for Geometry:

Modules

- Polyhedra
  Tables of polyhedra vertices, faces, and edges.

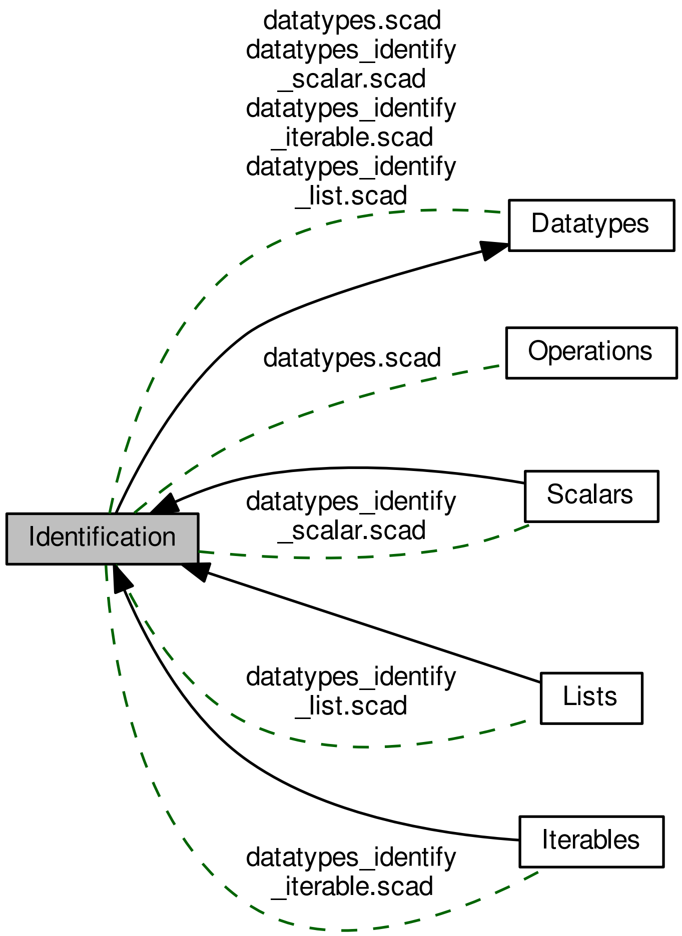
9.18.1 Detailed Description

Predefined geometry.

9.19 Identification

Compile-time data type identification and tests.

Collaboration diagram for Identification:



Modules

- Iterables

    Iterable data type identification.
- Lists

    List data type identification.
- Scalars

    Scalar data type identification.

Files

- file datatypes.scad

  Data type identification and operations.
- file datatypes_identify_scalar.scad

  Scalar data type identification.
- file datatypes_identify_iterable.scad

  Iterable data type identification.
- file datatypes_identify_list.scad

  List data type identification.

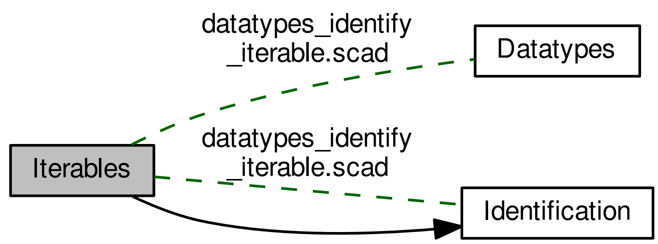
9.19.1 Detailed Description

Compile-time data type identification and tests.

9.20 Iterables

Iterable data type identification.

Collaboration diagram for Iterables:



Files

- file datatypes_identify_iterable.scad

  Iterable data type identification.

Functions

- function all_equal (v, cv)

  Test if a list of values equal a comparison value.
- function any_equal (v, cv)

  Test if any element of a list of values equal a comparison value.
- function all_defined (v)

  Test if no element of a list of values is undefined.
- function any_undefined (v)

  Test if any element of a list of values is undefined.
- function all_scalars (v)

  Test if all elements of a list of values are scalars.
- function all_lists (v)

  Test if all elements of a list of values are lists.
- function all_strings (v)

  Test if all elements of a list of values are strings.
- function all_numbers (v)

  Test if all elements of a list of values are numbers.
- function all_len (v, l)

  Test if all elements of a list of values are lists of a specified length.

9.20.1 Detailed Description

Iterable data type identification.

See validation results.

9.20.2 Function Documentation

9.20.2.1 function all_defined (v)

Test if no element of a list of values is undefined.

Parameters

v	<list> A list of values.

Returns

<boolean> **true** when no element is undefined and **false** otherwise.

Warning

Always returns **true** when the list is empty.

9.20.2.2 function all_equal (v, cv)

Test if a list of values equal a comparison value.

Parameters

v	<list> A list of values.
cv	<value> A comparison value.

Returns

<boolean> **true** when all elements equal the value `cv` and **false** otherwise.

Warning

Always returns **true** when the list is empty.

9.20.2.3 function all_len (v, l)

Test if all elements of a list of values are lists of a specified length.

Parameters

v	<list> A list of values.
l	<integer> The test length.

Returns

<boolean> **true** when all elements are lists of the specified length and **false** otherwise. Returns **true** when the list is a single list of length `l`.

Warning

Always returns **true** when `v` is empty.

9.20.2.4 function all_lists (v)

Test if all elements of a list of values are lists.

Parameters

v	<list> A list of values.

Returns

<boolean> **true** when all elements are lists and **false** otherwise. Returns **true** when the list is a single list value.

Warning

Always returns **true** when the list is empty.

9.20.2.5 function all_numbers (v)

Test if all elements of a list of values are numbers.

Parameters

v	<list> A list of values.

Returns

<boolean> **true** when all elements are numerical values and **false** otherwise. Returns **true** when the list is a single numerical value.

Warning

Always returns **true** when the list is empty.

9.20.2.6 function all_scalars (v)

Test if all elements of a list of values are scalars.

Parameters

v	<list> A list of values.

Returns

<boolean> **true** when all elements are scalar values and **false** otherwise. Returns **true** when the list is a single scalar value.

Warning

Always returns **true** when the list is empty.

9.20.2.7 function all_strings (v)

Test if all elements of a list of values are strings.

Parameters

v	<list> A list of values.

Returns

<boolean> **true** when all elements are string values and **false** otherwise. Returns **true** when the list is a single string value.

Warning

Always returns **true** when the list is empty.

9.20.2.8 function any_equal (v , cv)

Test if any element of a list of values equal a comparison value.

Parameters

v	<list> A list of values.
cv	<value> A comparison value.

Returns

<boolean> **true** when any element equals the value `cv` and **false** otherwise.

Warning

Always returns **false** when the list is empty.

9.20.2.9 function any_undefined (v)

Test if any element of a list of values is undefined.

Parameters

v	<list> A list of values.

Returns

<boolean> **true** when any element is undefined and **false** otherwise.

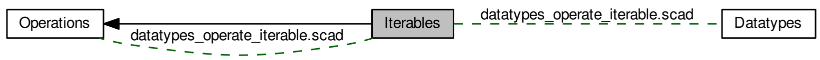
Warning

Always returns **false** when the list is empty.

9.21 Iterables

Iterable data type operations.

Collaboration diagram for Iterables:



Files

- file datatypes_operate_iterable.scad

  Iterable data type operations.

Functions

- function edefined_or (v, i, d)

  Return an iterable element when it exists or a default value when it does not.
- function find (mv, v, c=1, i, i1=0, i2)

  Find the occurrences of a match value in an iterable value.
- function count (mv, v, s=true, i)

  Count all occurrences of a match value in an iterable value.
- function exists (mv, v, s=true, i)

  Check for the existence of a match value in an iterable value.
- function first (v)

  Return the first element of an iterable value.
- function second (v)

  Return the second element of an iterable value.
- function third (v)

  Return the third element of an iterable value.
- function last (v)

  Return the last element of an iterable value.
- function nfirst (v, n=1)

  Return a list containing the first n elements of an iterable value.
- function nlast (v, n=1)

  Return a list containing the last n elements of an iterable value.
- function nhead (v, n=1)

  Return a list containing all but the last n elements of an iterable value.
- function ntail (v, n=1)

  Return a list containing all but the first n elements of an iterable value.
- function reverse (v)

  Reverse the elements of an iterable value.
- function rselect (v, i)

  Select a range of elements from an iterable value.

- function nssequence (v, n=1, s=1, w=false)

  Return a list of all n-element sequential-subsets of an iterable value.
- function eappend (nv, v, r=true, j=true, l=true)

  Append a value to each element of an iterable value.
- function insert (nv, v, i=0, mv, mi=0, s=true, si)

  Insert a new value into an iterable value.
- function delete (v, i, mv, mc=1, s=true, si)

  Delete elements from an iterable value.
- function strip (v, mv=empty_lst)

  Strip all matching values from an iterable value.
- function unique (v)

  Return the unique elements of an iterable value.

9.21.1 Detailed Description

Iterable data type operations.

See validation results.

9.21.2 Function Documentation

9.21.2.1 function count (mv , v , s = `true`, i)

Count all occurrences of a match value in an iterable value.

Parameters

mv	<value> A match value.
v	<value> An iterable value.
s	<boolean> Use `search` for element matching (assign **false** to use find()).
i	<integer> The element index to consider for iterable elements.

Returns

<integer> The number of times `mv` occurs in the list.

9.21.2.2 function delete (v , i , mv , mc = `1`, s = `true`, si)

Delete elements from an iterable value.

Parameters

v	<value> An iterable value.
i	<range|list|integer> Deletion Indexes.
mv	<list|string|value> Match value candidates.
mc	<integer> A match count. For `(mc>=1)`, remove the first `mc` matches. For `(mc<=0)`, remove all matches.

s	<boolean> Use `search` for element matching (assign **false** to use find()).
si	<integer> The element column index when matching.

Returns

<list> A list with all specified elements removed. Returns **undef** when `i` does not map to an element of `v`. Returns **undef** when `v` is not defined or is not iterable.

The elements to delete can be specified by an index position, a list of index positions, an index range, an element match value, or a list of element match values (when using `search`). When `mv` is a list of match values, all values of `mv` that exists in `v` are candidates for deletion. For each matching candidate, `mc` indicates the quantity to remove. When more than one deletion criteria is specified, the order of precedence is: `mv`, `i`.

9.21.2.3 function eappend (nv , v , r = `true`, j = `true`, l = `true`)

Append a value to each element of an iterable value.

Parameters

nv	<value> A new value to append.
v	<value> An iterable value.
r	<boolean> Reduce list element value before appending.
j	<boolean> Join each appendage as a separate list.
l	<boolean> Append new value to last element.

Returns

<list> A list with `nv` appended to each element of `v`. Returns **undef** when `v` is not defined or is not iterable.

Example

```
v1=[["a"], ["b"], ["c"], ["d"]];
v2=[1, 2, 3];

echo( eappend( v2, v1 ) );
echo( eappend( v2, v1, r=false ) );
echo( eappend( v2, v1, j=false, l=false ) );
```

Result

```
ECHO: [["a", 1, 2, 3], ["b", 1, 2, 3], ["c", 1, 2, 3], ["d", 1, 2, 3]]
ECHO: [[["a"], 1, 2, 3], [["b"], 1, 2, 3], [["c"], 1, 2, 3], [["d"], 1, 2, 3]]
ECHO: ["a", 1, 2, 3, "b", 1, 2, 3, "c", 1, 2, 3, "d"]
```

Note

Appending with reduction causes `nv` to be appended to the *elements* of each iterable value. Otherwise, `nv` is appended to the iterable value itself.

9.21.2.4 function edefined_or (v , i , d)

Return an iterable element when it exists or a default value when it does not.

Parameters

v	<value> An iterable value.
i	<integer> An element index.
d	<value> A default value.

Returns

<value> `v[i]` when it is defined or `d` otherwise.

9.21.2.5 function exists (mv , v , s = `true`, i)

Check for the existence of a match value in an iterable value.

Parameters

mv	<value> A match value.
v	<value> An iterable value.
s	<boolean> Use `search` for element matching (assign **false** to use find()).
i	<integer> The element index to consider for iterable elements.

Returns

<boolean> **true** when `mv` exists in the list and **false** otherwise.

9.21.2.6 function find (mv , v , c = `1`, i , i1 = `0`, i2)

Find the occurrences of a match value in an iterable value.

Parameters

mv	<value> A match value.
v	<value> An iterable value.
c	<integer> A match count. For `(c>=1)`, return the first `c` matches. For `(c<=0)`, return all matches.
i	<integer> The element index to consider for iterable elements.
i1	<integer> The element index where find begins (default: first).
i2	<integer> The element index where find ends (default: last).

Returns

<list> A list of indexes where elements match `mv`. Returns **empty_lst** when no element of `v` matches `mv` or when `v` is not iterable.

The use-cases for find() and `search()` are summarized in the following tables.

Find:

mv / v	string	list of scalars	list of iterables
scalar		(a)	(b) see note 1
string	(c)		(b) see note 1
list of scalars			(b) see note 1

list of iterables			(b) see note 1

Search:

mv / v	string	list of scalars	list of iterables
scalar		(a)	(b)
string	(d)	invalid	(e) see note 2
list of scalars		(f)	(g)
list of iterables			(g)

Key:

- (a) Identify each element of `v` that equals `mv`.
- (b) Identify each element of `v` where `mv` equals the element at the specified column index, `i`, of each iterable value in `v`.
- (c) If, and only if, `mv` is a single character, identify each character in `v` that equals `mv`.
- (d) For each character of `mv`, identify where it exists in `v`. **empty_lst** is returned for each character of `mv` absent from `v`.
- (e) For each character of `mv`, identify where it exists in `v` either as a numeric value or as a character at the specified column index, `i`. **empty_lst** is returned for each character of `mv` absent from `v`.
- (f) For each scalar of `mv`, identify where it exists in `v`. **empty_lst** is returned for each scalar of `mv` absent from `v`.
- (g) For each element of `mv`, identify where it equals the element at the specified column index, `i`, of each iterable value in `v`. **empty_lst** is returned for each element of `mv` absent from `v` in the specified column index.

Note

1: When `i` is specified, that element column is compared. Otherwise, the entire element is compared. Functions find() and `search()` behave differently in this regard.
2: Invalid use combination when any element of `v` is a string. However, an element that is a list of one or more strings is valid. In which case, only the first character of each string element is considered.

9.21.2.7 function first (v)

Return the first element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The first element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

Note

Value may also be a range.

9.21.2.8 function insert (nv , v , i = 0, mv , mi = 0, s = true, si)

Insert a new value into an iterable value.

Parameters

nv	<value> A new value to insert.
v	<value> An iterable value.
i	<integer> An insert position index.
mv	<list|string|value> Match value candidates.
mi	<integer> A match index.
s	<boolean> Use `search` for element matching (assign **false** to use find()).
si	<integer> The element column index when matching.

Returns

<list> A list with `nv` inserted into `v` at the specified position. Returns **undef** when no value of `mv` exists in `v`. Returns **undef** when `(mi + 1)` exceeds the matched element count. Returns **undef** when `i` does not map to an element of `v`. Returns **undef** when `v` is not defined or is not iterable.

The insert position can be specified by an index, an element match value, or list of potential match values (when using `search`). When multiple matches exists, `mi` indicates the insert position. When more than one insert position criteria is specified, the order of precedence is: `mv`, `i`.

9.21.2.9 function last (v)

Return the last element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The last element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

9.21.2.10 function nfirst (v, n = 1)

Return a list containing the first n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> The element count.

Returns

<list> A list containing the first `n` elements of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

9.21.2.11 function nhead (v, n = 1)

Return a list containing all but the last n elements of an iterable value.

Parameters

v	<value> An iterable value.

n	<integer> The element count.

Returns

<list> A list containing all but the last `n` elements of `v`. Returns **empty_lst** when `v` contains fewer than `n` elements. Returns **undef** when `v` is not defined, is not iterable, or is empty.

9.21.2.12 function nlast (v , n = 1)

Return a list containing the last n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> The element count.

Returns

<list> A list containing the last `n` elements of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

9.21.2.13 function nssequence (v , n = 1, s = 1, w = false)

Return a list of all n-element sequential-subsets of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> The number of elements for each subset.
s	<integer> The iteration step size.
w	<boolean> Use wrap-at-end circular subset selection.

Returns

<list-list> A list of all n-element sequential-subsets of `v` skipping `s` elements between each subset selection. Returns **empty_lst** when `v` is empty, is not defined or is not iterable.

Example

```
v = [1, 2, 3, 4];

nssequence( v, 3, 1, false ); // [ [1,2,3], [2,3,4] ]
nssequence( v, 3, 1, true );  // [ [1,2,3], [2,3,4], [3,4,1], [4,1,2] ]
```

9.21.2.14 function ntail (v , n = 1)

Return a list containing all but the first n elements of an iterable value.

Parameters

v	<value> An iterable value.
n	<integer> The element count.

Returns

<list> A list containing all but the first n elements of `v`. Returns **empty_lst** when `v` contains fewer than `n` elements. Returns **undef** when `v` is not defined, is not iterable, or is empty.

9.21.2.15 function reverse (v)

Reverse the elements of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<list> A list containing the elements of `v` in reversed order. Returns **empty_lst** when `v` is empty. Returns **undef** when `v` is not defined or is not iterable.

9.21.2.16 function rselect (v , i)

Select a range of elements from an iterable value.

Parameters

v	<value> An iterable value.
i	<range|list|integer> The index selection.

Returns

<list> A list containing the identified element(s). Returns **undef** when `i` does not map to an element of `v`. Returns **empty_lst** when `v` is empty. Returns **undef** when `v` is not defined or is not iterable.

9.21.2.17 function second (v)

Return the second element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The second element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

Note

Value may also be a range.

9.21.2.18 function strip (v , mv = empty_lst)

Strip all matching values from an iterable value.

Parameters

v	<value> An iterable value.
mv	<value> The match value.

Returns

<list> A list with all elements equal to `mv` removed. Returns **undef** when `v` is not defined or is not iterable.

9.21.2.19 function third (v)

Return the third element of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<value> The second element of `v`. Returns **undef** when `v` is not defined, is not iterable, or is empty.

Note

Value may also be a range.

9.21.2.20 function unique (v)

Return the unique elements of an iterable value.

Parameters

v	<value> An iterable value.

Returns

<list> A list of unique elements with order preserved. Returns **undef** when `v` is not defined or is not iterable.

9.22 Lengths

Length units and conversions.

Collaboration diagram for Lengths:

Files

- file units_length.scad

  Length units and conversions.

Functions

- function unit_length_name (u=base_unit_length, d=1, w=false)

  Return the name for a length unit identifier with dimension.
- function convert_length (v, from=base_unit_length, to=base_unit_length, d=1)

  Convert a value from from one units to another with dimensions.

Variables

- base_unit_length = "mm"

  <string> The base unit for length measurements.

9.22.1 Detailed Description

Length units and conversions.

These functions allow for lengths to be specified with units. Lengths specified with units are independent of (base_unit_length). There are also unit conversion functions for converting from one unit to another.

The table below enumerates the supported units.

units id	description
pm	picometer
nm	nanometer
um	micrometer
mm	millimeter
cm	centimeter

dm	decimeter
m	meter
km	kilometer
thou, mil	thousandth of an inch
in	inch
ft	feet
yd	yard
mi	mile

Example

```
include <units/units_length.scad>;

base_unit_length = "mm";

// get base unit name
un = unit_length_name();

// absolute length measurements in base unit.
c1 = convert_length(1/8, "in");
c2 = convert_length(3.175, "mm");
c3 = convert_length(25, "mil");
c4 = convert_length(1, "ft", d=3);

// convert between units.
c5 = convert_length(10, from="mil", to="in");
c6 = convert_length(10, from="ft", to="mm");
```

Result (base_unit_length = **mm**):

```
ECHO: un = "millimeter"
ECHO: c1 = 3.175
ECHO: c2 = 3.175
ECHO: c3 = 0.635
ECHO: c4 = 2.83168e+07
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Result (base_unit_length = **cm**):

```
ECHO: un = "centimeter"
ECHO: c1 = 0.3175
ECHO: c2 = 0.3175
ECHO: c3 = 0.0635
ECHO: c4 = 28316.8
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Result (base_unit_length = **mil**):

```
ECHO: un = "thousandth"
ECHO: c1 = 125
ECHO: c2 = 125
ECHO: c3 = 25
ECHO: c4 = 1.728e+12
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Result (base_unit_length = **in**):

```
ECHO: un = "inch"
ECHO: c1 = 0.125
ECHO: c2 = 0.125
ECHO: c3 = 0.025
ECHO: c4 = 1728
ECHO: c5 = 0.01
ECHO: c6 = 3048
```

Example (equivalent lengths)

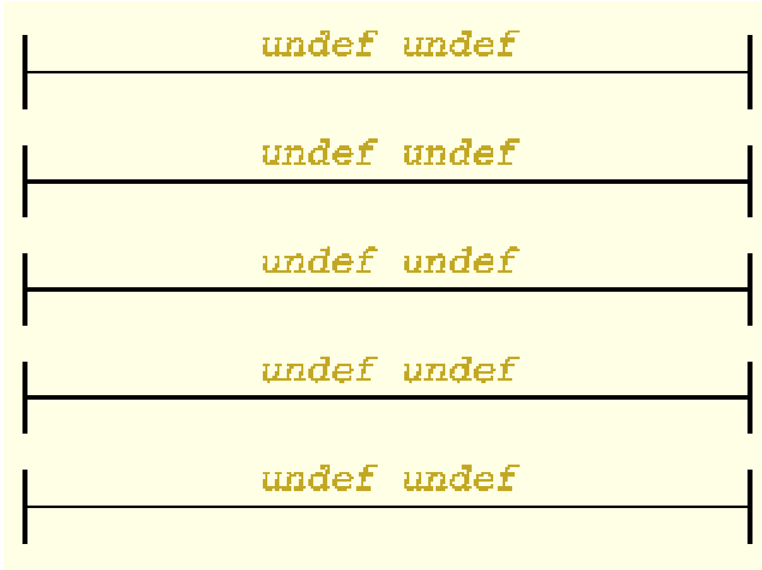


Figure 50: Unit Lengths

9.22.2 Function Documentation

9.22.2.1 function convert_length (v , from = **base_unit_length**, to = **base_unit_length**, d = 1)

Convert a value from from one units to another with dimensions.

Parameters

v	<decimal> A value to convert.
from	<string> The units of the value to be converted.
to	<string> A units to which the value should be converted.
d	<integer> A dimension. One of [1|2|3].

Returns

<decimal> The conversion result. Returns **undef** for identifiers or dimensions that are not defined.

9.22.2.2 function unit_length_name (u = **base_unit_length**, d = 1, w = false)

Return the name for a length unit identifier with dimension.

Parameters

w	<boolean> **true** for word and **false** for symbol format.
u	<string> A length unit identifier.
d	<integer> A dimension. One of [1|2|3].

Returns

<string> The long name for a length unit identifier with dimension. Returns **undef** for identifiers or dimensions that are not defined.

9.23 Linear Algebra

Linear algebra transformations on Euclidean coordinates.

Collaboration diagram for Linear Algebra:

Files

- file math_linalg.scad

  Linear algebra mathematical functions.

Functions

- function multmatrix_lp (c, m)

  Multiply all coordinates by a 4x4 3d-transformation matrix.
- function translate_lp (c, v)

  Translate all coordinates dimensions.
- function rotate_lp (c, a, v, o=origin3d)

  Rotate all coordinates about one or more axes in Euclidean 2d or 3d space.
- function scale_lp (c, v)

  Scale all coordinates dimensions.
- function resize_lp (c, v)

  Scale all coordinates dimensions proportionately to fit inside a region.

9.23.1 Detailed Description

Linear algebra transformations on Euclidean coordinates.

9.23.2 Function Documentation

9.23.2.1 function multmatrix_lp (c , m)

Multiply all coordinates by a 4x4 3d-transformation matrix.

Parameters

c	<coords-3d> A list of 3d coordinate points.
m	<matrix-4x4> A 4x4 transformation matrix (decimal-list-4-list4).

Returns

<coords-3d> A list of 3d coordinate points multiplied by the transformation matrix.

See `Wikipedia` and `multmatrix` for more information.

9.23.2.2 function resize_lp (c , v)

Scale all coordinates dimensions proportionately to fit inside a region.

Parameters

c	<coords-nd> A list of nd coordinate points.
v	<decimal-list-n> A list of bounds for each dimension.

Returns

<coords-nd> A list of proportionately scaled coordinate points which exactly fit the region bounds `v`.

9.23.2.3 function rotate_lp (c , a , v , o = origin3d)

Rotate all coordinates about one or more axes in Euclidean 2d or 3d space.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
a	<decimal-list-3|decimal> The axis rotation angle. A list [ax, ay, az] or a single decimal to specify az only.
v	<vector-3d> An arbitrary axis for the rotation. When specified, the rotation angle will be `a` or az about the line `v` that passes through point `o`.
o	<point-3d> A 3d point origin for the rotation. Ignored when `v` is not specified.

Returns

<coords-3d|coords-2d> A list of 3d or 2d rotated coordinates. Rotation order is rz, ry, rx.

See `Wikipedia` for more information on `transformation matrix` and `axis rotation`.

9.23.2.4 function scale_lp (c , v)

Scale all coordinates dimensions.

Parameters

c	<coords-nd> A list of nd coordinate points.
v	<decimal-list-n> A list of scalers for each dimension.

Returns

<coords-nd> A list of scaled coordinate points.

9.23.2.5 function translate_lp (c , v)

Translate all coordinates dimensions.

Parameters

c	<coords-nd> A list of nd coordinate points.
v	<decimal-list-n> A list of translations for each dimension.

Returns

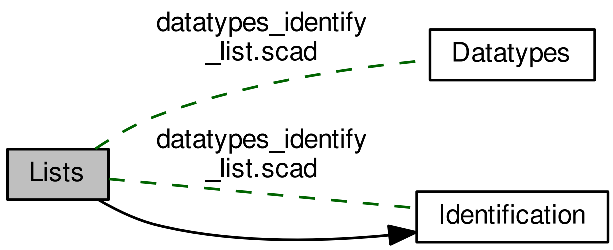
<coords-nd> A list of translated coordinate points.

See `Wikipedia` for more information and `transformation matrix`.

9.24 Lists

List data type identification.

Collaboration diagram for Lists:



Files

- file datatypes_identify_list.scad

    List data type identification.

Functions

- function n_almost_equal (v1, v2, p=6)

    Test if all elements of two lists of numbers are sufficiently equal.
- function almost_equal (v1, v2, p=6)

    Test if all numerical elements of two lists of values are sufficiently equal.
- function compare (v1, v2, s=true)

    Order to lists of arbitrary values.

9.24.1 Detailed Description

List data type identification.

See validation results.

9.24.2 Function Documentation

9.24.2.1 function almost_equal (v1 , v2 , p = 6)

Test if all numerical elements of two lists of values are sufficiently equal.

Parameters

v1	<list> A list of values 1.
v2	<list> A list of values 2.
p	<number> The numerical precision.

Returns

<boolean> **true** when all elements of each lists are sufficiently equal and **false** otherwise.

The 'distance' between two numbers must be less than pow(10,-p) to be considered almost equal. All numerical comparisons are performed to the specified precision. All non-numeric comparisons test for equality.

Note

If the lists are scalar numbers, the function n_almost_equal() provides a more efficient test.

Warning

Always returns **true** when both lists are empty.

9.24.2.2 function compare (v1 , v2 , s = `true`)

Order to lists of arbitrary values.

Parameters

v1	<list> A list of values 1.
v2	<list> A list of values 2.
s	<boolean> Order ranges by their numerical sum.

Returns

<integer> **-1** when `(v2 < v1)`, **+1** when `(v2 > v1)`, and **0** when `(v2 == v1)`.

The following table summarizes how values are ordered.

order	type	s	intra-type ordering
1	**undef**		(singular)
2	number		numerical comparison
3	boolean		**false** < **true**
4	string		lexical comparison
5	list		lengths then element-wise comparison
6	range	**true**	compare sum of range elements
6	range	**false**	lengths then element-wise comparison

Note

When comparing two lists of equal length, the comparison continue element-by-element until an ordering can be determined. Two lists are equal when all elements have been compared and no ordering has been determined.

Warning

The performance of element-wise comparisons of lists degrades with list size.
The sum of a range may exceeded the intermediate variable storage capacity for long ranges.

9.24.2.3 function n_almost_equal (v1 , v2 , p = 6)

Test if all elements of two lists of numbers are sufficiently equal.

Parameters

v1	<number-list> A list of numbers 1.
v2	<number-list> A list of numbers 2.
p	<number> The numerical precision.

Returns

<boolean> **true** when the distance between `v1` and `v2` is less than `d` and **false** otherwise. Returns **false** when either list contains a non-numerica values, or when the lists are not of the same length.

The 'distance' between two numbers must be less than pow(10,-p) to be considered almost equal.

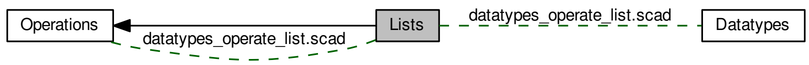
Note

To compare general lists of values see almost_equal().

9.25 Lists

List data type operations.

Collaboration diagram for Lists:



Files

- file datatypes_operate_list.scad

    List data type operations.

Functions

- function lstr (v)

    Convert a list of values to a concatenated string.
- function lstr_html (v, b, p, a, f, d=false)

    Convert a list of values to a concatenated HTML-formatted string.
- function consts (l, v, u=false)

    Create a sequence of constant or incrementing elements.
- function get_index (l, s=true, rs)

    Create a sequence for a list index sequence specification.
- function pad (l, w, p=0, r=true)

    Pad a list to a constant width of elements.
- function dround (v, d=6)

    Round all numerical values of a list to a fixed number of decimal point digits.
- function sround (v, d=6)

    Round all numerical values of a list to a fixed number of significant figures.
- function limit (v, l, u)

    Limit all numerical values of a list between an upper and lower bounds.
- function sum (v, i1, i2)

    Compute the sum of a list of numbers.
- function mean (v)

    Compute the mean/average of a list of numbers.
- function ciselect (v, i)

    Case-like select a value from a list of ordered value options.
- function cmvselect (v, mv)

    Case-like select a value from a list of mapped key-value options.
- function eselect (v, f=true, l=false, i)

    Select a specified element from each iterable value of a list.
- function smerge (v, r=false)

    Serial-merge lists of iterable values.

- function pmerge (v, j=true)

  Parallel-merge lists of iterable values.
- function qsort (v, i, r=false)

  Sort the numeric or string elements of a list using quick sort.
- function qsort2 (v, i, d=0, r=false, s=true)

  Hierarchically sort an arbitrary data list using quick sort.

9.25.1 Detailed Description

List data type operations.

See validation results.

9.25.2 Function Documentation

9.25.2.1 function ciselect (v , i)

Case-like select a value from a list of ordered value options.

Parameters

v	<list> A list of values.
i	<integer> Element selection index.

Returns

<value> The value of the list element at the specified index. Returns the default value when `i` does not map to an element.

Behaves like a case statement for selecting values from a list of *ordered options*. The default value is: `last(v)`.

Example

```
ov = [ "value1", "value2", "default" ];

ciselect( ov );      // "default"
ciselect( ov, 4 );  // "default"
ciselect( ov, 0 );  // "value1"
```

9.25.2.2 function cmvselect (v , mv)

Case-like select a value from a list of mapped key-value options.

Parameters

v	<matrix-2xN> A matrix of N key-value mapped pairs [[key, value], ...].
mv	<value> Element selection key match value.

Returns

<value> The value from the map that matches the key `mv`. Returns the default value when `mv` does not match any of the element identifiers of `v` or when `mv` is undefined.

Behaves like a case statement for selecting values from a list of *mapped options*. The default value is↩ : `second(last(v))`.

Example

```
ov = [ [0,"value0"], ["a","value1"], ["b","value2"], ["c","default"] ];

cmvselect( ov );      // "default"
cmvselect( ov, "x" ); // "default"
cmvselect( ov, 0 );   // "value0"
cmvselect( ov, "b" ); // "value2"
```

9.25.2.3 function consts (l, v, u = `false`)

Create a sequence of constant or incrementing elements.

Parameters

l	<integer> The list length.
v	<value> The element value.
u	<boolean> Element values are **undef**.

Returns

<list> A list of `l` copies of the element. Returns **empty_lst** when `l` is not a number or if `(l < 1)`.

Note

When `v` is not specified and `u` is **false**, each element is assigned the value of its index position.

9.25.2.4 function dround (v, d = `6`)

Round all numerical values of a list to a fixed number of decimal point digits.

Parameters

v	<list> A list of values.
d	<integer> The (maximum) number of decimals.

Returns

<list> The list with all numeric values truncated to `d` decimal digits and rounded-up if the following digit is 5 or greater. Non-numeric values are unchanged.

9.25.2.5 function eselect (v, f = `true`, l = `false`, i)

Select a specified element from each iterable value of a list.

Parameters

v	<list> A list of iterable values.
f	<boolean> Select the first element.
l	<boolean> Select the last element.
i	<integer> Select a numeric element index position.

Returns

<list> A list containing the selected element of each iterable value of `v`. Returns **empty_lst** when `v` is empty. Returns **undef** when `v` is not defined or is not iterable.

Note

When more than one selection criteria is specified, the order of precedence is: `i`, `l`, `f`.

9.25.2.6 function get_index (l, s = `true`, rs)

Create a sequence for a list index sequence specification.

Parameters

l	<list> The list.
s	<index> The index sequence specification.
rs	<integer> An optional seed for random sequences.

Returns

<number-list> An index sequence based on the specification. Returns **empty_lst** for any `v` that does not fall into one of the specification forms.

See Index sequence for argument specification and conventions.

9.25.2.7 function limit (v, l, u)

Limit all numerical values of a list between an upper and lower bounds.

Parameters

v	<list> A list of values.
l	<number> The minimum value.
u	<number> The maximum value.

Returns

<list> The list with all numeric values limited to the range `[l : u]`. A value will be assigned `l` when it is less than `l` and `u` when it is greater than `u`. Non-numeric values are unchanged.

9.25.2.8 function lstr (v)

Convert a list of values to a concatenated string.

Parameters

v	<list> A list of values.

Returns

<string> Constructed by converting each element of the list to a string and concatenating together. Returns **undef** when the list is not defined.

Example

```
v1=["a", "b", "c", "d"];
v2=[1, 2, 3];

echo( lstr(concat(v1, v2)) );
```

Result

```
ECHO: "abcd123"
```

9.25.2.9 function lstr_html (v, b, p, a, f, d = `false`)

Convert a list of values to a concatenated HTML-formatted string.

Parameters

v	<list> A list of values.
b	<tag-list-list> A list of tag lists. *Unpaired* HTML tag(s) to add before the value.
p	<tag-list-list> A list of tag lists. *Paired* HTML tag(s) to enclose the value.
a	<tag-list-list> A list of tag lists. *Unpaired* HTML tag(s) to add after the value.
f	<attr-list-list> A list of tag attribute lists for `fs`, where `fs=["color","size","face"]` is the font tag to enclose the value. Not all attributes are required, but the order is significant.
d	<boolean> Debug. When **true** angle brackets are replaced with curly brackets to prevent console decoding.

Returns

<string> Constructed by converting each element of the list to a string with specified HTML markup and concatenating. Returns **undef** when the list is not defined.

When there are fewer tag lists in `b`, `p`, `a`, or `f`, than there are value elements in `v`, the last specified tag list is used for all subsequent value elements.

For a list of the *paired* and *unpaired* HTML tags supported by the console see: HTML subset.

Example

```
echo( lstr_html(v="bold text", p="b", d=true) );
echo( lstr_html(v=[1,"x",3], f=[["red",6,"helvetica"],undef,["blue",10,"courier"]], d=true) );

v = ["result", "=", "mc", "2"];
b = ["hr", undef];
p = ["i", undef, ["b", "i"], ["b","sup"]];
a = concat(consts(3, u=true), "hr");
f = [undef, ["red"], undef, ["blue",4]];

echo( lstr_html(v=v, b=b, p=p, a=a, f=f, d=true) );
```

Result

```
ECHO: "{b}bold text{/b}"
ECHO: "{font color="red" size="6" face="helvetica"}1{/font}x{font color="blue" size="10" face="courier"
      }3{/font}"
ECHO: "{hr}{i}result{/i}{font color="red"}={/font}{b}{i}mc{/i}{/b}{b}{sup}{font color="blue" size="4"
      }2{/font}{/sup}{/b}{hr}"
```

9.25.2.10 function mean (v)

Compute the mean/average of a list of numbers.

Parameters

v	<number-list|range> A list of numerical values or a range.

Returns

<number\|number-list> The sum divided by the number of elements. Returns 0 when the list is empty. Returns **undef** when list non-numerical.

See Wikipedia for more information.

9.25.2.11 function pad (l, w, p = 0, r = true)

Pad a list to a constant width of elements.

Parameters

l	<list> The list.
w	<integer> The padded width.
p	<value> The padding value.
r	<boolean> Use right padding (**false** for left).

Returns

<list> A list padded to `w` elements.

When the list has greater than `w` elements, the list is returned unchanged. The empty list, **empty_lst**, has zero elements. When `l` is a string, characters are counted as individual elements. Use function lstr() to join padded values back into a single string if desired.

9.25.2.12 function pmerge (v , j = `true`)

Parallel-merge lists of iterable values.

Parameters

v	<list> A list of iterable values.
j	<boolean> Join each merge as a separate list.

Returns

<list> A list containing the parallel-wise element concatenation of each iterable value in `v`. Returns **empty_lst** when any element value in `v` is empty. Returns **undef** when `v` is not defined or when any element value in `v` is not iterable.

Example

```
v1=["a", "b", "c", "d"];
v2=[1, 2, 3];

echo( pmerge( [v1, v2], true ) );
echo( pmerge( [v1, v2], false ) );
```

Result

```
ECHO: [["a", 1], ["b", 2], ["c", 3]]
ECHO: ["a", 1, "b", 2, "c", 3]
```

Note

The resulting list length will be limited by the iterable value with the shortest length.
A single string, although iterable, is treated as a merged unit.

9.25.2.13 function qsort (v , i , r = `false`)

Sort the numeric or string elements of a list using quick sort.

Parameters

v	<number-list|string-list> A list of values.
i	<integer> The sort column index for iterable elements.
r	<boolean> Reverse the sort order.

Returns

<list> A list with elements sorted in ascending order. Returns **undef** when `v` is not defined or is not a list.

Warning

This implementation relies on the comparison operators '<' and '>' which expect the operands to be either two scalar numbers or two strings. Therefore, this function will not correctly sort lists elements that are not numbers or strings. Elements with unknown order are placed at the end of the list.

See `Wikipedia` for more information.

9.25.2.14 function qsort2 (v , i , d = `0`, r = `false`, s = `true`)

Hierarchically sort an arbitrary data list using quick sort.

Parameters

v	<data> A list of values.
i	<integer> The sort column index for iterable elements.
d	<integer> The recursive sort depth.
r	<boolean> Reverse the sort order.
s	<boolean> Order ranges by their numerical sum.

Returns

<list> With all elements sorted in ascending order. Returns **undef** when `v` is not defined or is not a list.

Elements are ordered using compare(). See its documentation for a description of the parameter `s`. To recursively sort all elements, set `d` greater than, or equal to, the maximum level of hierarchy in `v`.

See `Wikipedia` for more information.

9.25.2.15 function smerge (v , r = `false`)

Serial-merge lists of iterable values.

Parameters

v	<list> A list of iterable values.
r	<boolean> Recursively merge elements that are iterable.

Returns

<list> A list containing the serial-wise element concatenation of each element in `v`. Returns **empty_lst** when `v` is empty. Returns **undef** when `v` is not defined.

Note

A single string, although iterable, is treated as a merged unit.

9.25.2.16 function sround (v, d = 6)

Round all numerical values of a list to a fixed number of significant figures.

Parameters

v	<list> A list of values.
d	<integer> The (maximum) number of significant figures.

Returns

<list> The list with all numeric values rounded-up to `d` significant figures. Non-numeric values are unchanged.

See `Wikipedia` for more information.

9.25.2.17 function sum (v , i1 , i2)

Compute the sum of a list of numbers.

Parameters

v	<number-list|range> A list of numerical values or a range.
i1	<integer> The element index at which to begin summation (first when not specified).
i2	<integer> The element index at which to end summation (last when not specified).

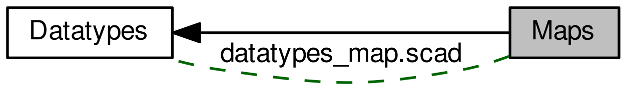
Returns

<number\|number-list> The sum over the index range. Returns 0 when `the` list is empty. Returns **undef** when list non-numerical.

9.26 Maps

Map data type operations.

Collaboration diagram for Maps:



Files

- file datatypes_map.scad

  Map data type operations.

Functions

- function get_map_i (m, k)

  Return the index of a map key.
- function map_exists (m, k)

  Test if a key exists.
- function get_map_v (m, k)

  Get the map value associated with a key.
- function get_map_kl (m)

  Get a list of all map keys.
- function get_map_vl (m)

  Get a list of all map values.
- function get_map_size (m)

  Get the number of map entries.
- module map_check (m, verbose=false)

  Perform some basic validation/checks on a map.
- module map_dump (m, sort=true, number=true, p=3)

  Dump each map entry to the console.

9.26.1 Detailed Description

Map data type operations.

Example

```
use <datatypes/datatypes_map.scad>;

map =
[
  ["part1",        ["screw10", [10, 11, 13]]],
```

```
  ["part2",       ["screw12", [20, 21, 30]]],
  ["part3",       ["screw10", [10, 10, -12]]],
  ["config",      ["top", "front", "rear"]],
  ["version",     [21, 5, 0]],
  ["runid",       10]
];

map_check(map, true);

echo( str("is part0 = ", map_exists(map, "part0")) );
echo( str("is part1 = ", map_exists(map, "part1")) );

p1 = get_map_v(map, "part1");
echo( c=second(p1) );

keys = get_map_kl(map);
parts = delete(keys, mv=["config", "version", "runid"]);

for ( p = parts )
  echo
  (
    n=p,
    p=first(get_map_v(map, p)),
    l=second(get_map_v(map, p))
  );

map_dump(map);
```

Result

```
ECHO: "[ INFO ] map_check(); begin map check"
ECHO: "[ INFO ] map_check(); checking map format and keys."
ECHO: "[ INFO ] map_check(); map size: 6 entries."
ECHO: "[ INFO ] map_check(); end map check"
ECHO: "is part0 = false"
ECHO: "is part1 = true"
ECHO: c = [10, 11, 13]
ECHO: n = "part1", p = "screw10", l = [10, 11, 13]
ECHO: n = "part2", p = "screw12", l = [20, 21, 30]
ECHO: n = "part3", p = "screw10", l = [10, 10, -12]
ECHO: "003:   'config' = '["top", "front", "rear"]'"
ECHO: "000:    'part1' = '["screw10", [10, 11, 13]]'"
ECHO: "001:    'part2' = '["screw12", [20, 21, 30]]'"
ECHO: "002:    'part3' = '["screw10", [10, 10, -12]]'"
ECHO: "005:    'runid' = '10'"
ECHO: "004:  'version' = '[21, 5, 0]'"
ECHO: "map size: 6 entries."
```

9.26.2 Function Documentation

9.26.2.1 function get_map_i (m , k)

Return the index of a map key.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.
k	<string> A map key.

Returns

<integer> The index of the map entry if it exists. Returns **undef** if `key` is not a string or does not exists.

9.26.2.2 function get_map_kl (m)

Get a list of all map keys.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.

Returns

<string-list-N> A list of key strings for all N map entries.

9.26.2.3 function get_map_size (m)

Get the number of map entries.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.

Returns

<integer> The number of map entries.

9.26.2.4 function get_map_v (m , k)

Get the map value associated with a key.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.
k	<string> A map key.

Returns

<value> The value associated with `key`. Returns **undef** if `key` does not exists.

9.26.2.5 function get_map_vl (m)

Get a list of all map values.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.

Returns

<list-N> A list of values for all N map entries.

9.26.2.6 module map_check (m , verbose = `false`)

Perform some basic validation/checks on a map.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.

verbose	<boolean> Be verbose during check.

Check that: (1) each entry has key-value 2-tuple, (2) each key is a string, and (3) key identifiers are unique.

Definition at line 149 of file datatypes_map.scad.

9.26.2.7 module map_dump (m , sort = `true`, number = `true`, p = `3`)

Dump each map entry to the console.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.
sort	<boolean> Sort the output by key.
number	<boolean> Output index number.
p	<integer> Number of places for zero-padded numbering.

Definition at line 223 of file datatypes_map.scad.

9.26.2.8 function map_exists (m , k)

Test if a key exists.

Parameters

m	<matrix-2xN> A list of N key-value map pairs.
k	<string> A map key.

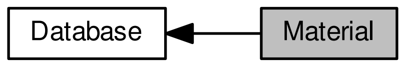
Returns

<boolean> **true** when the key exists and **false** otherwise.

9.27 Material

Material specifications.

Collaboration diagram for Material:

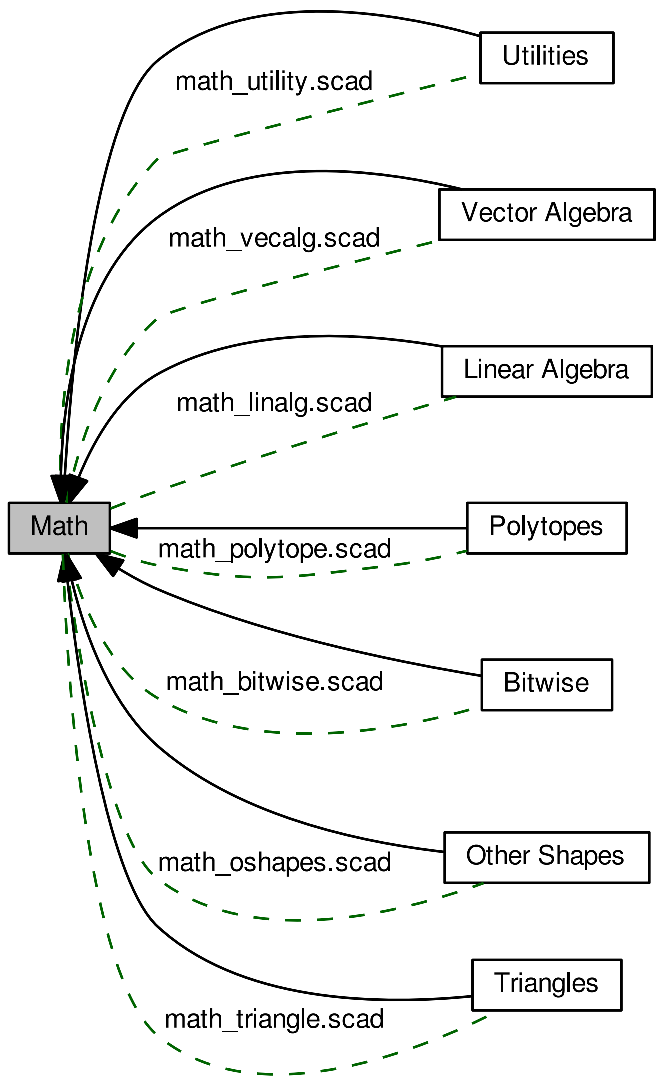


Material specifications.

9.28 Math

Mathematical functions.

Collaboration diagram for Math:



Modules

- Bitwise

    Base-two bitwise binary operations.
- Linear Algebra

    Linear algebra transformations on Euclidean coordinates.

- Other Shapes

    Mathematical functions for other shapes.
- Polytopes

    Polygon and polyhedron mathematical functions.
- Triangles

    Triangle mathematical functions.
- Utilities

    Miscellaneous mathematical utilities.
- Vector Algebra

    Algebraic operations on Euclidean vectors.

Files

- file math.scad

    Mathematical function primitives.
- file math_bitwise.scad

    Mathematical base-two bitwise binary functions.
- file math_linalg.scad

    Linear algebra mathematical functions.
- file math_oshapes.scad

    Other shapes mathematical functions.
- file math_polytope.scad

    Polygon and polyhedron mathematical functions.
- file math_triangle.scad

    Triangle solutions mathematical functions.
- file math_utility.scad

    Miscellaneous mathematical utilities.
- file math_vecalg.scad

    Vector algebra mathematical functions.

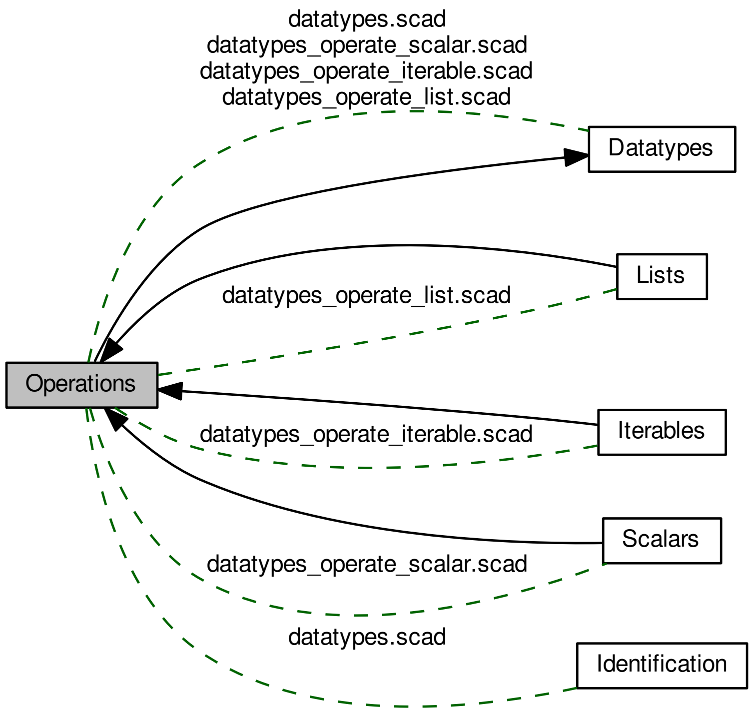
9.28.1 Detailed Description

Mathematical functions.

9.29 Operations

Data type operation.

Collaboration diagram for Operations:



Modules

- Iterables

  Iterable data type operations.
- Lists

  List data type operations.
- Scalars

  Scalar data type operations.

Files

- file datatypes.scad

  Data type identification and operations.
- file datatypes_operate_scalar.scad

  Scalar data type operations.
- file datatypes_operate_iterable.scad

Iterable data type operations.

- file datatypes_operate_list.scad

  List data type operations.

9.29.1 Detailed Description

Data type operation.

9.30 Other Shapes

Mathematical functions for other shapes.

Collaboration diagram for Other Shapes:

Files

- file math_oshapes.scad

  Other shapes mathematical functions.

Functions

- function rpolygon_lp (n, r, a, vr, cw=true)

  Compute the coordinates for an n-sided regular polygon.
- function rpolygon_area (n, r, a)

  Compute the area of an n-sided regular polygon.
- function rpolygon_perimeter (n, r, a)

  Compute the perimeter of an n-sided regular polygon.

9.30.1 Detailed Description

Mathematical functions for other shapes.

9.30.2 Function Documentation

9.30.2.1 function rpolygon_area (n , r , a)

Compute the area of an n-sided regular polygon.

Parameters

n	<integer> The number of sides.
r	<decimal> The vertex circumradius of the circumcircle.
a	<decimal> The inradius of the incircle.

Returns

<decimal> Area of the n-sided regular polygon.

The radius can be specified by either the circumradius `r` or the inradius `a`. If both are specified, `r` is used.

9.30.2.2 function rpolygon_lp (n , r , a , vr , cw = `true`)

Compute the coordinates for an n-sided regular polygon.

Parameters

n	<integer> The number of sides.
r	<decimal> The vertex circumradius of the circumcircle.
a	<decimal> The inradius of the incircle.
vr	<decimal> The vertex rounding radius.
cw	<boolean> Use clockwise point ordering.

Returns

<coords-2d> A list of coordinates points [[x, y], ...].

The radius can be specified by either the circumradius `r` or the inradius `a`. If both are specified, `r` is used.

Example

```
vr=5;

hull()
{
  for ( p = rpolygon_lp( r=20, n=5, vr=vr ) )
    translate( p )
    circle( r=vr );
}
```

9.30.2.3 function rpolygon_perimeter (n , r , a)

Compute the perimeter of an n-sided regular polygon.

Parameters

n	<integer> The number of sides.
r	<decimal> The vertex circumradius of the circumcircle.
a	<decimal> The inradius of the incircle.

Returns

<decimal> Perimeter length of the n-sided regular polygon.

The radius can be specified by either the circumradius `r` or the inradius `a`. If both are specified, `r` is used.

9.31 Parts

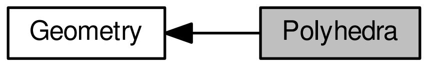
Parametric parts and assemblies.

Parametric parts and assemblies.

9.32 Polyhedra

Tables of polyhedra vertices, faces, and edges.

Collaboration diagram for Polyhedra:



Files

- file anti_prisms.scad

  Table of polyhedra data group: `anti_prisms`.
- file archimedean_duals.scad

  Table of polyhedra data group: `archimedean_duals`.
- file archimedean.scad

  Table of polyhedra data group: `archimedean`.
- file cupolas.scad

  Table of polyhedra data group: `cupolas`.
- file dipyramids.scad

  Table of polyhedra data group: `dipyramids`.
- file johnson.scad

  Table of polyhedra data group: `johnson`.
- file platonic.scad

  Table of polyhedra data group: `platonic`.
- file prisms.scad

  Table of polyhedra data group: `prisms`.
- file pyramids.scad

  Table of polyhedra data group: `pyramids`.
- file trapezohedron.scad

  Table of polyhedra data group: `trapezohedron`.
- file polyhedra_all.scad

  Table of polyhedra data group: `polyhedra_all`.

Variables

- dtc_polyhedra_anti_prisms
- dtr_polyhedra_anti_prisms
- dtc_polyhedra_archimedean_duals
- dtr_polyhedra_archimedean_duals
- dtc_polyhedra_archimedean
- dtr_polyhedra_archimedean

- dtc_polyhedra_cupolas
- dtr_polyhedra_cupolas
- dtc_polyhedra_dipyramids
- dtr_polyhedra_dipyramids
- dtc_polyhedra_johnson
- dtr_polyhedra_johnson
- dtc_polyhedra_platonic
- dtr_polyhedra_platonic
- dtc_polyhedra_prisms
- dtr_polyhedra_prisms
- dtc_polyhedra_pyramids
- dtr_polyhedra_pyramids
- dtc_polyhedra_trapezohedron
- dtr_polyhedra_trapezohedron
- dtc_polyhedra_polyhedra_all
- dtr_polyhedra_polyhedra_all

9.32.1 Detailed Description

Tables of polyhedra vertices, faces, and edges.

Database of polyhedra vertices, faces, and edges. Classes of polyhedra are grouped into separate files. The file polyhedra_all.scad contains all polyhedra from all files. Each table uses the following column data structure.

feild	description
id	identifier
n	name
o	other name
g	group
d	data source
c	cartesian vertices
s	spherical vertices
f	faces
e	edges

Use the functions get_table_v() to retrieve feild data as show in the following example. To see a list of table identifiers consider the function get_table_ridl(), get_table_cidl(), or module table_dump(). See datatypes_table.scad for other available table functions.

Example

```
include <units/units_coordinate.scad>;
include <tools/tools_polytope.scad>;
include <datatypes/datatypes_table.scad>;
include <database/geometry/polyhedra/platonic.scad>;

tc = dtc_polyhedra_platonic;
tr = dtr_polyhedra_platonic;

id = "dodecahedron";

pv = get_table_v(tr, tc, id, "c");
pf = get_table_v(tr, tc, id, "f");

sv = coordinates_csc(pv, 100);

polytope_number(sv, pf, to=[0,0,5]);
polytope_frame(sv, pf) {circle(r=2); color("grey") sphere(r=4);}
polyhedron(sv, pf);
```

Autotests

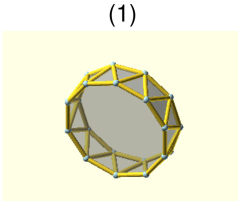
```
ECHO: "checking table: "
ECHO: "colums = dtc_polyhedra_polyhedra_all"
ECHO: "  rows = dtr_polyhedra_polyhedra_all"
ECHO: "[ INFO ] table_check(); begin table check"
ECHO: "[ INFO ] table_check(); row identifier found at column zero."
ECHO: "[ INFO ] table_check(); checking row column counts."
ECHO: "[ INFO ] table_check(); checking for repeat column identifiers."
ECHO: "[ INFO ] table_check(); checking for repeat row identifiers."
ECHO: "[ INFO ] table_check(); table size: 164 rows by 9 columns."
ECHO: "[ INFO ] table_check(); end table check"
ECHO:
ECHO: "confirming calculations stored in table:"
ECHO: "  (1) spherical with converted cartesian coordinates:"
ECHO: "  (2) edge list from faces definition:"
ECHO: "[ INFO ] root(); edge-list not sorted: id=anti_prisms_triangular_antiprism"
ECHO: "[ INFO ] root(); edge-list not sorted: id=archimedean_duals_rhombic_dodecahedron"
ECHO: "[ INFO ] root(); edge-list not sorted: id=archimedean_cuboctahedron"
ECHO: "[ INFO ] root(); edge-list not sorted: id=cupolas_triangular_gyrobicupola"
ECHO: "[ INFO ] root(); edge-list not sorted: id=platonic_octahedron"
ECHO:
ECHO: "164 polyhedra checked."
```

For more information see Wikipedia on Polyhedron.

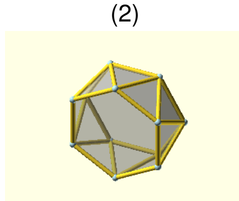
Group: anti_prisms

Table 2: anti_prisms

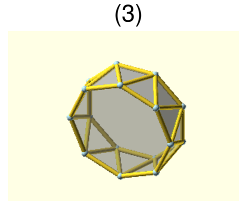
(1)



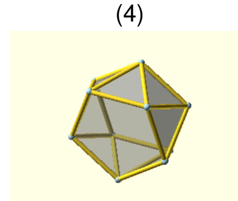
(2)



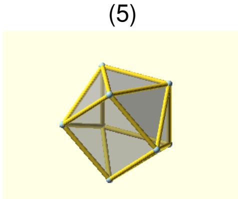
(3)



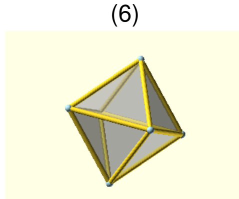
(4)



(5)



(6)



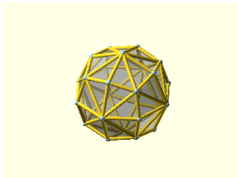
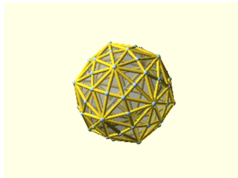
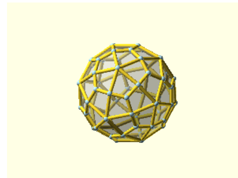
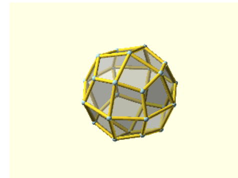
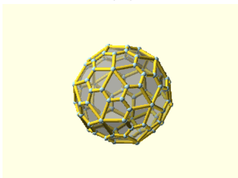
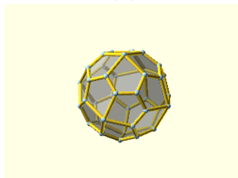
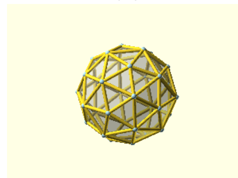
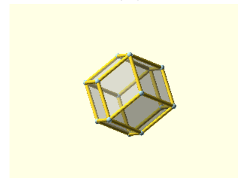
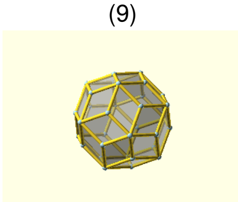
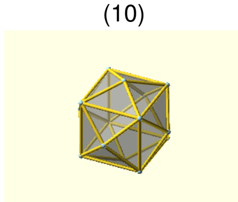
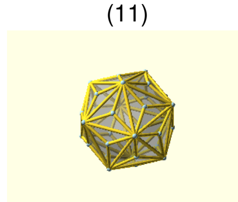
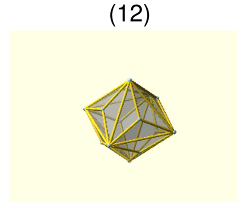
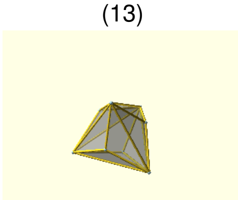
no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	decagonal_antiprism	-	20	22	40	3^{x20}, 10^{x2}	20.8^{x40}, 35.8^{x40}, 84.8^{x40}, 95.2^{x40}	0.597^{x40}	36^{x20}, 120^{x60}

2	hexagonal_antiprism	-	12	14	24	3^{x12}, 6^{x2}	34.8^{x24}, 59.2^{x24}, 81.1^{x24}, 98.9^{x24}	0.919^{x24}	60^{x12}, 120^{x36}
3	octagonal_antiprism	-	16	18	32	3^{x16}, 8^{x2}	26^{x32}, 44.7^{x32}, 83.4^{x32}, 96.6^{x32}	0.727^{x32}	45^{x16}, 120^{x48}
4	pentagonal_antiprism	-	10	12	20	3^{x10}, 5^{x2}	41.8^{x20}, 70.5^{x20}, 79.2^{x20}, 100.8^{x20}	1.05^{x20}	72^{x10}, 120^{x30}
5	square_antiprism	-	8	10	16	3^{x8}, 4^{x2}	52.4^{x16}, 76.2^{x16}, 86.7^{x16}, 103.8^{x16}	1.22^{x16}	90^{x8}, 120^{x24}
6	triangular_antiprism	Octahedron	6	8	12	3^{x8}	70.5^{x24}, 109.5^{x24}	1.41^{x12}	120^{x24}

Group: archimedean_duals

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	disdyakis_dodecahedron	Hexakis-Octahedron	26	48	72	3^{x48}	6.9^{x48}, 32.7^{x48}, 38.2^{x96}, 43.1^{x96}, 47.3^{x96}, 50.6^{x48}, 54.6^{x48}, 55.1^{x96}	0.606^{x24}, 0.765^{x24}, 0.919^{x24}	96.7^{x48}, 124.2^{x48}, 139.1^{x48}
2	disdyakis_triacontahedron	Hexakis-Icosahedron	62	120	180	3^{x120}	15.1^{x360}, 21.4^{x120}, 26.3^{x240}, 29^{x240}, 30.5^{x120}, 40.3^{x240}, 47.7^{x240}, 50.4^{x120}	0.37^{x60}, 0.581^{x60}, 0.683^{x60}	91^{x120}, 121.8^{x120}, 147.2^{x120}

Table 3: archimedean_duals

(1)

(2)

(3)

(4)

(5)

(6)

(7)

(8)

(9)

(10)

(11)

(12)

(13)


3	kite_hexecontahedron	-	62	60	120	4^{x60}	25.9^{x240}, 36.9^{x120}, 42.5^{x120}	0.37^{x60}, 0.569^{x60}	61.7^{x60}, 93^{x120}, 112.2^{x60}
4	kite_icositetrahedron	-	26	24	48	4^{x24}	41.9^{x96}, 60.7^{x72}	0.592^{x24}, 0.765^{x24}	64.7^{x24}, 98.4^{x72}
5	pentagonal_hexecontahedron	-	92	60	150	5^{x60}	26.8^{x300}, 44.1^{x120}	0.278^{x90}, 0.486^{x60}	61.9^{x240}, 112.5^{x60}
6	pentagonal_icositetrahedron	-	38	24	60	5^{x24}	43.7^{x120}, 63.5^{x24}	0.436^{x36}, 0.619^{x24}	65.2^{x96}, 99.2^{x24}

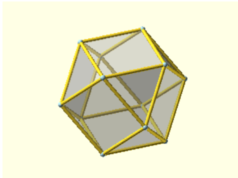
7	pentakis↩ _↩ dodecahedron	-	32	60	90	3^{x60}	23.3^{x180}, 38.1^{x120}, 40.9^{x240}, 47.6^{x120}	0.↩ 678^{x60}, 0.764^{x30}	111.↩ 4^{x60}, 124.↩ 3^{x120}
8	rhombic↩ _↩ dodecahedron	-	14	12	24	4^{x12}	60^{x48}, 90^{x24}	0.866^{x24}	70.5^{x24}, 109.5^{x24}
9	rhombic↩ _↩ triacontahedron	-	32	30	60	4^{x30}	36^{x120}, 60^{x120}	0.691^{x60}	63.4^{x60}, 116.6^{x60}
10	tetrakis↩ _↩ hexahedron	-	14	24	36	3^{x24}	36.9^{x72}, 53.1^{x24}, 66.4^{x96}, 78.5^{x48}	1.06^{x24}, 1.41^{x12}	96.4^{x24}, 131.8^{x48}
11	triakis↩ _↩ icosahedron	-	32	60	90	3^{x60}	19.4^{x180}, 37.4^{x240}, 52.3^{x240}, 62.4^{x240}, 66^{x120}	0.↩ 717^{x60}, 1.24^{x30}	61^{x60}, 149.↩ 5^{x120}
12	triakis↩ _↩ octahedron	-	14	24	36	3^{x24}	32.6^{x72}, 62.6^{x96}, 85.5^{x96}, 94.5^{x48}	1.17^{x24}, 2^{x12}	62.8^{x24}, 148.6^{x48}
13	triakis↩ _↩ tetrahedron	-	8	12	18	3^{x12}	50.5^{x36}, 95.2^{x48}, 117^{x24}	1.7^{x12}, 2.83^{x6}	67.1^{x12}, 146.4^{x24}

Group: archimedean

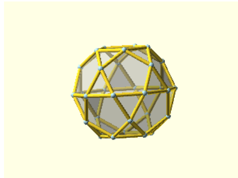
no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	cubocta-hedron	-	12	14	24	3^{x8}, 4^{x6}	54.7^{x48}, 70.5^{x24}, 90^{x24}	1.41^{x24}	90^{x24}, 120^{x24}
2	icosido-decahe-dron	-	30	32	60	3^{x20}, 5^{x12}	37.4^{x120}, 41.8^{x60}, 63.4^{x60}	0.65^{x60}	72^{x60}, 120^{x60}

Table 4: archimedean

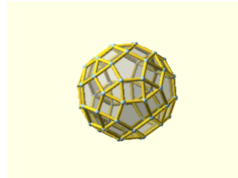
(1)



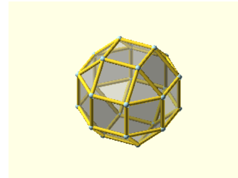
(2)



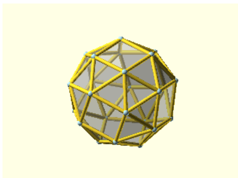
(3)



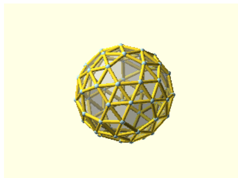
(4)



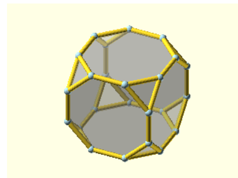
(5)



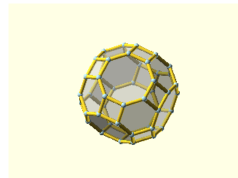
(6)



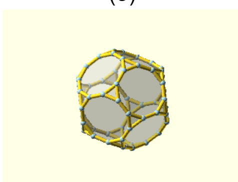
(7)



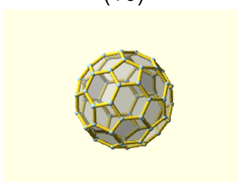
(8)



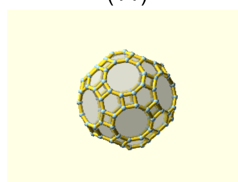
(9)



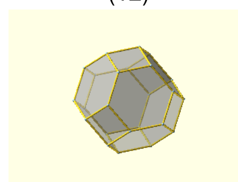
(10)



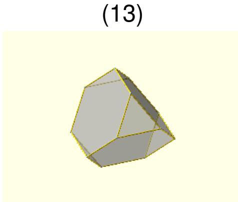
(11)



(12)



(13)



3	rhombicosidodecahedron	-	60	62	120	3^{x20}, 4^{x30}, 5^{x12}	20.9^{x120}, 31.7^{x120}, 36^{x120}, 37.4^{x120}	0.46^{x120}	72^{x60}, 90^{x120}, 120^{x60}
4	rhombicuboctahedron	-	24	26	48	3^{x8}, 4^{x18}	35.3^{x48}, 45^{x48}, 54.7^{x48}, 60^{x48}	0.765^{x48}	90^{x72}, 120^{x24}
5	snub_cuboctahedron	-	24	38	60	3^{x32}, 4^{x6}	26.8^{x72}, 37^{x48}, 45.9^{x96}, 50.4^{x48}, 54.7^{x96}	0.802^{x60}	90^{x24}, 120^{x96}

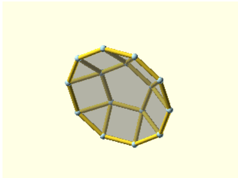
6	snub_icosidodecahedron	Snub Dodecahedron	60	92	150	3^{x80}, 5^{x12}	15.8^{x180}, 27.1^{x120}, 27.3^{x240}, 31^{x120}, 37.4^{x240}	0.477^{x150}	72^{x60}, 120^{x240}
7	truncated_cube	-	24	14	36	3^{x8}, 8^{x6}	54.7^{x48}, 90^{x24}	0.586^{x36}	45^{x48}, 120^{x24}
8	truncated_cuboctahedron	Great Rhombicuboctahedron	48	26	72	4^{x12}, 6^{x8}, 8^{x6}	35.3^{x48}, 45^{x48}, 54.7^{x48}	0.442^{x72}	45^{x48}, 60^{x48}, 90^{x48}
9	truncated_dodecahedron	-	60	32	90	3^{x20}, 10^{x12}	37.4^{x120}, 63.4^{x60}	0.342^{x90}	36^{x120}, 120^{x60}
10	truncated_icosahedron	Soccer Ball	60	32	90	5^{x12}, 6^{x20}	37.4^{x120}, 41.8^{x60}	0.412^{x90}	60^{x120}, 72^{x60}
11	truncated_icosidodecahedron	Great Rhombicosidodecahedron	120	62	180	4^{x30}, 6^{x20}, 10^{x12}	20.9^{x120}, 31.7^{x120}, 37.4^{x120}	0.265^{x180}	36^{x120}, 60^{x120}, 90^{x120}
12	truncated_octahedron	-	24	14	36	4^{x6}, 6^{x8}	54.7^{x48}, 70.5^{x24}	1.41^{x36}	60^{x48}, 90^{x24}
13	truncated_tetrahedron	-	12	8	18	3^{x4}, 6^{x4}	70.5^{x24}, 109.5^{x12}	2.83^{x18}	60^{x24}, 120^{x12}

Group: cupolas

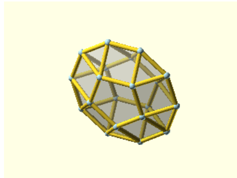
no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	pentagonal_cupola	J5	15	12	25	3^{x5}, 4^{x5}, 5^{x1}, 10^{x1}	20.9^{x20}, 31.7^{x10}, 36^{x10}, 37.4^{x10}, 142.6^{x10}, 148.3^{x10}	0.614^{x25}	36^{x10}, 72^{x5}, 90^{x20}, 120^{x15}

Table 5: cupolas

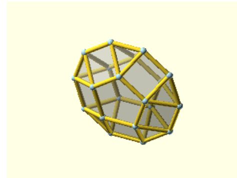
(1)



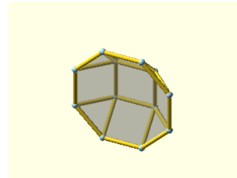
(2)



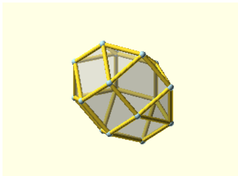
(3)



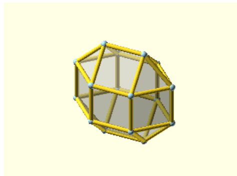
(4)



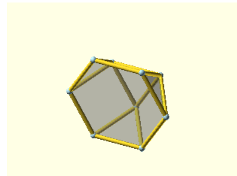
(5)



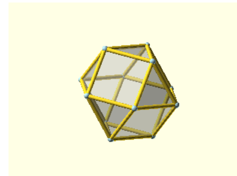
(6)



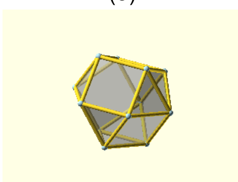
(7)



(8)



(9)



2	pentagonal_gyrobicupola	J31	20	22	40	3^{x10}, 4^{x10}, 5^{x2}	20.9^{x40}, 31.7^{x20}, 36^{x20}, 37.4^{x20}, 109.5^{x20}, 110.9^{x20}, 120^{x20}	0.618^{x40}	72^{x10}, 90^{x40}, 120^{x30}
3	pentagonal_orthobicupola	J30	20	22	40	3^{x10}, 4^{x10}, 5^{x2}	20.9^{x40}, 31.7^{x20}, 36^{x20}, 37.4^{x20}, 105.2^{x10}, 114.7^{x40}, 116.6^{x10}	0.618^{x40}	72^{x10}, 90^{x40}, 120^{x30}

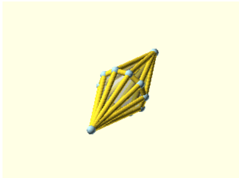
4	square← _cupola	J4	12	10	20	3^{x4}, 4^{x5}, 8^{x1}	35.3^{x16}, 45^{x8}, 54.7^{x8}, 60^{x8}, 125.3^{x8}, 135^{x8}	0.753^{x20}	45^{x8}, 90^{x20}, 120^{x12}
5	square← _← gyrobicupola	J29	16	18	32	3^{x8}, 4^{x10}	35.3^{x32}, 45^{x16}, 54.7^{x16}, 60^{x16}, 80.3^{x16}, 82.1^{x16}, 98.4^{x16}	0.765^{x32}	90^{x40}, 120^{x24}
6	square← _← orthobicupola	J28	16	18	32	3^{x8}, 4^{x10}	35.3^{x32}, 45^{x16}, 54.7^{x16}, 60^{x16}, 70.5^{x8}, 90^{x40}	0.765^{x32}	90^{x40}, 120^{x24}
7	triangular← _cupola	J3	9	8	15	3^{x4}, 4^{x3}, 6^{x1}	54.7^{x18}, 70.5^{x6}, 90^{x6}, 109.5^{x6}, 125.3^{x6}	0.965^{x15}	60^{x6}, 90^{x12}, 120^{x12}
8	triangular← _← gyrobicupola	Cuboc- tahe- dron	12	14	24	3^{x8}, 4^{x6}	54.7^{x48}, 70.5^{x24}, 90^{x24}	1^{x24}	90^{x24}, 120^{x24}
9	triangular← _← orthobicupola	J27	12	14	24	3^{x8}, 4^{x6}	38.9^{x6}, 54.7^{x36}, 70.5^{x18}, 78.9^{x24}, 90^{x12}	1^{x24}	90^{x24}, 120^{x24}

Group: dipyramids

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	decagonal← _bi_← pyramid	-	12	20	30	3^{x20}	34.3^{x60}, 49.4^{x40}, 68.3^{x40}, 101.← 2^{x40}, 130.← 6^{x40}, 145.7^{x20}	0.← 201^{x10}, 1.05^{x20}	95.5^{x40}, 169^{x20}

Table 6: dipyramids

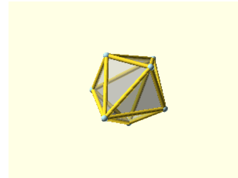
(1)



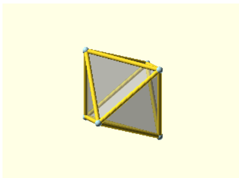
(2)

(3)

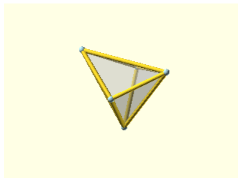
(4)



(5)



(6)



2	hexagonal_bi_pyramid	-	8	12	18	3^{x12}	53.1^{x36}, 78.5^{x24}, 101.5^{x24}, 126.9^{x12}	0.577^{x6}, 1.15^{x12}	104.5^{x24}, 151^{x12}
3	octagonal_di_pyramid	-	10	16	24	3^{x16}	41.9^{x48}, 60.7^{x32}, 82.7^{x32}, 119.3^{x32}, 138.1^{x16}	0.317^{x8}, 1.08^{x16}	98.4^{x32}, 163.2^{x16}
4	pentagonal_di_pyramid	-	7	10	15	3^{x10}	60.9^{x30}, 91.6^{x20}, 110.2^{x20}	0.854^{x5}, 1.24^{x10}	110.2^{x20}, 139.6^{x10}
5	square_di_pyramid	Octahedron	6	8	12	3^{x8}	70.5^{x24}, 109.5^{x24}	1.41^{x12}	120^{x24}
6	triangular_di_pyramid	-	5	6	9	3^{x6}	81.8^{x18}, 135.6^{x12}	1.15^{x6}, 1.73^{x3}	82.8^{x6}, 138.6^{x12}

Group: johnson

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	augmented↩ _↩ dodecahedron	J58	21	16	35	3^{x5}, 5^{x11}	26.1^{x10}, 41.8^{x10}, 58.5^{x20}, 63.4^{x50}, 70.5^{x10}	0.641^{x35}	72^{x55}, 120^{x15}
2	augmented↩ _↩ hexagonal↩ _prism	J54	13	11	22	3^{x4}, 4^{x5}, 6^{x2}	5.3^{x4}, 35.3^{x4}, 60^{x8}, 70.5^{x8}, 73.2^{x8}, 90^{x28}, 109.5^{x4}	0.689^{x22}	60^{x12}, 90^{x20}, 120^{x12}
3	augmented↩ _↩ pentagonal↩ _prism	J52	11	10	19	3^{x4}, 4^{x4}, 5^{x2}	17.3^{x4}, 35.3^{x4}, 70.5^{x8}, 72^{x6}, 79.7^{x8}, 90^{x24}, 109.5^{x4}	0.788^{x19}	72^{x10}, 90^{x16}, 120^{x12}
4	augmented↩ _↩ sphenocorona	J87	11	17	26	3^{x16}, 4^{x1}	8.2^{x2}, 15.7^{x4}, 20.1^{x4}, 27.8^{x2}, 31^{x4}, 36.5^{x8}, 44^{x8}, 48.6^{x2}, 61.1^{x8}, 61.7^{x4}, 62^{x4}, 68.2^{x8}, 69.7^{x4}, 70.5^{x12}, 70.7^{x4}, 72^{x8}, 72.3^{x8}, 74^{x4}, 74.8^{x4}, 78.9^{x8}, 81.3^{x4}, 82.5^{x2}, 82.7^{x8}, 85.7^{x4}, 87.5^{x8}, 104.5^{x4}, 109.5^{x4}	0.858^{x26}	90^{x4}, 120^{x48}

5	augmented_triangular_prism	J49	7	8	13	3^{x6}, 4^{x2}	35.3^{x4}, 65.3^{x4}, 70.5^{x8}, 90^{x16}, 106.8^{x8}, 109.5^{x4}, 120^{x2}	1.14^{x13}	90^{x8}, 120^{x18}
6	augmented_tridiminished_icosahedron	J64	10	10	18	3^{x7}, 5^{x3}	8.7^{x6}, 41.8^{x6}, 70.5^{x6}, 79.2^{x12}, 100.8^{x6}, 109.5^{x6}, 113.6^{x12}, 116.6^{x6}	0.976^{x18}	72^{x15}, 120^{x21}
7	augmented_truncated_cube	J66	28	22	48	3^{x12}, 4^{x5}, 8^{x5}	9.7^{x8}, 35.3^{x24}, 36.4^{x16}, 45^{x8}, 54.7^{x48}, 60^{x24}, 90^{x16}	0.506^{x48}	45^{x40}, 90^{x20}, 120^{x36}
8	augmented_truncated_dodecahedron	J68	65	42	105	3^{x25}, 4^{x5}, 5^{x1}, 10^{x11}	5.7^{x10}, 20.9^{x20}, 21.6^{x20}, 26.1^{x10}, 31.7^{x10}, 36^{x10}, 37.4^{x120}, 40.5^{x20}, 63.4^{x50}	0.316^{x105}	36^{x110}, 72^{x5}, 90^{x20}, 120^{x75}
9	augmented_truncated_tetrahedron	J65	15	14	27	3^{x8}, 4^{x3}, 6^{x3}	15.8^{x6}, 38.9^{x6}, 54.7^{x18}, 56.3^{x12}, 70.5^{x24}, 78.9^{x12}, 90^{x6}, 109.5^{x6}	0.7^{x27}	60^{x18}, 90^{x12}, 120^{x24}

10	biaugmented←_←pentagonal←_prism	J53	12	13	23	3^{x8}, 4^{x3}, 5^{x2}	17.3^{x8}, 35.3^{x8}, 70.5^{x16}, 72^{x2}, 79.7^{x16}, 90^{x28}, 109.5^{x8}	0.727^{x23}	72^{x10}, 90^{x12}, 120^{x24}
11	biaugmented←_←triangular←_prism	J50	8	11	17	3^{x10}, 4^{x1}	10.5^{x2}, 35.3^{x8}, 60^{x4}, 65.3^{x4}, 70.5^{x16}, 76^{x8}, 90^{x20}, 106.8^{x8}, 109.5^{x8}	1.06^{x17}	90^{x4}, 120^{x30}
12	biaugmented←_←truncated←_cube	J67	32	30	60	3^{x16}, 4^{x10}, 8^{x4}	9.7^{x16}, 35.3^{x48}, 36.4^{x32}, 45^{x16}, 54.7^{x48}, 60^{x48}, 90^{x8}	0.49^{x60}	45^{x32}, 90^{x40}, 120^{x48}
13	bigyrate←_←diminished←_←rhombicosidodecahedron	J79	55	52	105	3^{x15}, 4^{x25}, 5^{x11}, 10^{x1}	20.9^{x70}, 26.1^{x20}, 26.6^{x20}, 31.7^{x80}, 33.3^{x40}, 36^{x50}, 37.4^{x50}, 40.5^{x40}, 58.3^{x10}, 63.4^{x10}	0.←423^{x105}	36^{x10}, 72^{x55}, 90^{x100}, 120^{x45}
14	bilun-abiro-tunda	J91	14	14	26	3^{x8}, 4^{x2}, 5^{x4}	20.9^{x8}, 37.4^{x16}, 41.8^{x4}, 63.4^{x4}, 69.1^{x8}, 70.5^{x16}, 79.2^{x16}, 90^{x16}, 116.6^{x4}	0.714^{x26}	72^{x20}, 90^{x8}, 120^{x24}

15	diminished↩ _↩ rhombicosidodecahedron	J76	55	52	105	3^{x15}, 4^{x25}, 5^{x11}, 10^{x1}	20.9^{x90}, 31.7^{x100}, 36^{x90}, 37.4^{x90}, 58.3^{x10}, 63.4^{x10}	0.↩ 423^{x105}	36^{x10}, 72^{x55}, 90^{x100}, 120^{x45}
16	disphe- nocingu- lum	J90	16	24	38	3^{x20}, 4^{x4}	13.2^{x8}, 25.6^{x8}, 31.6^{x16}, 39.7^{x16}, 43.7^{x16}, 46.4^{x16}, 49.3^{x16}, 53.9^{x16}, 55.3^{x8}, 62.3^{x16}, 64.2^{x16}, 66.6^{x16}, 74.3^{x16}, 77.7^{x8}, 79.8^{x4}, 82.6^{x16}	0.825^{x38}	90^{x16}, 120^{x60}
17	elongated↩ _↩ pentagonal↩ _cupola	J20	25	22	45	3^{x5}, 4^{x15}, 5^{x1}, 10^{x1}	20.9^{x20}, 31.7^{x10}, 36^{x30}, 37.4^{x10}, 52.6^{x10}, 58.3^{x10}, 60.6^{x20}, 64.8^{x20}, 90^{x20}	0.567^{x45}	36^{x10}, 72^{x5}, 90^{x60}, 120^{x15}
18	elongated↩ _↩ pentagonal↩ _↩ dipyramid	J16	12	15	25	3^{x10}, 4^{x5}	41.8^{x20}, 52.6^{x20}, 70.5^{x20}, 72^{x10}, 79.2^{x40}	0.975^{x25}	90^{x20}, 120^{x30}
19	elongated↩ _↩ pentagonal↩ _↩ gyrobicupola	J39	30	32	60	3^{x10}, 4^{x20}, 5^{x2}	20.9^{x40}, 31.7^{x20}, 36^{x40}, 37.4^{x20}, 52.6^{x20}, 58.3^{x20}, 60.6^{x40}, 64.8^{x40}	0.59^{x60}	72^{x10}, 90^{x80}, 120^{x30}

20	elongated_pentagonal_gyrobirotunda	J43	40	42	80	3^{x20}, 4^{x10}, 5^{x12}	10.8^{x20}, 26.6^{x20}, 36^{x20}, 37.4^{x140}, 41.8^{x40}, 43.6^{x40}, 63.4^{x40}	0.485^{x80}	72^{x60}, 90^{x40}, 120^{x60}
21	elongated_pentagonal_gyrocupolarotunda	J41	35	37	70	3^{x15}, 4^{x15}, 5^{x7}	10.8^{x10}, 20.9^{x20}, 26.6^{x10}, 31.7^{x10}, 36^{x30}, 37.4^{x80}, 41.8^{x20}, 43.6^{x20}, 52.6^{x10}, 58.3^{x10}, 60.6^{x20}, 63.4^{x20}, 64.8^{x20}	0.552^{x70}	72^{x35}, 90^{x60}, 120^{x45}
22	elongated_pentagonal_orthobicupola	J38	30	32	60	3^{x10}, 4^{x20}, 5^{x2}	20.9^{x40}, 31.7^{x20}, 36^{x40}, 37.4^{x20}, 52.6^{x20}, 58.3^{x20}, 60.6^{x40}, 64.8^{x40}	0.59^{x60}	72^{x10}, 90^{x80}, 120^{x30}
23	elongated_pentagonal_orthobirotunda	J42	40	42	80	3^{x20}, 4^{x10}, 5^{x12}	10.8^{x20}, 26.6^{x20}, 36^{x20}, 37.4^{x140}, 41.8^{x40}, 43.6^{x40}, 63.4^{x40}	0.485^{x80}	72^{x60}, 90^{x40}, 120^{x60}
24	elongated_pentagonal_orthocupolarotunda	J40	35	37	70	3^{x15}, 4^{x15}, 5^{x7}	10.8^{x10}, 20.9^{x20}, 26.6^{x10}, 31.7^{x10}, 36^{x30}, 37.4^{x80}, 41.8^{x20}, 43.6^{x20}, 52.6^{x10}, 58.3^{x10}, 60.6^{x20}, 63.4^{x20}, 64.8^{x20}	0.552^{x70}	72^{x35}, 90^{x60}, 120^{x45}

25	elongated_pentagonal_pyramid	J9	11	11	20	3^{x5}, 4^{x5}, 5^{x1}	41.8^{x10}, 52.6^{x10}, 70.5^{x10}, 72^{x10}, 79.2^{x20}, 90^{x10}	0.964^{x20}	72^{x5}, 90^{x20}, 120^{x15}
26	elongated_pentagonal_rotunds	J21	30	27	55	3^{x10}, 4^{x10}, 5^{x6}, 10^{x1}	10.8^{x10}, 26.6^{x10}, 36^{x20}, 37.4^{x70}, 41.8^{x20}, 43.6^{x20}, 63.4^{x20}, 90^{x20}	0.52^{x55}	36^{x10}, 72^{x30}, 90^{x40}, 120^{x30}
27	elongated_square_cupola	J19	20	18	36	3^{x4}, 4^{x13}, 8^{x1}	35.3^{x24}, 45^{x32}, 54.7^{x24}, 60^{x24}, 90^{x16}	0.666^{x36}	45^{x8}, 90^{x52}, 120^{x12}
28	elongated_square_dipyramid	J15	10	12	20	3^{x8}, 4^{x4}	35.3^{x16}, 70.5^{x16}, 90^{x40}, 109.5^{x8}	0.828^{x20}	90^{x16}, 120^{x24}
29	elongated_square_gyrobicupola	J37	24	26	48	3^{x8}, 4^{x18}	35.3^{x48}, 45^{x48}, 54.7^{x48}, 60^{x48}	0.715^{x48}	90^{x72}, 120^{x24}
30	elongated_square_pyramid	J8	9	9	16	3^{x4}, 4^{x5}	35.3^{x8}, 70.5^{x8}, 90^{x32}, 109.5^{x4}	0.932^{x16}	90^{x20}, 120^{x12}
31	elongated_triangular_cupola	J18	15	14	27	3^{x4}, 4^{x9}, 6^{x1}	19.5^{x6}, 35.3^{x6}, 54.7^{x18}, 60^{x12}, 61.9^{x12}, 65.9^{x12}, 70.5^{x6}, 90^{x18}	0.795^{x27}	60^{x6}, 90^{x36}, 120^{x12}

32	elongated↩ _↩ triangular↩ _↩ dipyramid	J14	8	9	15	3^{x6}, 4^{x3}	19.5^{x12}, 109.↩ 5^{x12}, 118.↩ 1^{x24}, 120^{x6}	0.76^{x15}	90^{x12}, 120^{x18}
33	elongated↩ _↩ triangular↩ _↩ gyrobicupola	J36	18	20	36	3^{x8}, 4^{x12}	19.5^{x12}, 35.3^{x12}, 54.7^{x36}, 60^{x12}, 61.9^{x24}, 65.9^{x24}, 70.5^{x12}, 90^{x12}	0.696^{x36}	90^{x48}, 120^{x24}
34	elongated↩ _↩ triangular↩ _↩ orthobicupola	J35	18	20	36	3^{x8}, 4^{x12}	19.5^{x12}, 35.3^{x12}, 54.7^{x36}, 60^{x12}, 61.9^{x24}, 65.9^{x24}, 70.5^{x12}, 90^{x12}	0.696^{x36}	90^{x48}, 120^{x24}
35	elongated↩ _↩ triangular↩ _↩ pyramid	J7	7	7	12	3^{x4}, 4^{x3}	19.5^{x6}, 90^{x6}, 109.5^{x6}, 118.↩ 1^{x12}, 120^{x6}	0.886^{x12}	90^{x12}, 120^{x12}
36	gyrate↩ _↩ bidiminished↩ _↩ rhombicosidodecahedron	J82	50	42	90	3^{x10}, 4^{x20}, 5^{x10}, 10^{x2}	20.9^{x50}, 26.1^{x10}, 26.6^{x10}, 31.7^{x70}, 33.3^{x20}, 36^{x40}, 37.4^{x40}, 40.5^{x20}, 58.3^{x20}, 63.4^{x20}	0.41^{x90}	36^{x20}, 72^{x50}, 90^{x80}, 120^{x30}

37	gyrate↩ _↩ rhombicosidodecahedron	J72	60	62	120	3^{x20}, 4^{x30}, 5^{x12}	20.9^{x110}, 26.1^{x10}, 26.6^{x10}, 31.7^{x110}, 33.3^{x20}, 36^{x100}, 37.4^{x100}, 40.5^{x20}	0.↩ 448^{x120}	72^{x60}, 90^{x120}, 120^{x60}
38	gyrobi- fastigium	J26	8	8	14	3^{x4}, 4^{x4}	30^{x8}, 90^{x24}, 104.5^{x8}, 120^{x4}	1^{x14}	90^{x16}, 120^{x12}
39	gyroelongated↩ _↩ pentagonal↩ _↩ bicupola	J46	30	42	70	3^{x30}, 4^{x10}, 5^{x2}	20.8^{x40}, 20.9^{x40}, 31.7^{x20}, 35.8^{x40}, 36^{x20}, 37.4^{x20}, 47.4^{x20}, 53^{x20}, 55.8^{x40}, 59.9^{x80}, 65.2^{x40}	0.597^{x70}	72^{x10}, 90^{x40}, 120^{x90}
40	gyroelongated↩ _↩ pentagonal↩ _↩ birotunda	J48	40	52	90	3^{x40}, 5^{x12}	5.6^{x20}, 20.8^{x40}, 21.3^{x20}, 24.1^{x40}, 35.8^{x40}, 36.1^{x40}, 36.3^{x40}, 37.4^{x100}, 40.4^{x40}, 41.8^{x40}, 63.4^{x40}	0.501^{x90}	72^{x60}, 120^{x120}

41	gyroelongated↩ _↩ pentagonal↩ _cupola	J24	25	32	55	3^{x25}, 4^{x5}, 5^{x1}, 10^{x1}	20.8^{x40}, 20.9^{x20}, 31.7^{x10}, 35.8^{x40}, 36^{x10}, 37.4^{x10}, 47.4^{x10}, 53^{x10}, 55.8^{x20}, 59.8^{x12}, 59.9^{x28}, 65.2^{x20}, 84.8^{x20}, 95.2^{x20}	0.577^{x55}	36^{x10}, 72^{x5}, 90^{x20}, 120^{x75}
42	gyroelongated↩ _↩ pentagonal↩ _↩ cupolarotunda	J47	35	47	80	3^{x35}, 4^{x5}, 5^{x7}	5.6^{x10}, 20.8^{x40}, 20.9^{x20}, 21.3^{x10}, 24.1^{x20}, 31.7^{x10}, 35.8^{x40}, 36^{x10}, 36.1^{x20}, 36.3^{x20}, 37.4^{x60}, 40.4^{x20}, 41.8^{x20}, 47.4^{x10}, 53^{x10}, 55.8^{x20}, 59.9^{x40}, 63.4^{x20}, 65.2^{x20}	0.563^{x80}	72^{x35}, 90^{x20}, 120^{x105}

43	gyroelongated↩ _↩ pentagonal↩ _↩ pyramid	J11	11	16	25	3^{x15}, 5^{x1}	41.8^{x40}, 70.5^{x70}, 79.2^{x10}, 100.8^{x10}	1.01^{x25}	72^{x5}, 120^{x45}
44	gyroelongated↩ _↩ pentagonal↩ _↩ rotunda	J25	30	37	65	3^{x30}, 5^{x6}, 10^{x1}	5.6^{x10}, 20.8^{x40}, 21.3^{x10}, 24.1^{x20}, 35.8^{x40}, 36.1^{x20}, 36.3^{x20}, 37.4^{x50}, 40.4^{x20}, 41.8^{x20}, 63.4^{x20}, 84.8^{x20}, 95.2^{x20}	0.591^{x65}	36^{x10}, 72^{x30}, 120^{x90}
45	gyroelongated↩ _↩ square↩ _↩ bicupola	J45	24	34	56	3^{x24}, 4^{x10}	26^{x32}, 28.7^{x16}, 35.3^{x32}, 38.4^{x16}, 44.7^{x32}, 45^{x16}, 46.9^{x32}, 50.2^{x32}, 54.7^{x48}, 55.4^{x32}, 60^{x16}	0.727^{x56}	90^{x40}, 120^{x72}
46	gyroelongated↩ _↩ square↩ _cupola	J23	20	26	44	3^{x20}, 4^{x5}, 8^{x1}	26^{x32}, 28.7^{x8}, 35.3^{x16}, 38.4^{x8}, 44.7^{x32}, 45^{x8}, 46.9^{x16}, 50.2^{x16}, 54.7^{x24}, 55.4^{x16}, 60^{x8}, 83.4^{x16}, 96.6^{x16}	0.741^{x44}	45^{x8}, 90^{x20}, 120^{x60}

47	gyroelongated_square_dipyramid	J17	10	16	24	3^{x16}	21.4^{x16}, 52.4^{x16}, 65^{x32}, 70.5^{x16}, 82.1^{x32}, 86.7^{x16}, 109.5^{x8}	0.887^{x24}	120^{x48}
48	gyroelongated_square_pyramid	J10	9	13	20	3^{x12}, 4^{x1}	21.4^{x8}, 52.4^{x16}, 65^{x16}, 70.5^{x8}, 76.2^{x8}, 82.1^{x16}, 86.7^{x16}, 103.8^{x8}, 109.5^{x4}	0.998^{x20}	90^{x4}, 120^{x36}
49	gyroelongated_triangular_bicupola	J44	18	26	42	3^{x20}, 4^{x6}	10.6^{x12}, 26.4^{x12}, 34.8^{x24}, 41^{x24}, 52.5^{x24}, 54.7^{x36}, 58.8^{x24}, 59.2^{x24}, 60.5^{x24}, 70.5^{x12}, 90^{x12}	0.729^{x42}	90^{x24}, 120^{x60}
50	gyroelongated_triangular_cupola	J22	15	20	33	3^{x16}, 4^{x3}, 6^{x1}	10.6^{x6}, 26.4^{x6}, 34.8^{x24}, 41^{x12}, 52.5^{x12}, 54.7^{x18}, 58.8^{x12}, 59.2^{x24}, 60.5^{x12}, 70.5^{x6}, 81.1^{x12}, 90^{x6}, 98.9^{x12}	0.891^{x33}	60^{x6}, 90^{x12}, 120^{x48}

51	hebe-sphe-nomega-corona	J89	14	21	33	3^{x18}, 4^{x3}	22.9^{x16}, 27^{x4}, 30.4^{x2}, 38.7^{x8}, 39.3^{x8}, 46^{x12}, 51.5^{x16}, 51.8^{x8}, 53.3^{x8}, 56.4^{x8}, 60^{x8}, 65.1^{x16}, 65.5^{x4}, 68.3^{x4}, 70.2^{x8}, 76^{x8}, 77.5^{x4}, 80.7^{x16}, 81.3^{x16}, 85.3^{x8}	0.889^{x33}	90^{x12}, 120^{x54}
52	metabiaugmented↩ _↩ dodecahedron	J60	22	20	40	3^{x10}, 5^{x10}	26.1^{x20}, 41.8^{x20}, 58.5^{x40}, 63.4^{x40}, 70.5^{x20}	0.625^{x40}	72^{x50}, 120^{x30}
53	metabiaugmented↩ _↩ hexagonal↩ _prism	J56	14	14	26	3^{x8}, 4^{x4}, 6^{x2}	5.3^{x8}, 35.3^{x8}, 60^{x4}, 70.5^{x16}, 73.2^{x16}, 90^{x32}, 109.5^{x8}	0.658^{x26}	60^{x12}, 90^{x16}, 120^{x24}
54	metabiaugmented↩ _↩ truncated↩ _↩ dodecahedron	J70	70	52	120	3^{x30}, 4^{x10}, 5^{x2}, 10^{x10}	5.7^{x20}, 20.9^{x40}, 21.6^{x40}, 26.1^{x20}, 31.7^{x20}, 36^{x20}, 37.4^{x120}, 40.5^{x40}, 63.4^{x40}	0.↩ 313^{x120}	36^{x100}, 72^{x10}, 90^{x40}, 120^{x90}

55	metabidiminished↩ _↩ icosahedron	J62	10	12	20	3^{x10}, 5^{x2}	41.8^{x22}, 70.5^{x32}, 79.2^{x16}, 100.↩ 8^{x12}, 116.6^{x2}	0.964^{x20}	72^{x10}, 120^{x30}
56	metabidiminished↩ _↩ rhombicosidodecahedron	J81	50	42	90	3^{x10}, 4^{x20}, 5^{x10}, 10^{x2}	20.9^{x60}, 31.7^{x80}, 36^{x60}, 37.4^{x60}, 58.3^{x20}, 63.4^{x20}	0.41^{x90}	36^{x20}, 72^{x50}, 90^{x80}, 120^{x30}
57	metabigyrate↩ _↩ rhombicosidodecahedron	J74	60	62	120	3^{x20}, 4^{x30}, 5^{x12}	20.9^{x100}, 26.1^{x20}, 26.6^{x20}, 31.7^{x100}, 33.3^{x40}, 36^{x80}, 37.4^{x80}, 40.5^{x40}	0.↩ 448^{x120}	72^{x60}, 90^{x120}, 120^{x60}
58	metagyrate↩ _↩ diminished↩ _↩ rhombicosidodecahedron	J78	55	52	105	3^{x15}, 4^{x25}, 5^{x11}, 10^{x1}	20.9^{x80}, 26.1^{x10}, 26.6^{x10}, 31.7^{x90}, 33.3^{x20}, 36^{x70}, 37.4^{x70}, 40.5^{x20}, 58.3^{x10}, 63.4^{x10}	0.↩ 423^{x105}	36^{x10}, 72^{x55}, 90^{x100}, 120^{x45}
59	parabiaugmented↩ _↩ dodecahedron	J59	22	20	40	3^{x10}, 5^{x10}	26.1^{x20}, 41.8^{x20}, 58.5^{x40}, 63.4^{x40}, 70.5^{x20}	0.61^{x40}	72^{x50}, 120^{x30}
60	parabiaugmented↩ _↩ hexagonal↩ _prism	J55	14	14	26	3^{x8}, 4^{x4}, 6^{x2}	5.3^{x8}, 35.3^{x8}, 60^{x4}, 70.5^{x16}, 73.2^{x16}, 90^{x32}, 109.5^{x8}	0.636^{x26}	60^{x12}, 90^{x16}, 120^{x24}

61	parabiaugmented↩ _↩ truncated↩ _↩ dodecahedron	J69	70	52	120	3^{x30}, 4^{x10}, 5^{x2}, 10^{x10}	5.7^{x20}, 20.9^{x40}, 21.6^{x40}, 26.1^{x20}, 31.7^{x20}, 36^{x20}, 37.4^{x120}, 40.5^{x40}, 63.4^{x40}	0.↩ 319^{x120}	36^{x100}, 72^{x10}, 90^{x40}, 120^{x90}
62	parabidiminished↩ _↩ rhombicosidodecahedron	J80	50	42	90	3^{x10}, 4^{x20}, 5^{x10}, 10^{x2}	20.9^{x60}, 31.7^{x80}, 36^{x60}, 37.4^{x60}, 58.3^{x20}, 63.4^{x20}	0.448^{x90}	36^{x20}, 72^{x50}, 90^{x80}, 120^{x30}
63	parabigyrate↩ _↩ rhombicosidodecahedron	J73	60	62	120	3^{x20}, 4^{x30}, 5^{x12}	20.9^{x100}, 26.1^{x20}, 26.6^{x20}, 31.7^{x100}, 33.3^{x40}, 36^{x80}, 37.4^{x80}, 40.5^{x40}	0.↩ 448^{x120}	72^{x60}, 90^{x120}, 120^{x60}
64	paragyrate↩ _↩ diminished↩ _↩ rhombicosidodecahedron	J77	55	52	105	3^{x15}, 4^{x25}, 5^{x11}, 10^{x1}	20.9^{x80}, 26.1^{x10}, 26.6^{x10}, 31.7^{x90}, 33.3^{x20}, 36^{x70}, 37.4^{x70}, 40.5^{x20}, 58.3^{x10}, 63.4^{x10}	0.↩ 423^{x105}	36^{x10}, 72^{x55}, 90^{x100}, 120^{x45}
65	pentagonal↩ _cupola	J5	15	12	25	3^{x5}, 4^{x5}, 5^{x1}, 10^{x1}	20.9^{x20}, 31.7^{x10}, 36^{x10}, 37.4^{x10}, 142.↩ 6^{x10}, 148.3^{x10}	0.614^{x25}	36^{x10}, 72^{x5}, 90^{x20}, 120^{x15}

66	pentagonal← _← dipyramid	J13	7	10	15	3^{x10}	41.8^{x20}, 70.5^{x20}, 105.← 2^{x10}, 121.2^{x20}	1.18^{x15}	120^{x30}
67	pentagonal← _← gyrobicupola	J31	20	22	40	3^{x10}, 4^{x10}, 5^{x2}	20.9^{x40}, 31.7^{x20}, 36^{x20}, 37.4^{x20}, 109.← 5^{x20}, 110.← 9^{x20}, 120^{x20}	0.618^{x40}	72^{x10}, 90^{x40}, 120^{x30}
68	pentagonal← _← gyrocupolarotunda	J33	25	27	50	3^{x15}, 4^{x5}, 5^{x7}	20.9^{x20}, 31.7^{x10}, 36^{x10}, 37.4^{x60}, 41.8^{x20}, 63.4^{x30}, 75^{x20}, 84.8^{x10}, 85.2^{x20}	0.605^{x50}	72^{x35}, 90^{x20}, 120^{x45}
69	pentagonal← _← orthobicupola	J30	20	22	40	3^{x10}, 4^{x10}, 5^{x2}	20.9^{x40}, 31.7^{x20}, 36^{x20}, 37.4^{x20}, 105.← 2^{x10}, 114.← 7^{x40}, 116.6^{x10}	0.618^{x40}	72^{x10}, 90^{x40}, 120^{x30}
70	pentagonal← _← orthobirotunda	J34	30	32	60	3^{x20}, 5^{x12}	21.6^{x10}, 37.4^{x100}, 41.8^{x40}, 51.2^{x40}, 53.1^{x10}, 63.4^{x40}	0.618^{x60}	72^{x60}, 120^{x60}
71	pentagonal← _← orthocupolarontunda	J32	25	27	50	3^{x15}, 4^{x5}, 5^{x7}	20.9^{x20}, 31.7^{x10}, 36^{x10}, 37.4^{x60}, 41.8^{x20}, 63.4^{x20}, 69.1^{x10}, 70.5^{x20}, 79.2^{x10}, 90^{x20}	0.605^{x50}	72^{x35}, 90^{x20}, 120^{x45}

72	pentagonal↩ _↩ pyramid	J2	6	6	10	3^{x5}, 5^{x1}	41.8^{x10}, 70.5^{x10}, 142.6^{x10}	1.17^{x10}	72^{x5}, 120^{x15}
73	pentagonal↩ _↩ rotunda	J6	20	17	35	3^{x10}, 5^{x6}, 10^{x1}	37.4^{x50}, 41.8^{x20}, 63.4^{x20}, 100.↩ 8^{x10}, 116.6^{x10}	0.584^{x35}	36^{x10}, 72^{x30}, 120^{x30}
74	snub_↩ disphenoid	J84	8	12	18	3^{x12}	13.5^{x2}, 13.6^{x6}, 58.2^{x4}, 58.3^{x12}, 65.7^{x16}, 83.8^{x12}, 90.6^{x4}, 90.7^{x12}, 94.9^{x8}, 111.5^{x16}	1.08^{x18}	120^{x36}
75	snub_↩ square↩ _↩ antiprism	J85	16	26	40	3^{x24}, 4^{x2}	15.7^{x16}, 34.6^{x16}, 35.9^{x32}, 45.5^{x32}, 47.3^{x16}, 59.9^{x32}, 61^{x16}, 65.4^{x16}, 73.9^{x32}, 85.9^{x32}	0.815^{x40}	90^{x8}, 120^{x72}
76	spheno-corona	J86	10	14	22	3^{x12}, 4^{x2}	20.1^{x4}, 36.5^{x8}, 44^{x8}, 48.6^{x2}, 61.1^{x8}, 62^{x4}, 63^{x2}, 68.2^{x8}, 70.5^{x8}, 72^{x8}, 72.3^{x8}, 74^{x4}, 81.3^{x8}, 82.5^{x4}, 82.7^{x8}, 87.5^{x8}, 104.5^{x8}	1^{x22}	90^{x8}, 120^{x36}

77	sphenomegacorona	J88	12	18	28	3^{x16}, 4^{x2}	8.3^{x4}, 8.4^{x4}, 18.6^{x2}, 25.3^{x8}, 36.3^{x8}, 41^{x8}, 42.8^{x4}, 48.4^{x8}, 50.6^{x8}, 55.1^{x8}, 61.6^{x4}, 62.6^{x4}, 62.7^{x4}, 65.8^{x8}, 67^{x12}, 67.1^{x4}, 72.8^{x4}, 72.9^{x4}, 83.8^{x8}, 93.3^{x8}, 97.4^{x8}, 100.6^{x4}, 105.4^{x8}, 107^{x2}, 114.9^{x8}	0.↩ 761^{x26}, 0.762^{x2}	90^{x8}, 120^{x48}

78	square← _cupola	J4	12	10	20	3^{x4}, 4^{x5}, 8^{x1}	35.3^{x16}, 45^{x8}, 54.7^{x8}, 60^{x8}, 125.3^{x8}, 135^{x8}	0.753^{x20}	45^{x8}, 90^{x20}, 120^{x12}
79	square← _← gyrobicupola	J29	16	18	32	3^{x8}, 4^{x10}	35.3^{x32}, 45^{x16}, 54.7^{x16}, 60^{x16}, 80.3^{x16}, 82.1^{x16}, 98.4^{x16}	0.765^{x32}	90^{x40}, 120^{x24}
80	square← _← orthobicupola	J28	16	18	32	3^{x8}, 4^{x10}	35.3^{x32}, 45^{x16}, 54.7^{x16}, 60^{x16}, 70.5^{x8}, 90^{x40}	0.765^{x32}	90^{x40}, 120^{x24}
81	square← _← pyramid	J1	5	5	8	3^{x4}, 4^{x1}	70.5^{x8}, 109.5^{x4}, 125.3^{x8}	1.41^{x8}	90^{x4}, 120^{x12}
82	triangular← _cupola	J3	9	8	15	3^{x4}, 4^{x3}, 6^{x1}	54.7^{x18}, 70.5^{x6}, 90^{x6}, 109.5^{x6}, 125.3^{x6}	0.965^{x15}	60^{x6}, 90^{x12}, 120^{x12}
83	triangular← _← dipyramid	J12	5	6	9	3^{x6}	38.9^{x6}, 109.← 5^{x12}, 123.7^{x12}	1.22^{x9}	120^{x18}
84	triangular← _← hebesphenorotunda	J92	18	20	36	3^{x13}, 4^{x3}, 5^{x3}, 6^{x1}	20.9^{x12}, 37.4^{x18}, 41.8^{x24}, 54.7^{x12}, 63.4^{x6}, 69.1^{x12}, 70.5^{x30}, 79.2^{x12}, 90^{x12}, 100.8^{x6}	0.661^{x36}	60^{x6}, 72^{x15}, 90^{x12}, 120^{x39}

85	triangular↵ _↵ orthobicupola	J27	12	14	24	3^{x8}, 4^{x6}	38.9^{x6}, 54.7^{x36}, 70.5^{x18}, 78.9^{x24}, 90^{x12}	1^{x24}	90^{x24}, 120^{x24}
86	triaugmented↵ _↵ dodecahedron	J61	23	24	45	3^{x15}, 5^{x9}	26.1^{x30}, 41.8^{x30}, 58.5^{x60}, 63.4^{x30}, 70.5^{x30}	0.613^{x45}	72^{x45}, 120^{x45}
87	triaugmented↵ _↵ hexagonal↵ _prism	J57	15	17	30	3^{x12}, 4^{x3}, 6^{x2}	5.3^{x12}, 35.3^{x12}, 70.5^{x24}, 73.2^{x24}, 90^{x36}, 109.5^{x12}	0.636^{x30}	60^{x12}, 90^{x12}, 120^{x36}
88	triaugmented↵ _↵ triangular↵ _prism	J51	9	14	21	3^{x14}	10.5^{x6}, 35.3^{x12}, 60^{x12}, 70.5^{x24}, 76^{x24}, 90^{x24}, 109.5^{x12}	1^{x21}	120^{x42}
89	triaugmented↵ _↵ truncated↵ _↵ dodecahedron	J71	75	62	135	3^{x35}, 4^{x15}, 5^{x3}, 10^{x9}	5.7^{x30}, 9.8^{x4}, 11.3^{x2}, 18^{x8}, 20.9^{x60}, 21.6^{x60}, 26.1^{x26}, 31.7^{x30}, 36^{x30}, 37.4^{x120}, 40.5^{x52}, 63.4^{x32}	0.↵ 307^{x135}	36^{x90}, 72^{x15}, 90^{x60}, 120^{x105}

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
90	tridiminished_icosahedron	J63	9	8	15	3^{x5}, 5^{x3}	41.8^{x6}, 70.5^{x6}, 79.2^{x18}, 100.8^{x6}, 116.6^{x6}	1^{x15}	72^{x15}, 120^{x15}
91	tridiminished_rhombicosidodecahedron	J83	45	32	75	3^{x5}, 4^{x15}, 5^{x9}, 10^{x3}	20.9^{x30}, 31.7^{x60}, 36^{x30}, 37.4^{x30}, 58.3^{x30}, 63.4^{x30}	0.424^{x75}	36^{x30}, 72^{x45}, 90^{x60}, 120^{x15}
92	trigyrate_rhombicosidodecahedron	J75	60	62	120	3^{x20}, 4^{x30}, 5^{x12}	20.9^{x90}, 26.1^{x30}, 26.6^{x30}, 31.7^{x90}, 33.3^{x60}, 36^{x60}, 37.4^{x60}, 40.5^{x60}	0.448^{x120}	72^{x60}, 90^{x120}, 120^{x60}

Group: platonic

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	cube	Hexahedron	8	6	12	4^{x6}	90^{x24}	2^{x12}	90^{x24}
2	dodecahedron	-	20	12	30	5^{x12}	63.4^{x60}	0.714^{x30}	72^{x60}
3	icosahedron	-	12	20	30	3^{x20}	41.8^{x60}, 70.5^{x120}	1.24^{x30}	120^{x60}
4	octahedron	-	6	8	12	3^{x8}	70.5^{x24}, 109.5^{x24}	1.41^{x12}	120^{x24}
5	tetrahedron	-	4	4	6	3^{x4}	109.5^{x12}	2.83^{x6}	120^{x12}

Group: prisms

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	decagonal_prism	-	20	12	30	4^{x10}, 10^{x2}	36^{x20}, 90^{x40}	0.59^{x30}	36^{x20}, 90^{x40}
2	hexagonal_prism	-	12	8	18	4^{x6}, 6^{x2}	60^{x12}, 90^{x24}	0.894^{x18}	60^{x12}, 90^{x24}
3	octagonal_prism	-	16	10	24	4^{x8}, 8^{x2}	45^{x16}, 90^{x32}	0.715^{x24}	45^{x16}, 90^{x32}

4	pentagonal_prism	-	10	7	15	4^{x5}, 5^{x2}	72^{x10}, 90^{x20}	1.01^{x15}	72^{x10}, 90^{x20}
5	square_prism	Cube	8	6	12	4^{x6}	90^{x24}	1.15^{x12}	90^{x24}
6	triangular_prism	-	6	5	9	3^{x2}, 4^{x3}	90^{x12}, 120^{x6}	1.31^{x9}	90^{x12}, 120^{x6}

Group: pyramids

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	pentagonal_dipyramid	J13	7	10	15	3^{x10}	41.8^{x20}, 70.5^{x20}, 105.2^{x10}, 121.2^{x20}	1.18^{x15}	120^{x30}
2	pentagonal_pyramid	J2	6	6	10	3^{x5}, 5^{x1}	41.8^{x10}, 70.5^{x10}, 142.6^{x10}	1.17^{x10}	72^{x5}, 120^{x15}
3	square_dipryamid	Octahedron	6	8	12	3^{x8}	70.5^{x24}, 109.5^{x24}	1.41^{x12}	120^{x24}
4	square_pyramid	J1	5	5	8	3^{x4}, 4^{x1}	70.5^{x8}, 109.5^{x4}, 125.3^{x8}	1.41^{x8}	90^{x4}, 120^{x12}
5	triangular_dipyramid	J12	5	6	9	3^{x6}	38.9^{x6}, 109.5^{x12}, 123.7^{x12}	1.22^{x9}	120^{x18}
6	triangular_pyramid	Tetrahedron	4	4	6	3^{x4}	109.5^{x12}	1.63^{x6}	120^{x12}

Group: trapezohedron

no.	table id	other name	vertices	faces	edges	face-verticies	face-angles	edge-lengths	edge-angles
1	decagonal_trapezohedron	-	22	20	40	4^{x20}	34.7^{x80}, 69.2^{x40}, 102.8^{x40}, 133.6^{x40}, 150.2^{x20}	0.0991^{x20}, 1.01^{x20}	63.2^{x60}, 170.4^{x20}
2	enneagonal_trapezohedron	-	20	18	36	4^{x18}	38.3^{x72}, 76.1^{x36}, 112.3^{x36}, 141.7^{x36}	0.122^{x18}, 1.02^{x18}	63.9^{x54}, 168.3^{x18}
3	heptagonal_trapezohedron	-	16	14	28	4^{x14}	48^{x56}, 94.2^{x28}, 132^{x28}	0.203^{x14}, 1.03^{x14}	66.4^{x42}, 160.9^{x14}
4	hexagonal_trapezohedron	-	14	12	24	4^{x12}	54.7^{x48}, 105.5^{x24}, 133.7^{x12}	0.277^{x12}, 1.04^{x12}	68.5^{x36}, 154.4^{x12}
5	octagonal_trapezohedron	-	18	16	32	4^{x16}	42.6^{x64}, 84.4^{x32}, 122.7^{x32}, 143.6^{x16}	0.155^{x16}, 1.02^{x16}	64.9^{x48}, 165.2^{x16}
6	pentagonal_trapezohedron	-	12	10	20	4^{x10}	63.4^{x40}, 116.6^{x20}	0.402^{x10}, 1.05^{x10}	72^{x30}, 144^{x10}
7	square_trapezohedron	-	10	8	16	4^{x8}	74.9^{x32}, 118.5^{x8}	0.634^{x8}, 1.08^{x8}	78^{x24}, 125.9^{x8}
8	triangular_trapezohedron	Cube	8	6	12	4^{x6}	90^{x24}	1.15^{x12}	90^{x24}

9.32.2 Variable Documentation

9.32.2.1 dtc_polyhedra_anti_prisms

<matrix-2x9> `anti_prisms` polyhedra data table columns definition.

Definition at line 114 of file anti_prisms.scad.

9.32.2.2 dtc_polyhedra_archimedean

<matrix-2x9> `archimedean` polyhedra data table columns definition.

Definition at line 114 of file archimedean.scad.

9.32.2.3 dtc_polyhedra_archimedean_duals

<matrix-2x9> `archimedean_duals` polyhedra data table columns definition.

Definition at line 114 of file archimedean_duals.scad.

9.32.2.4 dtc_polyhedra_cupolas

<matrix-2x9> `cupolas` polyhedra data table columns definition.

Definition at line 114 of file cupolas.scad.

9.32.2.5 dtc_polyhedra_dipyramids

<matrix-2x9> `dipyramids` polyhedra data table columns definition.

Definition at line 114 of file dipyramids.scad.

9.32.2.6 dtc_polyhedra_johnson

<matrix-2x9> `johnson` polyhedra data table columns definition.

Definition at line 114 of file johnson.scad.

9.32.2.7 dtc_polyhedra_platonic

<matrix-2x9> `platonic` polyhedra data table columns definition.

Definition at line 114 of file platonic.scad.

9.32.2.8 dtc_polyhedra_polyhedra_all

<matrix-2x9> `polyhedra_all` polyhedra data table columns definition.

Definition at line 125 of file polyhedra_all.scad.

9.32.2.9 dtc_polyhedra_prisms

<matrix-2x9> `prisms` polyhedra data table columns definition.

Definition at line 114 of file prisms.scad.

9.32.2.10 dtc_polyhedra_pyramids

<matrix-2x9> `pyramids` polyhedra data table columns definition.

Definition at line 114 of file pyramids.scad.

9.32.2.11 dtc_polyhedra_trapezohedron

<matrix-2x9> `trapezohedron` polyhedra data table columns definition.

Definition at line 114 of file trapezohedron.scad.

9.32.2.12 dtr_polyhedra_anti_prisms

<matrix-9xR> `anti_prisms` polyhedra data table rows.

Definition at line 129 of file anti_prisms.scad.

9.32.2.13 dtr_polyhedra_archimedean

<matrix-9xR> `archimedean` polyhedra data table rows.

Definition at line 129 of file archimedean.scad.

9.32.2.14 dtr_polyhedra_archimedean_duals

<matrix-9xR> `archimedean_duals` polyhedra data table rows.

Definition at line 129 of file archimedean_duals.scad.

9.32.2.15 dtr_polyhedra_cupolas

<matrix-9xR> `cupolas` polyhedra data table rows.

Definition at line 129 of file cupolas.scad.

9.32.2.16 dtr_polyhedra_dipyramids

<matrix-9xR> `dipyramids` polyhedra data table rows.

Definition at line 129 of file dipyramids.scad.

9.32.2.17 dtr_polyhedra_johnson

<matrix-9xR> `johnson` polyhedra data table rows.

Definition at line 129 of file johnson.scad.

9.32.2.18 dtr_polyhedra_platonic

<matrix-9xR> `platonic` polyhedra data table rows.

Definition at line 129 of file platonic.scad.

9.32.2.19 dtr_polyhedra_polyhedra_all

<matrix-9xR> `polyhedra_all` polyhedra data table rows.

Definition at line 140 of file polyhedra_all.scad.

9.32.2.20 dtr_polyhedra_prisms

<matrix-9xR> `prisms` polyhedra data table rows.

Definition at line 129 of file prisms.scad.

9.32.2.21 dtr_polyhedra_pyramids

<matrix-9xR> `pyramids` polyhedra data table rows.

Definition at line 129 of file pyramids.scad.

9.32.2.22 dtr_polyhedra_trapezohedron

<matrix-9xR> `trapezohedron` polyhedra data table rows.

Definition at line 129 of file trapezohedron.scad.

Table 7: johnson

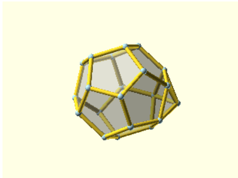
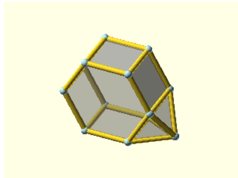
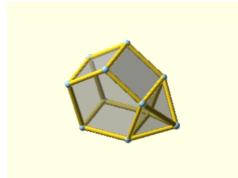
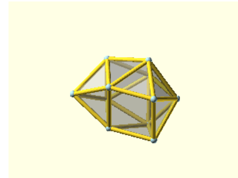
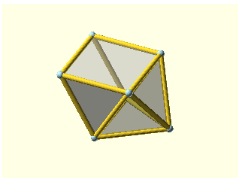
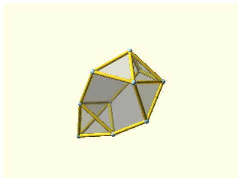
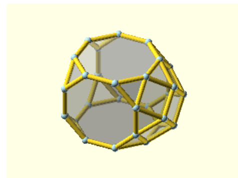
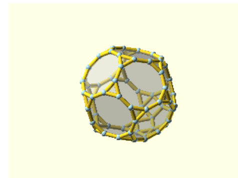
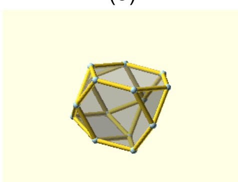
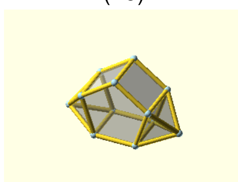
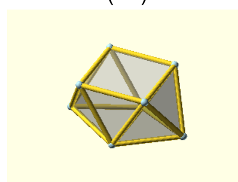
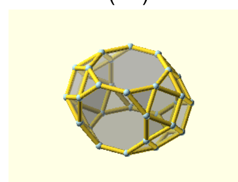
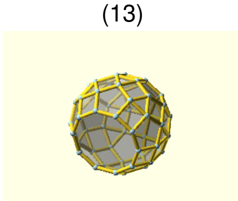
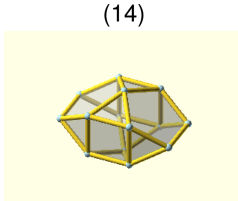
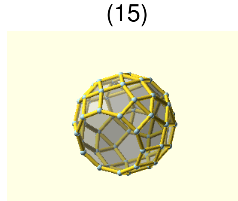
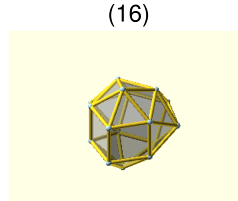
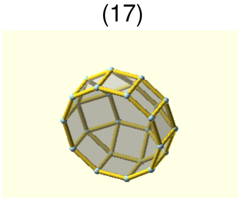
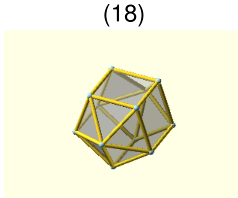
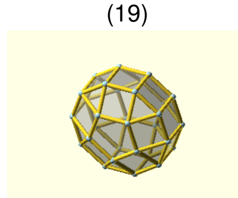
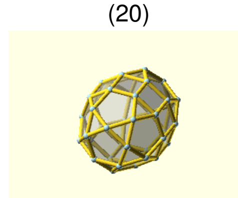
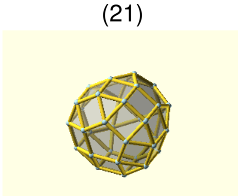
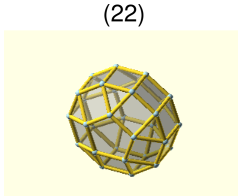
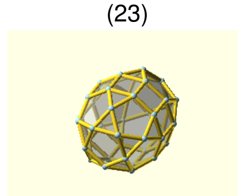
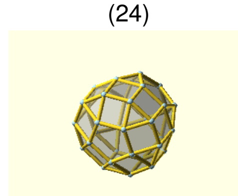
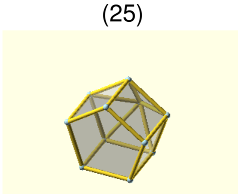
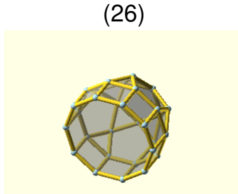
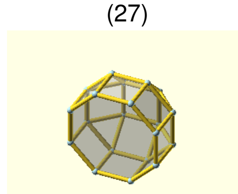
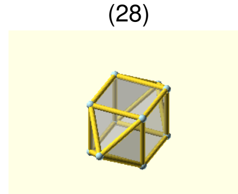
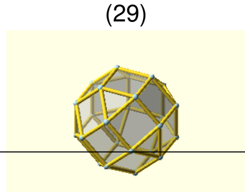
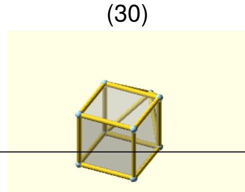
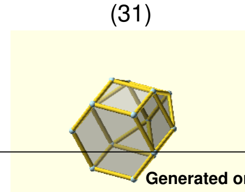
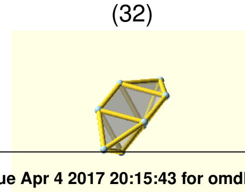
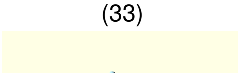
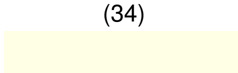
(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			
(9)	(10)	(11)	(12)
			
(13)	(14)	(15)	(16)
			
(17)	(18)	(19)	(20)
			
(21)	(22)	(23)	(24)
			
(25)	(26)	(27)	(28)
			
(29)	(30)	(31)	(32)
			
(33)	(34)	(35)	(36)
			

Table 8: platonic

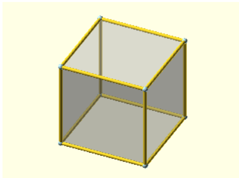
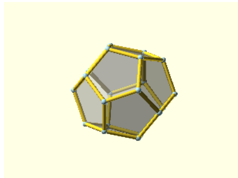
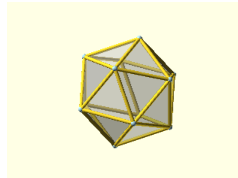
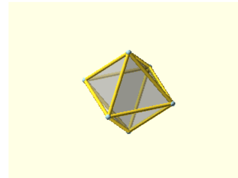
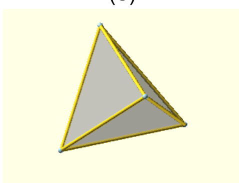
(1)

(2)

(3)

(4)

(5)


Table 9: prisms

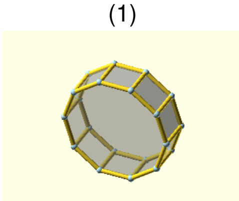
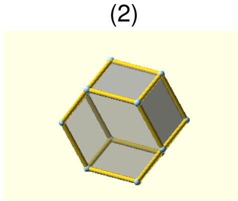
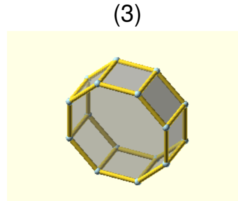
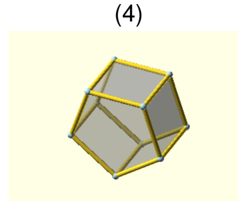
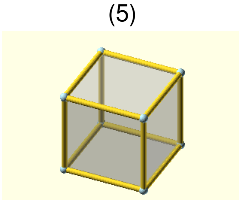
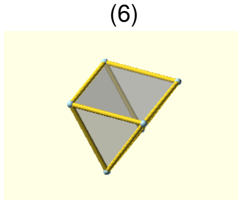
(1)

(2)

(3)

(4)

(5)

(6)


Table 10: pyramids

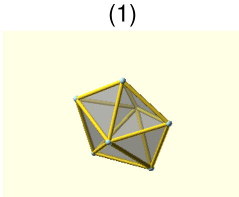
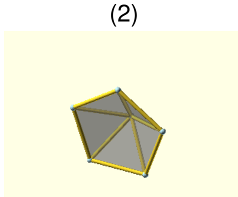
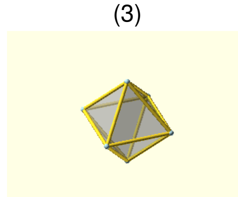
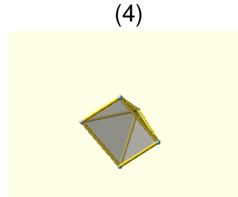
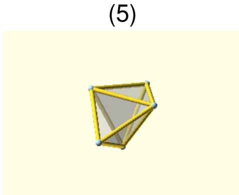
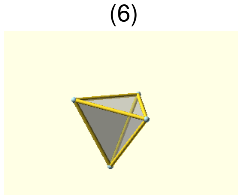
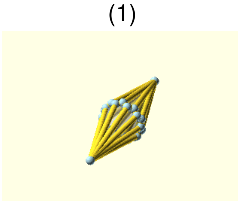
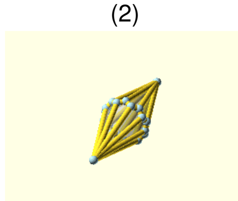
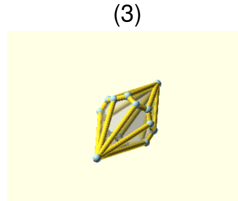
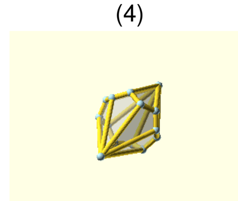
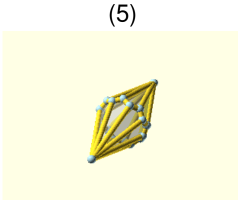
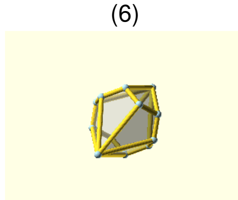
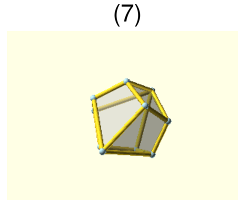
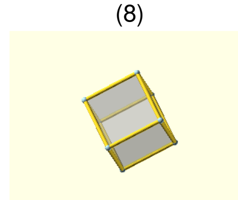
(1)

(2)

(3)

(4)

(5)

(6)


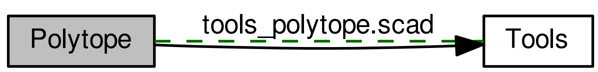
Table 11: trapezohedron

(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			

9.33 Polytope

Polygon and polyhedron tools.

Collaboration diagram for Polytope:



Files

- file tools_polytope.scad

  Polygon and polyhedron tools.

Functions

- module polytope_number (c, f, e, vi=true, fi=true, ei=true, sp=false, ts, th, to, tr=0)

  Label the vertices, paths, and edges of a polytope.
- module polytope_frame (c, f, e, vi=true, fi=true, ei=true, vc=1, fc=2, ec=0)

  Assemble a polytope skeletal frame using child objects.
- module polytope_bbox (c, f, a)

  The 3d or 2d bounding box shape for a polytope.

9.33.1 Detailed Description

Polygon and polyhedron tools.

9.33.2 Function Documentation

9.33.2.1 module polytope_bbox (c, f, a)

The 3d or 2d bounding box shape for a polytope.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d cartesian coordinates [[x, y (, z)], ...].
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
a	<decimal-list-1:3|decimal> The box padding. A list of lengths to equally pad the box dimensions.

Generates: (1) the 3d box shape that completely encloses the defined 3d polyhedron with the box sides oriented parallel to the coordinate axes. Or: (2) the 2d box shape that exactly encloses the defined 2d polygon with the box sides oriented parallel to the coordinate axes.

Note

When `f` is not given, the listed order of the coordinates `c` establishes the path.

See also

polytope_limits for warning about secondary Shapes.

Definition at line 298 of file tools_polytope.scad.

9.33.2.2 module polytope_frame (c , f , e , vi = `true`, fi = `true`, ei = `true`, vc = `1`, fc = `2`, ec = `0`)

Assemble a polytope skeletal frame using child objects.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.
vi	<index> Vertex index. An index sequence specification.
fi	<index> Face index. An index sequence specification.
ei	<index> Edge index. An index sequence specification.
vc	<integer> Vertex child index.
fc	<integer> Face child index.
ec	<integer> Edge child index.

This function constructs a skeletal frame for a given polytope. A 2d child object is linearly extruded along specified edges of the polytope to form the frame. Additional 3d child objects can be centered on specified vertices and/or the mean coordinates of specified faces.

Example

```
include <tools/tools_polytope.scad>;

s = second(xy_plane_os) * 25;
p = linear_extrude_pp2pf(s, h=50);

polytope_frame(first(p), second(p))
{
  circle(r=2);
  color("grey") sphere(r=4);
  color("blue") cube(4);
}
```

Note

To disable a child assignment to the vertices, faces, or edges, use an index that is less than zero or greater than the number of children.

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.

When `e` is not specified, it is computed from `f` using polytope_faces2edges().

Definition at line 216 of file tools_polytope.scad.

9.33.2.3 module polytope_number (c , f , e , vi = `true`, fi = `true`, ei = `true`, sp = `false`, ts , th , to , tr = `0`)

Label the vertices, paths, and edges of a polytope.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.
vi	<index> Vertex index. An index sequence specification.
fi	<index> Face index. An index sequence specification.
ei	<index> Edge index. An index sequence specification.
sp	<boolean> Show polyhedron shape.
ts	<decimal> The text size override.
th	<decimal> The text extrusion height override.
to	<vector-3d|vector-2d> The text offset override.
tr	<decimal-list-1:3|decimal> The text rotation (in degrees).

Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.
When `e` is not specified, it is computed from `f` using polytope_faces2edges().

Definition at line 78 of file tools_polytope.scad.

9.34 Polytopes

Polygon and polyhedron mathematical functions.

Collaboration diagram for Polytopes:

Files

- file math_polytope.scad

    Polygon and polyhedron mathematical functions.

Functions

- function polytope_faces2edges (f)

    List the edge coordinate index pairs of a polytope.
- function polytope_limits (c, f, a, d=[0:2], s=true)

    Determine the bounding limits of a polytope.
- function polytope_bbox_pf (c, f, a)

    Generate a bounding box polytope for another polytope in 3d or 2d.
- function polytope_line (c, f, e, i, l, r=false)

    Get a line from an edge or any two vetices of a polytope.
- function polytope_vertex_av (f, i)

    List the adjacent vertices for a given polytope vertex.
- function polytope_vertex_af (f, i)

    List the adjacent face indexes for a polytope vertex.
- function polytope_edge_af (f, e, i)

    List the adjacent face indexes for a polytope edge.
- function polytope_vertex_n (c, f, i)

    Get a normal vector for a polytope vertex.
- function polytope_edge_n (c, f, e, i)

    Get a normal vector for a polytope edge.
- function polytope_face_n (c, f, i, l, cw=true)

    Get the normal vector of a polytope face.
- function polytope_face_m (c, f, i, l)

    Get the mean coordinate of all vertices of a polytope face.
- function polytope_face_mn (c, f, i, l, cw=true)

    Get the mean coordinate and normal vector of a polytope face.
- function polytope_plane (c, f, i, l, cw=true)

Get a plane for a polytope face.
- function polytope_face_vcounts (f)
  List the vertex counts for all polytope faces.
- function polytope_face_angles (c, f)
  List the angles between all adjacent faces of a polyhedron.
- function polytope_edge_lengths (c, e)
  List the edge lengths of a polytope.
- function polytope_edge_angles (c, f)
  List the adjacent edge angles for each polytope vertex.
- function polytope_faces_are_regular (c, f, e, d=6)
  Test if the faces of a polytope are all regular.
- function polytope_triangulate_ft (f)
  Triangulate the faces of a convex polytope using fan triangulation.
- function polygon2d_perimeter (c, p)
  Calculate the perimeter length of a polygon in 2d.
- function polygon2d_area (c, p, s=false)
  Compute the signed area of a polygon in a Euclidean 2d-space.
- function polygon3d_area (c, p, n)
  Compute the area of a polygon in a Euclidean 3d-space.
- function polygon2d_centroid (c, p)
  Compute the center of mass of a polygon in a Euclidean 2d-space.
- function polygon2d_is_cw (c, p)
  Test the vertex ordering of a polygon in a Euclidean 2d-space.
- function polygon2d_is_convex (c, p)
  Test the convexity of a polygon in a Euclidean 2d-space.
- function polygon2d_winding (c, p, t)
  Compute the winding number of a polygon about a point in a Euclidean 2d-space.
- function polygon2d_is_pip_wn (c, p, t)
  Test if a point is inside a polygon in a Euclidean 2d-space using winding number.
- function polygon2d_is_pip_as (c, p, t)
  Test if a point is inside a polygon in a Euclidean 2d-space using angle summation.
- function polyhedron_area (c, f)
  Compute the surface area of a polyhedron in a Euclidean 3d-space.
- function polyhedron_volume_tf (c, f)
  Compute the volume of a triangulated polyhedron in a Euclidean 3d-space.
- function polyhedron_centroid_tf (c, f)
  Compute the center of mass of a triangulated polyhedron in a Euclidean 3d-space.
- function linear_extrude_pp2pf (c, p, h=1, centroid=false, center=false)
  Convert a polygon to a polyhedron by adding a height dimension.

9.34.1 Detailed Description

Polygon and polyhedron mathematical functions.

9.34.2 Function Documentation

9.34.2.1 function linear_extrude_pp2pf (c , p , h = `1`, centroid = `false`, center = `false`)

Convert a polygon to a polyhedron by adding a height dimension.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.
h	<decimal> The polyhedron height.
centroid	<boolean> Center polygon centroid at z-axis.
center	<boolean> Center polyhedron height about xy-plane.

Returns

<datastruct> A structure `[points, faces]`, where `points` are <coords-3d> and `faces` are a <integer-list-list>, that define the bounding box of the given polyhedron.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

9.34.2.2 function polygon2d_area (c, p, s = `false`)

Compute the signed area of a polygon in a Euclidean 2d-space.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.
s	<boolean> Return the vertex ordering sign.

Returns

<decimal> The area of the given polygon.

See `Wikipedia` for more information.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

Warning

This function does not track secondary shapes subtraction as implemented by the polygon() function.

9.34.2.3 function polygon2d_centroid (c, p)

Compute the center of mass of a polygon in a Euclidean 2d-space.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].

p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.

Returns

<point-2d> The center of mass of the given polygon.

See Wikipedia for more information.

Note

When p is not given, the listed order of the coordinates c establishes the path.

Warning

This function does not track secondary shapes subtraction as implemented by the polygon() function.

9.34.2.4 function polygon2d_is_convex (c , p)

Test the convexity of a polygon in a Euclidean 2d-space.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.

Returns

<boolean> **true** if the polygon is *convex*, **false** otherwise.

Note

When p is not given, the listed order of the coordinates c establishes the path.

9.34.2.5 function polygon2d_is_cw (c , p)

Test the vertex ordering of a polygon in a Euclidean 2d-space.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.

Returns

<boolean> **true** if the vertex are ordered *clockwise*, **false** if the vertex are *counterclockwise* ordered, and **undef** if the ordering can not be determined.

Note

When p is not given, the listed order of the coordinates c establishes the path.

9.34.2.6 function polygon2d_is_pip_as (c , p , t)

Test if a point is inside a polygon in a Euclidean 2d-space using angle summation.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.
t	<point-2d> A test point coordinate [x, y].

Returns

<boolean> **true** when the point is *inside* the polygon and **false** otherwise.

See `Wikipedia` for more information.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

Warning

This function does not track secondary shapes subtraction as implemented by the polygon() function.

9.34.2.7 function polygon2d_is_pip_wn (c , p , t)

Test if a point is inside a polygon in a Euclidean 2d-space using winding number.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.
t	<point-2d> A test point coordinate [x, y].

Returns

<boolean> **true** when the point is *inside* the polygon and **false** otherwise.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

See also

polygon2d_winding for warning about secondary Shapes.

9.34.2.8 function polygon2d_perimeter (c , p)

Calculate the perimeter length of a polygon in 2d.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.

Returns

<decimal> The sum of all polygon primary and secondary perimeter lengths.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

9.34.2.9 function polygon2d_winding (c , p , t)

Compute the winding number of a polygon about a point in a Euclidean 2d-space.

Parameters

c	<coords-2d> A list of 2d cartesian coordinates [[x, y], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.
t	<point-2d> A test point coordinate [x, y].

Returns

<integer> The winding number.

Computes the `winding number`, the total number of counterclockwise turns that the polygon paths makes around the test point in a Euclidean 2d-space. Will be 0 *iff* the point is outside of the polygon. Function patterned after `Dan Sunday, 2012`.

Copyright

Copyright 2000 softSurfer, 2012 Dan Sunday This code may be freely used and modified for any purpose providing that this copyright notice is included with it. iSurfer.org makes no warranty for this code, and cannot be held liable for any real or imagined damage resulting from its use. Users of this code must verify correctness for their application.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

Warning

Where there are secondary paths, the vertex ordering of each must be the same as the primary path.

9.34.2.10 function polygon3d_area (c , p , n)

Compute the area of a polygon in a Euclidean 3d-space.

Parameters

c	<coords-3d> A list of 3d cartesian coordinates [[x, y, z], ...].
p	<integer-list-list> An *optional* list of paths that define one or more closed shapes where each is a list of coordinate indexes.
n	<vector-3d> An *optional* normal vector, [x, y, z], to the polygon plane. When not given, a normal vector is constructed from the first three points of the primary path.

Returns

<decimal> The area of the given polygon.

Function patterned after `Dan Sunday, 2012`.

Note

When `p` is not given, the listed order of the coordinates `c` establishes the path.

Warning

This function does not track secondary shapes subtraction as implemented by the polygon() function.

9.34.2.11 function polyhedron_area (c , f)

Compute the surface area of a polyhedron in a Euclidean 3d-space.

Parameters

c	<coords-3d> A list of 3d cartesian coordinates [[x, y, z], ...].
f	<integer-list-list> A list of faces that enclose the shape where each face is a list of coordinate indexes.

Returns

<decimal> The surface area of the given polyhedron.

9.34.2.12 function polyhedron_centroid_tf (c , f)

Compute the center of mass of a triangulated polyhedron in a Euclidean 3d-space.

Parameters

c	<coords-3d> A list of 3d cartesian coordinates [[x, y, z], ...].
f	<integer-list-3-list> A list of triangular faces that enclose the polyhedron where each face is a list of three coordinate indexes.

Returns

<point-3d> The center of mass of the given polyhedron.

See `Wikipedia` for more information on centroid determined via the `divergence theorem` and midpoint quadrature.

Note

All faces are assumed to be a union of triangles oriented clockwise from the outside inwards.

9.34.2.13 function polyhedron_volume_tf (c , f)

Compute the volume of a triangulated polyhedron in a Euclidean 3d-space.

Parameters

c	<coords-3d> A list of 3d cartesian coordinates [[x, y, z], ...].
f	<integer-list-3-list> A list of triangular faces that enclose the polyhedron where each face is a list of three coordinate indexes.

Returns

<decimal> The volume of the given polyhedron.

See `Wikipedia` for more information on volumes determined using the `divergence theorem`.

Note

All faces are assumed to be a union of triangles oriented clockwise from the outside inwards.

9.34.2.14 function polytope_bbox_pf (c , f , a)

Generate a bounding box polytope for another polytope in 3d or 2d.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d cartesian coordinates [[x, y (, z)], ...].
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
a	<decimal-list-1:3|decimal> The box padding. A list of lengths to equally pad the box dimensions.

Returns

<datastruct> A structure: (1) `[points, faces]`, where `points` are <coords-3d> and `faces` are a <integer-list-list>, that define the bounding box of the given polyhedron. Or: (2) `[points, path]`, where `points` are <coords-2d> and `path` is a <integer-list-list>, that define the bounding box of the given polygon.

Polyhedron faces will be ordered *clockwise* when looking from outside the shape inwards. Polygon path will be ordered clockwise when looking from the top (positive z) downwards.

Note

When `f` is not specified, all coordinates are used to determine the geometric limits, which, simplifies the calculation. Parameter `f` is needed when a subset of the coordinates should be considered.

See also

polytope_limits for warning about secondary Shapes.

9.34.2.15 function polytope_edge_af (f , e , i)

List the adjacent face indexes for a polytope edge.

Parameters

f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.
i	<integer> The edge index.

Returns

<integer-list> The list of face indexes adjacent to the given polytope edge.

Note

When `e` is not specified, it is computed from `f` using polytope_faces2edges().

9.34.2.16 function polytope_edge_angles (c , f)

List the adjacent edge angles for each polytope vertex.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d cartesian coordinates [[x, y (, z)], ...].
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.

Returns

<decimal-list> A list of the polytope adjacent edge angles.

9.34.2.17 function polytope_edge_lengths (c , e)

List the edge lengths of a polytope.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d cartesian coordinates [[x, y (, z)], ...].
e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.

Returns

<decimal-list> A list of the polytope edge lengths.

9.34.2.18 function polytope_edge_n (c , f , e , i)

Get a normal vector for a polytope edge.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.

e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.
i	<integer> The edge index.

Returns

<vector-3d> A normal vector for the polytope edge.

The normal is computed as the mean of the adjacent faces.

Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.
When `e` is not specified, it is computed from `f` using polytope_faces2edges() iff the line is identified by `i`.

9.34.2.19 function polytope_face_angles (c , f)

List the angles between all adjacent faces of a polyhedron.

Parameters

c	<coords-3d> A list of 3d cartesian coordinates [[x, y, z], ...].
f	<integer-list-list> A list of faces that enclose the shape where each face is a list of coordinate indexes.

Returns

<decimal-list> A list of the polyhedron adjacent face angles.

See `Wikipedia` for more information on dihedral angles.

9.34.2.20 function polytope_face_m (c , f , i , l)

Get the mean coordinate of all vertices of a polytope face.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> The face specified as an face index.
l	<integer-list> The face specified as a list of all the coordinate indexes that define it.

Returns

<coords-3d> The mean coordinate of a polytope face.

The face can be identified using either parameter `i` or `l`. When using `l`, the parameter `f` is not required.

Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.

9.34.2.21 function polytope_face_mn (c , f , i , l , cw = `true`)

Get the mean coordinate and normal vector of a polytope face.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> The face specified as an face index.
l	<integer-list> The face specified as a list of all the coordinate indexes that define it.
cw	<boolean> Face vertex ordering.

Returns

<plane> `[mp, nv]`, where `mp` is `coords-3d`, the mean coordinate, and `nv` is `vector-3d`, the normal vector, of the polytope face-plane.

The face can be identified using either parameter `i` or `l`. When using `l`, the parameter `f` is not required.

Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.

9.34.2.22 function polytope_face_n (c , f , i , l , cw = `true`)

Get the normal vector of a polytope face.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> The face specified as an face index.
l	<integer-list> The face-plane specified as a list of three or more coordinate indexes that are a part of the face.
cw	<boolean> Face vertex ordering.

Returns

<vector-3d> The normal vector of a polytope face.

The face can be identified using either parameter `i` or `l`. When using `l`, the parameter `f` is not required.

Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.

9.34.2.23 function polytope_face_vcounts (f)

List the vertex counts for all polytope faces.

Parameters

f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.

Returns

<integer-list> A list with a vertex count of every face.

9.34.2.24 function polytope_faces2edges (f)

List the edge coordinate index pairs of a polytope.

Parameters

f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.

Returns

<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes that form the shape.

Note

Although the edge list is not sorted, each pair is sorted with the smallest index first.

9.34.2.25 function polytope_faces_are_regular (c , f , e , d = `6`)

Test if the faces of a polytope are all regular.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d cartesian coordinates [[x, y (, z)], ...].
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.
d	<integer> The number of significant figures used when comparing lengths and angles.

Returns

<boolean> **true** when there is both a single edge length and a single edge angle and **false** otherwise.

Note

When `e` is not specified, it is computed from `f` using polytope_faces2edges().

9.34.2.26 function polytope_limits (c , f , a , d = `[0:2]`, s = `true`)

Determine the bounding limits of a polytope.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d cartesian coordinates [[x, y (, z)], ...].
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
a	<decimal-list-1:3|decimal> The box padding. A list of lengths to equally pad the box dimensions.
d	<range|list|integer> The dimensions to consider. A range of dimensions, a list of dimensions, or a single dimension.
s	<boolean> Return box size rather than coordinate limits.

Returns

<datastruct> A list with the bounding-box limits (see: table).

The returned list will be of the following form:

	s	x	y	z	datastruct form
2d	false	[min,max]	[min,max]	-	decimal-list-2-list-2
2d	true	max-min	max-min	-	decimal-list-list-2
3d	false	[min,max]	[min,max]	[min,max]	decimal-list-2-list-3
3d	true	max-min	max-min	max-min	decimal-list-list-3

Note

When `f` is not specified, all coordinates are used to determine the geometric limits, which, simplifies the calculation. Parameter `f` is needed when a subset of the coordinates should be considered.

Warning

This function does not track secondary shapes subtraction as implemented by the polygon() function.

9.34.2.27 function polytope_line (c , f , e , i , l , r = `false`)

Get a line from an edge or any two vetices of a polytope.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
e	<integer-list-2-list> A list of edges where each edge is a list of two coordinate indexes.
i	<integer> A line specified as an edge index.
l	<integer-list-2> A line specified as a list of coordinate index pairs.
r	<boolean> Reverse the line start and end points.

Returns

<line-3d\|line-2d> The line as a pair of coordinates.

Note

Parameter f is optional for polygons. When it is not given, the listed order of the coordinates c establishes the polygon path.
When e is not specified, it is computed from f using polytope_faces2edges() iff the line is identified by i.

9.34.2.28 function polytope_plane (c , f , i , l , cw = true)

Get a plane for a polytope face.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> The face specified as an face index.
l	<integer-list> The face specified as a list of all the coordinate indexes that define it.
cw	<boolean> Face vertex ordering.

Returns

<plane> [mp, nv], where mp is coords-3d, the mean coordinate, and nv is vector-3d, the normal vector, of the polytope face-plane.

The face can be identified using either parameter i or l. When using l, the parameter f is not required.

Note

Parameter f is optional for polygons. When it is not given, the listed order of the coordinates c establishes the polygon path.

9.34.2.29 function polytope_triangulate_ft (f)

Triangulate the faces of a convex polytope using fan triangulation.

Parameters

f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.

Returns

<integer-list-3-list> A list of triangular faces that enclose the polytope where each face is a list of three coordinate indexes with vertex ordering is maintained.

See Wikipedia for more information on fan triangulation.

Warning

This method does not support concave polytopes.

9.34.2.30 function polytope_vertex_af (f , i)

List the adjacent face indexes for a polytope vertex.

Parameters

f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> The vertex index.

Returns

<integer-list> The list of face indexes adjacent to the given polytope vertex.

9.34.2.31 function polytope_vertex_av (f , i)

List the adjacent vertices for a given polytope vertex.

Parameters

f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> A vertex index.

Returns

<integer-list> The list of adjacent vertex indexes for the given vertex index.

The adjacent vertices are those neighboring vertices that are directly connected to the given vertex by a common edge.

Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.

9.34.2.32 function polytope_vertex_n (c , f , i)

Get a normal vector for a polytope vertex.

Parameters

c	<coords-3d|coords-2d> A list of 3d or 2d coordinate points.
f	<integer-list-list> A list of faces (or paths) that enclose the shape where each face is a list of coordinate indexes.
i	<integer> The vertex index.

Returns

<vector-3d> A normal vector for the polytope vertex.

The normal is computed as the mean of the adjacent faces.

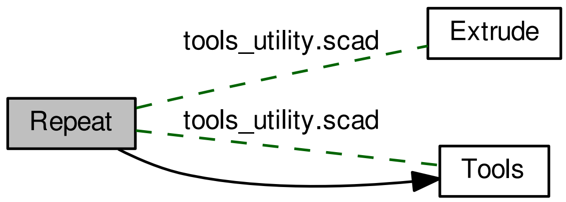
Note

Parameter `f` is optional for polygons. When it is not given, the listed order of the coordinates `c` establishes the polygon path.

9.35 Repeat

Shape repetition and distribution tools.

Collaboration diagram for Repeat:



Files

- file tools_utility.scad

  Shape transformation utility tools.

Functions

- module radial_repeat (n, r=1, angle=true, move=false)

  Distribute copies of a 2d or 3d shape equally about a z-axis radius.
- module grid_repeat (g, i, c=1, center=false)

  Distribute copies of 2d or 3d shapes about Cartesian grid.

9.35.1 Detailed Description

Shape repetition and distribution tools.

9.35.2 Function Documentation

9.35.2.1 module grid_repeat (g , i , c = 1, center = false)

Distribute copies of 2d or 3d shapes about Cartesian grid.

Parameters

g	<integer-list-3|integer> The grid division count. A list [x, y, z] of integers or a single integer for (x=y=z).

i	<decimal-list-3|decimal> The grid increment size. A list [x, y, z] of decimals or a single decimal for (x=y=z).
c	<integer> The number of copies. Number of times to iterate over children.
center	<boolean> Center distribution about origin.

Example

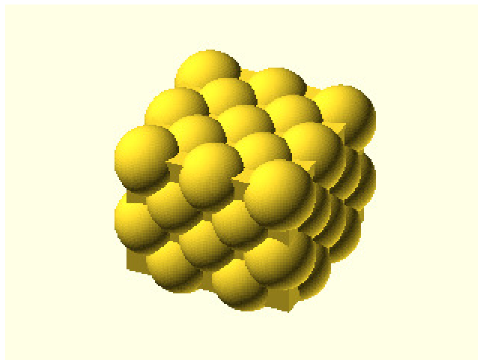


Figure 51: grid_repeat

```
grid_repeat( g=[5,5,4], i=10, c=50, center=true ) {cube(10, center=true); sphere(10);}
```

Definition at line 356 of file tools_utility.scad.

9.35.2.2 module radial_repeat (n , r = 1, angle = true, move = false)

Distribute copies of a 2d or 3d shape equally about a z-axis radius.

Parameters

n	<integer> The number of equally spaced radii.
r	<decimal> The shape move radius.
angle	<boolean> Rotate each copy about z-axis.
move	<boolean> Move each shape copy to radii coordinate.

Example

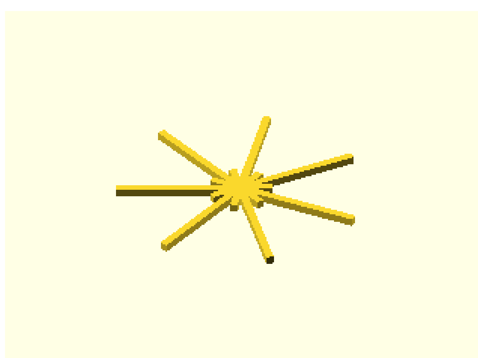


Figure 52: radial_repeat

```
radial_repeat( n=7, r=6, move=true ) square( [20,1], center=true );
```

Definition at line 325 of file tools_utility.scad.

9.36 Resolutions

Arch rendering resolution management.

Collaboration diagram for Resolutions:

Files

- file units_resolution.scad

  An abstraction for arc rendering resolution control.

Functions

- function resolution_fn (r)

  Return facets number for the given arc radius.
- function resolution_fs ()

  Return minimum facets size.
- function resolution_fa (r)

  Return the minimum facets angle.
- function resolution_reduced ()

  Return the radius at which arc resolution will begin to degrade.
- module resolution_info (r)

  Output resolution information to the console for given radius.
- function resolution_facets (r)

  Return facet count used to render a radius.
- function resolution_facetsv (r)

  Return facet count information list used to render a radius.

Variables

- $resolution_mode = "fast"

  <string> Global special variable that configures the arc resolution mode.
- $resolution_value = 0

  <number> Global special variable for modes that use custom resolutions.

9.36.1 Detailed Description

Arch rendering resolution management.

Functions, global variables, and configuration presets to provide a common mechanism for managing arc rendering resolution. Specifically, the number of fragments/facets with which arcs (circles, spheres, and cylinders, etc.) are rendered in OpenSCAD.

Example

```
include <units/units_resolution.scad>;

base_unit_length = "in";

// set resolution to 25 fpi
$resolution_mode  = "fpi";
$resolution_value = 25;

// use radius length of 1 inch
r = convert_length(1, "in");

$fs=resolution_fs();
$fa=resolution_fa( r );

resolution_info( r );

f = resolution_facets( r );
echo(str("for r = ", r, " ", unit_length_name(), ", facets = ", f));
```

Result (base_unit_length = **mm**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = millimeter"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 1.016"
ECHO: "resolution reduction at radius > 25.4 millimeter"
ECHO: "for radius = 25.4 millimeter facets limited to 158 by $fs=1.016 millimeter"
ECHO: "for r = 25.4 millimeter, facets = 158"
```

Result (base_unit_length = **cm**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = centimeter"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 0.1016"
ECHO: "resolution reduction at radius > 2.54 centimeter"
ECHO: "for radius = 2.54 centimeter facets limited to 158 by $fs=0.1016 centimeter"
ECHO: "for r = 2.54 centimeter, facets = 158"
```

Result (base_unit_length = **mil**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = thousandth"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 40"
ECHO: "resolution reduction at radius > 1000 thousandth"
ECHO: "for radius = 1000 thousandth facets limited to 158 by $fs=40 thousandth"
ECHO: "for r = 1000 thousandth, facets = 158"
```

Result (base_unit_length = **in**):

```
ECHO: "$resolution_mode = [fpi], $resolution_value = 25, base_unit_length = inch"
ECHO: "$fn = 0, $fa = 2.29183, $fs = 0.04"
ECHO: "resolution reduction at radius > 1 inch"
ECHO: "for radius = 1 inch facets limited to 158 by $fs=0.04 inch"
ECHO: "for r = 1 inch, facets = 158"
```

9.36.2 Function Documentation

9.36.2.1 function resolution_fa (r)

Return the minimum facets angle.

Parameters

r	<decimal> The arc radius.

Returns

<decimal> Minimum facet angle to be assigned to $fa.

The result of this function can be assigned to the special variables `$fa` to render arcs.

9.36.2.2 function resolution_facets (r)

Return facet count used to render a radius.

Parameters

r	<decimal> The arc radius.

Returns

<integer> The number of fragments/facets that will be used to render a radius given the current values for `$fn`, `$fa`, and `$fs`.

9.36.2.3 function resolution_facetsv (r)

Return facet count information list used to render a radius.

Parameters

r	<decimal> The arc radius.

Returns

<list-3> A 3-tuple list of the form: [**facets** <integer>,**limiter** <string>,**value** <decimal>].

Where `facets` is the number of fragments/facets that will be used to render the `radius` given the current values for `$fn`, `$fa`, and `$fs`. `limiter` identifies the special variable that currently limits the facets, and `value` is the current value assigned to the limiter.

9.36.2.4 function resolution_fn (r)

Return facets number for the given arc radius.

Parameters

r	<decimal> The arc radius.

Returns

<integer> The number of facets to be assigned to $fn.

The result of this function can be assigned to the special variables `$fn` to render arcs according to the resolution mode set by $resolution_mode and $resolution_value.

The following table shows the modes that require $resolution_value to be set prior to specifying the custom values used during resolution calculation.

$resolution_mode	$resolution_value sets	radius dependent
set	fixed value	no
upf	units per facet	yes
fpu	facets per unit	yes
fpi	facets per inch	yes

The following table has common resolution presets. Equivalent configuration can be obtained using $resolution_mode and $resolution_value as described in the preview table.

$resolution_mode	preset description	radius dependent
fast	fast rendering mode	no
low	low resolution	yes
medium	medium resolution	yes
high	high resolution	yes
50um	50 micron per facets	yes
100um	100 micron per facets	yes
200um	200 micron per facets	yes
300um	300 micron per facets	yes
400um	400 micron per facets	yes
500um	500 micron per facets	yes
50mil	50 thousandth per facets	yes
100mil	100 thousandth per facets	yes
200mil	200 thousandth per facets	yes
300mil	300 thousandth per facets	yes
400mil	400 thousandth per facets	yes
500mil	500 thousandth per facets	yes

9.36.2.5 function resolution_fs ( )

Return minimum facets size.

Returns

<integer> Minimum facet size to be assigned to $fs.

The result of this function can be assigned to the special variables `$fs` to render arcs according to the resolution mode set by $resolution_mode and $resolution_value.

The following table shows the modes that require $resolution_value to be set prior to calling this function in order to specify the custom values used during resolution calculation.

$resolution_mode	$resolution_value sets	radius dependent
set	fixed value	no
upf	units per facet	no
fpu	facets per unit	no
fpi	facets per inch	no

The following table has common resolution presets. Equivalent configuration can be obtained using $resolution_mode and $resolution_value as described in the preview table.

$resolution_mode	preset description	radius dependent
fast	fast rendering mode	no
low	low resolution	no
medium	medium resolution	no

high	high resolution	no
50um	50 micron per facets	no
100um	100 micron per facets	no
200um	200 micron per facets	no
300um	300 micron per facets	no
400um	400 micron per facets	no
500um	500 micron per facets	no
50mil	50 thousandth per facets	no
100mil	100 thousandth per facets	no
200mil	200 thousandth per facets	no
300mil	300 thousandth per facets	no
400mil	400 thousandth per facets	no
500mil	500 thousandth per facets	no

9.36.2.6 module resolution_info (r)

Output resolution information to the console for given radius.

Parameters

r	<decimal> The arc radius.

Definition at line 324 of file units_resolution.scad.

9.36.2.7 function resolution_reduced ( )

Return the radius at which arc resolution will begin to degrade.

Returns

<decimal> Transition radius where resolution reduction begins.

The special variables `$fs` and `$fa` work together when `$fn` = 0. For a given `$fs`, the fragment angle of a drawn arc gets smaller with increasing radius. In other words, the fragment angle is inversely proportional to the arc radius for a given fragment size.

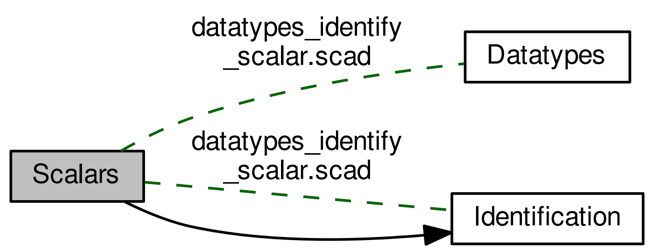
The special variable `$fa` enforces a minimum fragment angle limit and at some radius, the fragment angle would becomes smaller than this limit. At this point, OpenSCAD limits further reduction in the facet angle which forces the use of increased fragment size. This in effect begins the gradual reduction of arc resolution with increasing radius.

The return result of this function indicates the radius at which this enforced limiting begins. When `$fn` != 0, returns **undef**.

9.37 Scalars

Scalar data type identification.

Collaboration diagram for Scalars:



Files

- file datatypes_identify_scalar.scad

    Scalar data type identification.

Functions

- function is_defined (v)

    Test if a value is defined.
- function not_defined (v)

    Test if a value is not defined.
- function is_nan (v)

    Test if a numerical value is invalid.
- function is_inf (v)

    Test if a numerical value is infinite.
- function is_scalar (v)

    Test if a value is a single non-iterable value.
- function is_iterable (v)

    Test if a value has multiple parts and is iterable.
- function is_empty (v)

    Test if an iterable value is empty.
- function is_number (v)

    Test if a value is a number.
- function is_integer (v)

    Test if a value is an integer.
- function is_decimal (v)

    Test if a value is a decimal.
- function is_boolean (v)

Test if a value is a predefined boolean constant.

- function is_string (v)

  Test if a value is a string.
- function is_list (v)

  Test if a value is an iterable list of values.
- function is_range (v)

  Test if a value is a range definition.
- function is_even (v)

  Test if a numerical value is even.
- function is_odd (v)

  Test if a numerical value is odd.
- function is_between (v, l, u)

  Test if a numerical value is between an upper and lower bounds.

9.37.1 Detailed Description

Scalar data type identification.

See validation results.

9.37.2 Function Documentation

9.37.2.1 function is_between (v , l , u)

Test if a numerical value is between an upper and lower bounds.

Parameters

v	<number> A numerical value.
l	<number> The minimum value.
u	<number> The maximum value.

Returns

<boolean> **true** when the value is between the upper and lower bounds and **false** otherwise.

9.37.2.2 function is_boolean (v)

Test if a value is a predefined boolean constant.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is one of the predefined boolean constants `[true|false]` and **false** otherwise.

9.37.2.3 function is_decimal (v)

Test if a value is a decimal.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a decimal and **false** otherwise.

9.37.2.4 function is_defined (v)

Test if a value is defined.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is defined and **false** otherwise.

9.37.2.5 function is_empty (v)

Test if an iterable value is empty.

Parameters

v	<value> An iterable value.

Returns

<boolean> **true** when the iterable value has zero elements and **false** otherwise.

9.37.2.6 function is_even (v)

Test if a numerical value is even.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be *even* and **false** otherwise (The value may be positive or negative).

9.37.2.7 function is_inf (v)

Test if a numerical value is infinite.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be **inf** (greater than the largest representable number) and **false** otherwise.

9.37.2.8 function is_integer (v)

Test if a value is an integer.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is an integer and **false** otherwise.

9.37.2.9 function is_iterable (v)

Test if a value has multiple parts and is iterable.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is an iterable multi-part value and **false** otherwise.

data type	defined
number(s)	**false**
boolean	**false**
string	**true**
list	**true**
range	not defined
undef	**false**
inf	**false**
nan	**false**

9.37.2.10 function is_list (v)

Test if a value is an iterable list of values.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a list and **false** otherwise.

9.37.2.11 function is_nan (v)

Test if a numerical value is invalid.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be **nan** (Not A Number) and **false** otherwise.

9.37.2.12 function is_number (v)

Test if a value is a number.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a number and **false** otherwise.

Note

Returns **true** for **inf** and **nan** values.

9.37.2.13 function is_odd (v)

Test if a numerical value is odd.

Parameters

v	<value> A numerical value.

Returns

<boolean> **true** when the value is determined to be *odd* and **false** otherwise (The value may be positive or negative).

9.37.2.14 function is_range (v)

Test if a value is a range definition.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a range definition and **false** otherwise.

9.37.2.15 function is_scalar (v)

Test if a value is a single non-iterable value.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a single non-iterable value and **false** otherwise.

data type	defined
number(s)	**true**
boolean	**true**

string	**false**
list	**false**
range	not defined
undef	**true**
inf	**true**
nan	**true**

9.37.2.16 function is_string (v)

Test if a value is a string.

Parameters

v	<value> A value.

Returns

<boolean> **true** when the value is a string and **false** otherwise.

9.37.2.17 function not_defined (v)

Test if a value is not defined.

Parameters

v	<value> A value.

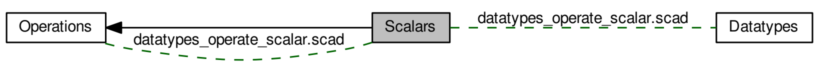
Returns

<boolean> **true** when the value is not defined and **false** otherwise.

9.38 Scalars

Scalar data type operations.

Collaboration diagram for Scalars:



Files

- file datatypes_operate_scalar.scad

  Scalar data type operations.

Functions

- function defined_or (v, d)

  Return a value when it is defined or a default value when it is not.
- function circular_index (i, l, f=0)

  Map an index position into a circularly indexed list.

9.38.1 Detailed Description

Scalar data type operations.

See validation results.

9.38.2 Function Documentation

9.38.2.1 function circular_index (i, l, f = 0)

Map an index position into a circularly indexed list.

Parameters

i	<integer> Any index, in or out of bounds.
l	<integer> The circular list length.
f	<integer> The starting index number.

Returns

<integer> A index position in the circular list within the range `[f : l+f-1]`.

9.38.2.2 function defined_or (v, d)

Return a value when it is defined or a default value when it is not.

Parameters

v	<value> A value.
d	<value> A default value.

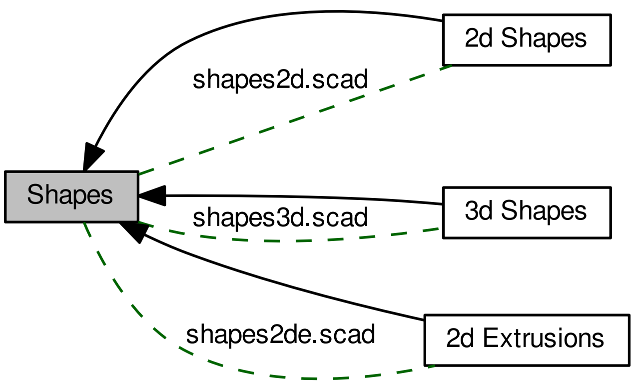
Returns

<value> `v` when it is defined and `d` otherwise.

9.39 Shapes

2d and 3d shapes.

Collaboration diagram for Shapes:



Modules

- [2d Extrusions]

    Extruded two-dimensional geometric shapes.
- [2d Shapes]

    Two-dimensional geometric shapes.
- [3d Shapes]

    Three-dimensional geometric shapes.

Files

- file shapes2d.scad

    Two-dimensional basic shapes.
- file shapes2de.scad

    Linearly extruded two-dimensional basic shapes.
- file shapes3d.scad

    Three-dimensional basic shapes.

9.39.1 Detailed Description

2d and 3d shapes.

Table 12: 2d Shapes

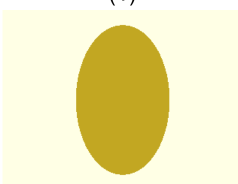
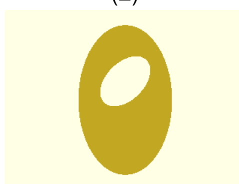
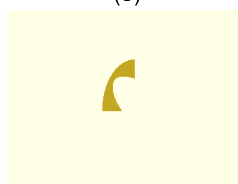
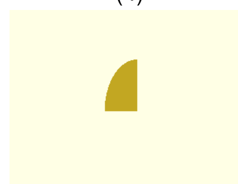
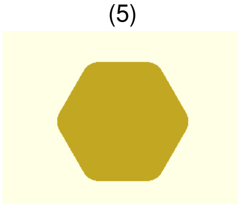
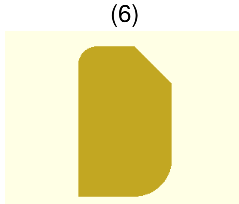
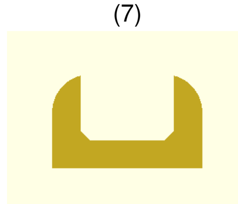
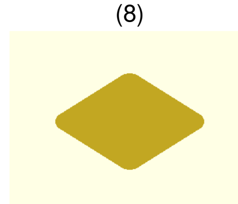
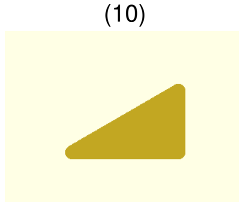
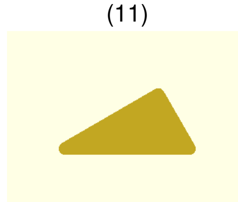
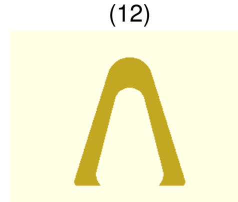
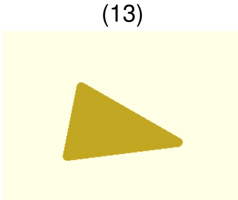
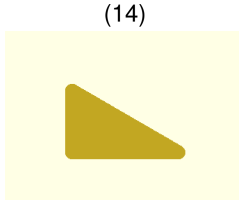
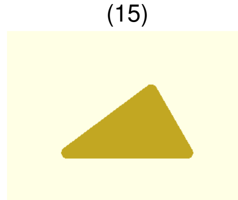
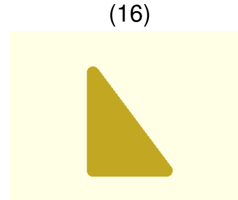
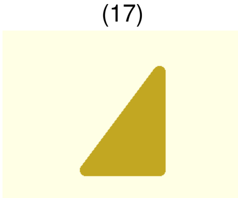
(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			
(9)	(10)	(11)	(12)
			
(13)	(14)	(15)	(16)
			
(17)			
			

Table 13: 2d Extrusions

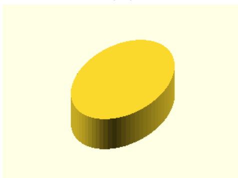
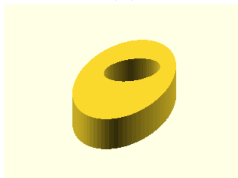
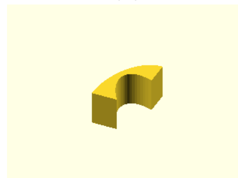
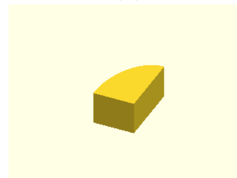
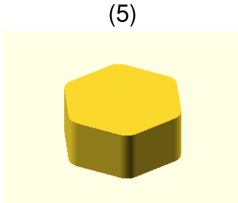
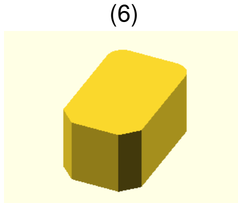
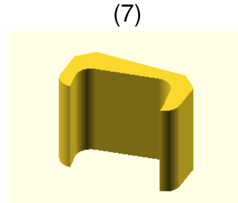
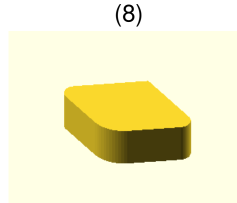
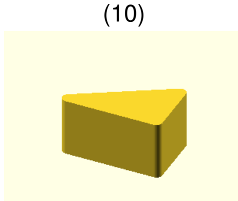
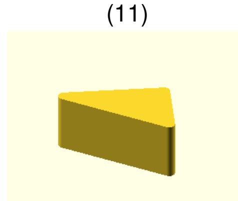
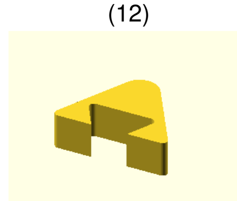
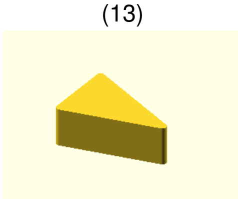
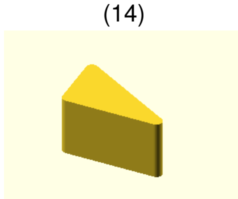
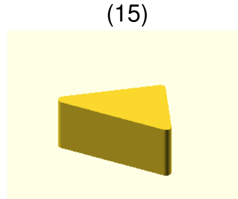
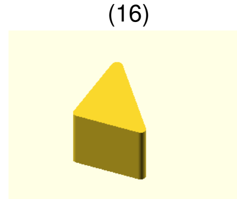
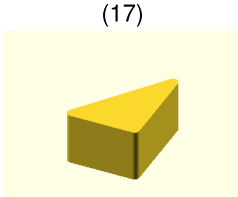
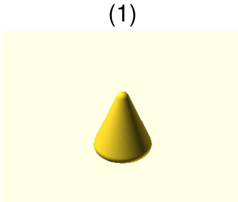
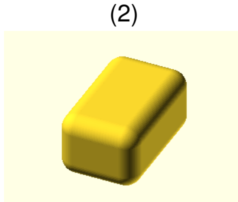
(1)	(2)	(3)	(4)
			
(5)	(6)	(7)	(8)
			
(9)	(10)	(11)	(12)
			
(13)	(14)	(15)	(16)
			
(17)			
			

Table 14: 3d Shapes

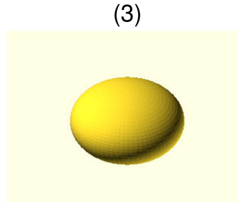
(1)



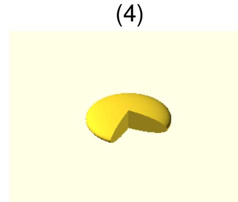
(2)



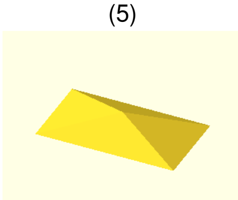
(3)



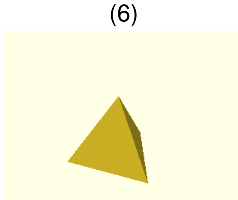
(4)



(5)

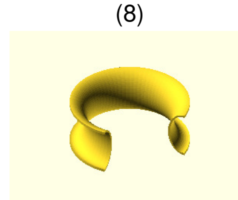


(6)

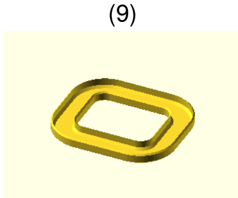


(7)

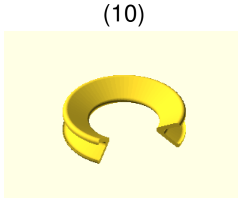
(8)



(9)



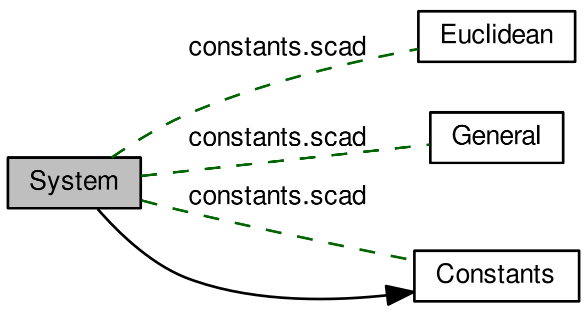
(10)



9.40 System

System and program limits.

Collaboration diagram for System:



Files

- file constants.scad

  Design constant definitions.

Variables

- number_max = 1e308

  <decimal> The largest representable number in OpenSCAD scripts.
- number_min = -1e308

  <decimal> The smallest representable number in OpenSCAD scripts.
- empty_str = ""

  <string> A string with no characters (the empty string).
- empty_lst = [ ]

  <list> A list with no values (the empty list).

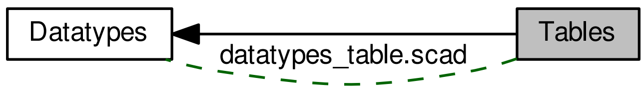
9.40.1 Detailed Description

System and program limits.

9.41 Tables

Table data type operations.

Collaboration diagram for Tables:



Files

- file datatypes_table.scad

    Table data type operations.

Functions

- function get_table_ri (r, ri)

    Get the table row index that matches a table row identifier.
- function get_table_r (r, ri)

    Get the table row that matches a table row identifier.
- function get_table_ci (c, ci)

    Get the table column index that matches a table column identifier.
- function get_table_c (c, ci)

    Get the table column that matches a table column identifier.
- function get_table_v (r, c, ri, ci)

    Get the table cell value for a specified row and column identifier.
- function get_table_crl (r, c, ci)

    Form a list of a select column across all table rows.
- function get_table_ridl (r)

    Form a list of all table row identifiers.
- function get_table_cidl (c)

    Form a list of all table column identifiers.
- function table_exists (r, c, ri, ci)

    Test the existence of a table row and column identifier.
- function get_table_size (r, c)

    Get the size of a table.
- function get_table_copy (r, c, rl, cl)

    Create a new matrix from select rows and columns of a table.
- function get_table_sum (r, c, rl, cl)

    Sum select rows and columns of a table.
- module table_check (r, c, verbose=false)

Perform some basic validation/checks on a table.

- module table_dump (r, c, rl, cl, number=true)

  Dump a table to the console.

9.41.1 Detailed Description

Table data type operations.

Example

```
use     <datatypes/datatypes_table.scad>;

base_unit_length = "mm";

table_cols =
[ // id,  description
  ["id",  "row identifier"],
  ["ht",  "head type [r|h|s]"],
  ["td",  "thread diameter"],
  ["tl",  "thread length"],
  ["hd",  "head diameter"],
  ["hl",  "head length"],
  ["nd",  "hex nut flat-to-flat width"],
  ["nl",  "hex nut length"]
];

table_rows =
[ //      id,  ht,     td,     tl,   hd,    hl,    nd,                  nl
  ["m3r08r", "r",  3.000,   8.00, 5.50, 3.000,  5.50, convert_length(1.00, "in")],
  ["m3r14r", "r",  3.000,  14.00, 5.50, 3.000,  5.50, convert_length(1.25, "in")],
  ["m3r16r", "r",  3.000,  16.00, 5.50, 3.000,  5.50, convert_length(1.50, "in")],
  ["m3r20r", "r",  3.000,  20.00, 5.50, 3.000,  5.50, convert_length(1.75, "in")]
];

table_check( table_rows, table_cols, true );
table_dump( table_rows, table_cols );

m3r16r_tl = get_table_v( table_rows, table_cols, "m3r16r", "tl" );

if ( table_exists( c=table_cols, ci="nl" ) )
  echo ( "metric 'nl' available" );

table_ids = get_table_ridl( table_rows );
table_cols_tl = get_table_crl( table_rows, table_cols, "tl" );

echo ( table_ids=table_ids );
echo ( table_cols_tl=table_cols_tl );

tnew = get_table_copy( table_rows, table_cols, cl=["tl", "nl"] );
tsum = get_table_sum( table_rows, table_cols, cl=["tl", "nl"] );

echo ( m3r16r_tl=m3r16r_tl );
echo ( tnew=tnew );
echo ( tsum=tsum );
```

Result

```
ECHO: "[ INFO ] table_check(); begin table check"
ECHO: "[ INFO ] table_check(); row identifier found at column zero."
ECHO: "[ INFO ] table_check(); checking row column counts."
ECHO: "[ INFO ] table_check(); checking for repeat column identifiers."
ECHO: "[ INFO ] table_check(); checking for repeat row identifiers."
ECHO: "[ INFO ] table_check(); table size: 4 rows by 8 columns."
ECHO: "[ INFO ] table_check(); end table check"
ECHO: ""
ECHO: "row: 0"
ECHO: "[m3r08r] [id] (row identifier)             = [m3r08r]"
ECHO: "[m3r08r] [ht] (head type [r|h|s])          = [r]"
ECHO: "[m3r08r] [td] (thread diameter)            = [3]"
ECHO: "[m3r08r] [tl] (thread length)              = [8]"
ECHO: "[m3r08r] [hd] (head diameter)              = [5.5]"
ECHO: "[m3r08r] [hl] (head length)                = [3]"
```

```
ECHO: "[m3r08r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r08r] [nl] (hex nut length)             = [25.4]"
ECHO: ""
ECHO: "row: 1"
ECHO: "[m3r14r] [id] (row identifier)             = [m3r14r]"
ECHO: "[m3r14r] [ht] (head type [r|h|s])          = [r]"
ECHO: "[m3r14r] [td] (thread diameter)            = [3]"
ECHO: "[m3r14r] [tl] (thread length)              = [14]"
ECHO: "[m3r14r] [hd] (head diameter)              = [5.5]"
ECHO: "[m3r14r] [hl] (head length)                = [3]"
ECHO: "[m3r14r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r14r] [nl] (hex nut length)             = [31.75]"
ECHO: ""
ECHO: "row: 2"
ECHO: "[m3r16r] [id] (row identifier)             = [m3r16r]"
ECHO: "[m3r16r] [ht] (head type [r|h|s])          = [r]"
ECHO: "[m3r16r] [td] (thread diameter)            = [3]"
ECHO: "[m3r16r] [tl] (thread length)              = [16]"
ECHO: "[m3r16r] [hd] (head diameter)              = [5.5]"
ECHO: "[m3r16r] [hl] (head length)                = [3]"
ECHO: "[m3r16r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r16r] [nl] (hex nut length)             = [38.1]"
ECHO: ""
ECHO: "row: 3"
ECHO: "[m3r20r] [id] (row identifier)             = [m3r20r]"
ECHO: "[m3r20r] [ht] (head type [r|h|s])          = [r]"
ECHO: "[m3r20r] [td] (thread diameter)            = [3]"
ECHO: "[m3r20r] [tl] (thread length)              = [20]"
ECHO: "[m3r20r] [hd] (head diameter)              = [5.5]"
ECHO: "[m3r20r] [hl] (head length)                = [3]"
ECHO: "[m3r20r] [nd] (hex nut flat-to-flat width) = [5.5]"
ECHO: "[m3r20r] [nl] (hex nut length)             = [44.45]"
ECHO: ""
ECHO: "table size: 4 rows by 8 columns."
ECHO: "metric 'nl' available"
ECHO: table_ids = ["m3r08r", "m3r14r", "m3r16r", "m3r20r"]
ECHO: table_cols_tl = [8, 14, 16, 20]
ECHO: m3r16r_tl = 16
ECHO: tnew = [[8, 25.4], [14, 31.75], [16, 38.1], [20, 44.45]]
ECHO: tsum = [58, 139.7]
```

9.41.2 Function Documentation

9.41.2.1 function get_table_c (c , ci)

Get the table column that matches a table column identifier.

Parameters

c	<matrix-2xC> The table column matrix (2 x C-columns).
ci	<string> The column identifier.

Returns

<list-2> The table column where the column identifier exists. If the identifier does not exists, returns **undef**.

9.41.2.2 function get_table_ci (c , ci)

Get the table column index that matches a table column identifier.

Parameters

c	<matrix-2xC> The table column matrix (2 x C-columns).
ci	<string> The column identifier.

Returns

<integer> The column index where the identifier exists. If the identifier does not exists, returns **empty_lst**.

9.41.2.3 function get_table_cidl (c)

Form a list of all table column identifiers.

Parameters

c	<matrix-2xC> The table column matrix (2 x C-columns).

Returns

<list> The list of all column identifiers.

Note

This functions assumes the first element of each table column to be the column identifier.

9.41.2.4 function get_table_copy (r , c , rl , cl)

Create a new matrix from select rows and columns of a table.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
rl	<string-list> A list of selected row identifiers.
cl	<string-list> A list of selected column identifiers.

Returns

<matrix> A matrix of the selected rows and columns.

9.41.2.5 function get_table_crl (r , c , ci)

Form a list of a select column across all table rows.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
ci	<string> The column identifier.

Returns

<list> The list of a select column across all rows. If the identifier does not exists, returns **undef**.

9.41.2.6 function get_table_r (r , ri)

Get the table row that matches a table row identifier.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
ri	<string> The row identifier.

Returns

<list-C> The table row where the row identifier exists. If the identifier does not exists, returns **undef**.

9.41.2.7 function get_table_ri (r , ri)

Get the table row index that matches a table row identifier.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
ri	<string> The row identifier.

Returns

<integer> The row index where the identifier exists. If the identifier does not exists, returns **empty_lst**.

9.41.2.8 function get_table_ridl (r)

Form a list of all table row identifiers.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).

Returns

<list> The list of all row identifiers.

Note

This functions assumes the first element of each table row to be the row identifier, as enforced by the table_check(). As an alternative, the function get_table_crl(), of the form get_table_crl(r, c, "id"), may be used without this assumption.

9.41.2.9 function get_table_size (r, c)

Get the size of a table.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).

Returns

<decimal> The table size.

The size is reported as: (1) The number of rows when only the `r` parameter is specified. (2) The number of columns when only the `c` parameter is specified. (3) The (r $*$ columns) when both parameters are specified.

9.41.2.10 function get_table_sum (r, c, rl, cl)

Sum select rows and columns of a table.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
rl	<string-list> A list of selected row identifiers.

cl	<string-list> A list of selected column identifiers.

Returns

<list> A list with the sum of each selected rows and columns.

9.41.2.11 function get_table_v (r, c, ri, ci)

Get the table cell value for a specified row and column identifier.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
ri	<string> The row identifier.
ci	<string> The column identifier.

Returns

<value> The value of the matrix cell [ri, ci]. If either identifier does not exists, returns **undef**.

9.41.2.12 module table_check (r, c, verbose = `false`)

Perform some basic validation/checks on a table.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
verbose	<boolean> Be verbose during check.

Check that: (1) the first table column identifier is 'id'. (2) Make sure that each row has the same number of columns as defined in the columns vector. (3) Make sure that there are no repeating column identifiers. (4) Make sure that there are no repeating row identifiers.

Definition at line 297 of file datatypes_table.scad.

9.41.2.13 module table_dump (r, c, rl, cl, number = `true`)

Dump a table to the console.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
rl	<string-list> A list of selected row identifiers.
cl	<string-list> A list of selected column identifiers.
number	<boolean> Number the table rows.

Output each table row to the console. To output only select rows and columns, assign the desired identifiers to `rl` and `cl`. For example to output only the column identifiers 'c1' and 'c2', assign `cl = ["c1", "c2"]`.

Definition at line 377 of file datatypes_table.scad.

9.41.2.14 function table_exists (r, c, ri, ci)

Test the existence of a table row and column identifier.

Parameters

r	<matrix-CxR> The table data matrix (C-columns x R-rows).
c	<matrix-2xC> The table column matrix (2 x C-columns).
ri	<string> The row identifier.
ci	<string> The column identifier.

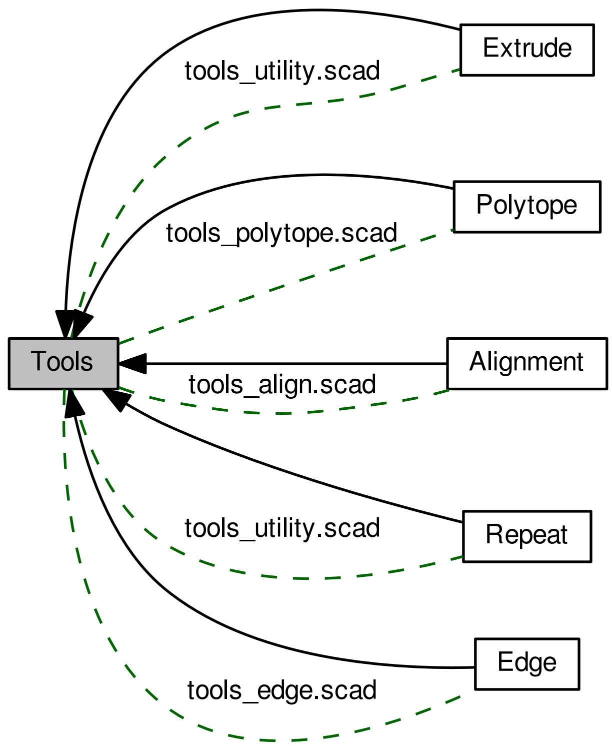
Returns

true if the row and column identifier exists, and **false** otherwise.

9.42 Tools

Design tools and techniques.

Collaboration diagram for Tools:



Modules

- Alignment

  Shape alignment tools.
- Edge

  Shape edge finishing tools.
- Extrude

  Shape extrusion tools.
- Polytope

  Polygon and polyhedron tools.
- Repeat

  Shape repetition and distribution tools.

Files

- file tools_align.scad

  Shape alignment tools.
- file tools_edge.scad

  Shape edge finishing tools.
- file tools_polytope.scad

  Polygon and polyhedron tools.
- file tools_utility.scad

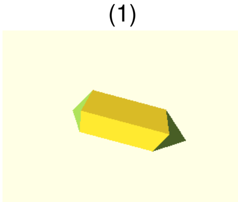
  Shape transformation utility tools.

9.42.1 Detailed Description

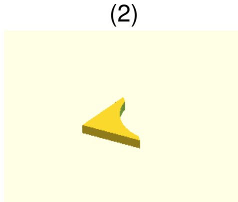
Design tools and techniques.

Table 15: Edge

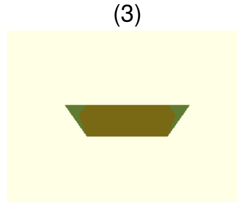
(1)



(2)



(3)



(4)

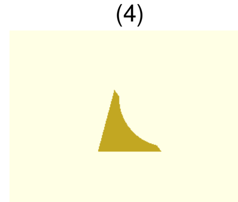
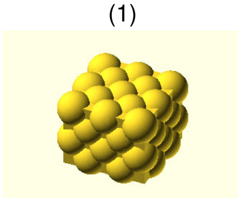
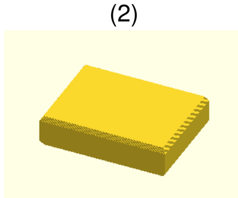


Table 16: Transformation Utilities

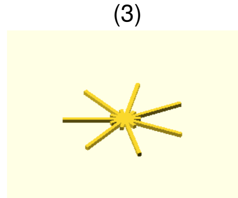
(1)



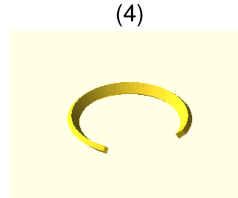
(2)



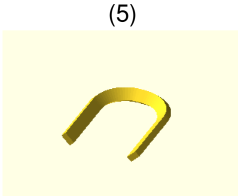
(3)



(4)



(5)



9.43 Triangles

Triangle mathematical functions.

Collaboration diagram for Triangles:

Files

- file math_triangle.scad

  Triangle solutions mathematical functions.

Functions

- function triangle_sss2lp (s1, s2, s3, cw=true)

  Compute the vertex coordinates of a triangle given its side lengths.
- function triangle_ls2lp (v, cw=true)

  Compute the vertex coordinates of a triangle given its side lengths.
- function triangle_ppp2ls (v1, v2, v3)

  Compute the side lengths of a triangle given its vertex coordinates.
- function triangle_lp2ls (v)

  Compute the side lengths of a triangle given its vertex coordinates.
- function triangle_area_ppp (v1, v2, v3, s=false)

  Compute the signed area of a triangle given its vertex coordinates.
- function triangle_area_lp (v, s=false)

  Compute the signed area of a triangle given its vertex coordinates.
- function triangle_centroid_ppp (v1, v2, v3)

  Compute the centroid (geometric center) of a triangle.
- function triangle_centroid_lp (v)

  Compute the centroid (geometric center) of a triangle.
- function triangle_incenter_ppp (v1, v2, v3)

  Compute the coordinate for the triangle's incircle.
- function triangle_incenter_lp (v)

  Compute the coordinate for the triangle's incircle.
- function triangle_inradius_ppp (v1, v2, v3)

  Compute the inradius of a triangle's incircle.
- function triangle_inradius_lp (v)

  Compute the inradius of a triangle's incircle.
- function triangle_is_cw_ppp (v1, v2, v3)

Test the vertex ordering, or orientation, of a triangle.

- function triangle_is_cw_lp (v)

  Test the vertex ordering, or orientation, of a triangle.
- function triangle_is_pit_ppp (v1, v2, v3, t)

  Test if a point is inside a triangle in a Euclidean 2d-space using Barycentric.
- function triangle_is_pit_lp (v, t)

  Test if a point is inside a triangle in a Euclidean 2d-space using Barycentric.

9.43.1 Detailed Description

Triangle mathematical functions.

See `Wikipedia` for more information.

9.43.2 Function Documentation

9.43.2.1 function triangle_area_lp (v , s = `false`)

Compute the signed area of a triangle given its vertex coordinates.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].
s	<boolean> Return the vertex ordering sign.

Returns

<decimal> The area of the given triangle.

9.43.2.2 function triangle_area_ppp (v1 , v2 , v3 , s = `false`)

Compute the signed area of a triangle given its vertex coordinates.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.
s	<boolean> Return the vertex ordering sign.

Returns

<decimal> The area of the given triangle.

9.43.2.3 function triangle_centroid_lp (v)

Compute the centroid (geometric center) of a triangle.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].

Returns

<point-2d> The centroid coordinate point [x, y].

9.43.2.4 function triangle_centroid_ppp (v1 , v2 , v3)

Compute the centroid (geometric center) of a triangle.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.

Returns

<point-2d> The centroid coordinate point [x, y].

9.43.2.5 function triangle_incenter_lp (v)

Compute the coordinate for the triangle's incircle.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].

Returns

<point-2d> The incircle coordinate point [x, y].

The interior point for which distances to the sides of the triangle are equal.

9.43.2.6 function triangle_incenter_ppp (v1 , v2 , v3)

Compute the coordinate for the triangle's incircle.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.

Returns

<point-2d> The incircle coordinate point [x, y].

The interior point for which distances to the sides of the triangle are equal.

9.43.2.7 function triangle_inradius_lp (v)

Compute the inradius of a triangle's incircle.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].

Returns

<decimal> The incircle radius.

9.43.2.8 function triangle_inradius_ppp (v1 , v2 , v3)

Compute the inradius of a triangle's incircle.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.

Returns

<decimal> The incircle radius.

9.43.2.9 function triangle_is_cw_lp (v)

Test the vertex ordering, or orientation, of a triangle.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].

Returns

<boolean> **true** if the vertices are ordered clockwise, **false** if the vertices are ordered counterclockwise, and **undef** if the ordering can not be determined.

9.43.2.10 function triangle_is_cw_ppp (v1 , v2 , v3)

Test the vertex ordering, or orientation, of a triangle.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.

Returns

<boolean> **true** if the vertices are ordered clockwise, **false** if the vertices are ordered counterclockwise, and **undef** if the ordering can not be determined.

9.43.2.11 function triangle_is_pit_lp (v , t)

Test if a point is inside a triangle in a Euclidean 2d-space using Barycentric.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].
t	<point-2d> A test point coordinate [x, y].

Returns

<boolean> **true** when the point is inside the polygon and **false** otherwise.

9.43.2.12 function triangle_is_pit_ppp (v1 , v2 , v3 , t)

Test if a point is inside a triangle in a Euclidean 2d-space using Barycentric.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.
t	<point-2d> A test point coordinate [x, y].

Returns

<boolean> **true** when the point is inside the polygon and **false** otherwise.

See `Wikipedia` for more information.

9.43.2.13 function triangle_lp2ls (v)

Compute the side lengths of a triangle given its vertex coordinates.

Parameters

v	<coords-2d> A list of vertex coordinates [v1, v2, v3].

Returns

<decimal-list-3> A list of side lengths [s1, s2, s3].

Note

Side lengths ordered according to vertex ordering.

9.43.2.14 function triangle_ls2lp (v , cw = `true`)

Compute the vertex coordinates of a triangle given its side lengths.

Parameters

v	<decimal-list-3> The list of side lengths [s1, s2, s3].
cw	<boolean> Order vertices clockwise.

Returns

<coords-2d> A list of vertex coordinates [v1, v2, v3].

Geometry requires that `s1` + `s2` is greater then `s3`. A coordinates will be **'nan'** when specified triangle does not exists.

Note

Vertex `v1` at the origin. Side length `s1` is measured along the positive x-axis.

9.43.2.15 function triangle_ppp2ls (v1 , v2 , v3)

Compute the side lengths of a triangle given its vertex coordinates.

Parameters

v1	<point-2d> A vertex coordinate [x, y] for vertex 1.
v2	<point-2d> A vertex coordinate [x, y] for vertex 2.
v3	<point-2d> A vertex coordinate [x, y] for vertex 3.

Returns

<decimal-list-3> A list of side lengths [s1, s2, s3].

Note

Side lengths ordered according to vertex ordering.

9.43.2.16 function triangle_sss2lp (s1 , s2 , s3 , cw = true)

Compute the vertex coordinates of a triangle given its side lengths.

Parameters

s1	<decimal> The length of the side 1.
s2	<decimal> The length of the side 2.
s3	<decimal> The length of the side 3.
cw	<boolean> Order vertices clockwise.

Returns

<coords-2d> A list of vertex coordinates [v1, v2, v3].

Geometry requires that `s1` + `s2` is greater then `s3`. A coordinates will be **'nan'** when specified triangle does not exists.

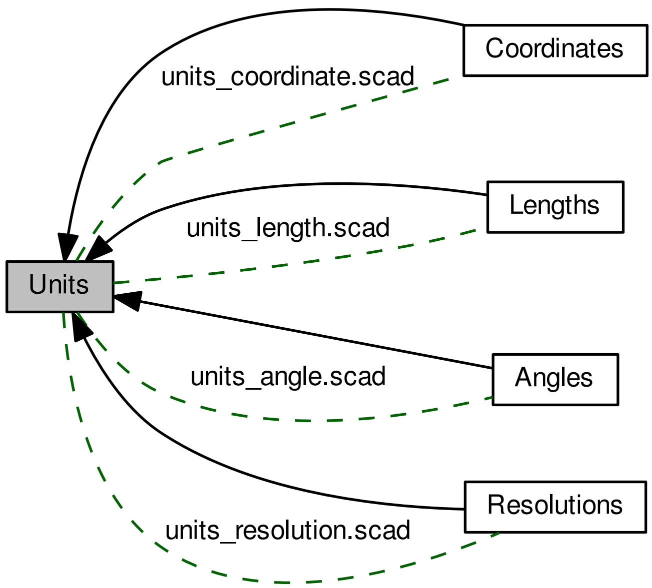
Note

Vertex `v1` at the origin. Side length `s1` is measured along the positive x-axis.

9.44 Units

Units and unit conversions.

Collaboration diagram for Units:



Modules

- Angles
  Angle units and conversions.
- Coordinates
  Coordinate systems and conversions.
- Lengths
  Length units and conversions.
- Resolutions
  Arch rendering resolution management.

Files

- file units_angle.scad
  Angle units and conversions.
- file units_coordinate.scad
  Coordinate systems and conversions.
- file units_length.scad
  Length units and conversions.

- file units_resolution.scad

  An abstraction for arc rendering resolution control.

9.44.1 Detailed Description

Units and unit conversions.

9.45 Utilities

Miscellaneous mathematical utilities.

Collaboration diagram for Utilities:

Files

- file math_utility.scad

  Miscellaneous mathematical utilities.

Functions

- function hist (v, m=0, cs, cb, cp, ca, cf, d=false)

  Generate a histogram for the elements of a list of values.

9.45.1 Detailed Description

Miscellaneous mathematical utilities.

9.45.2 Function Documentation

9.45.2.1 function hist (v , m = 0, cs , cb , cp , ca , cf , d = false)

Generate a histogram for the elements of a list of values.

Parameters

v	<data> A list of values.
m	<integer> The output mode (a 5-bit encoded integer).
cs	<string-list-4> A list of strings [s1, s2, s3, fs] (for custom field formatting).

Returns

<list|string> with the occurrence frequency of the elements of `v`.

The custom formatting strings are inserted in the output stream as follows:

```
s1, value, s2, value-frequency, s3, fs
```

See lstr_html() for description of the html formatting parameters `cb`, `cp`, `ca`, `cf`, and `d`.

Output mode selection:

bit	Description	0	1
0	output mode	numerical	string
1-3	string mode format	see table	see table
4	field separator mode	not at end	all

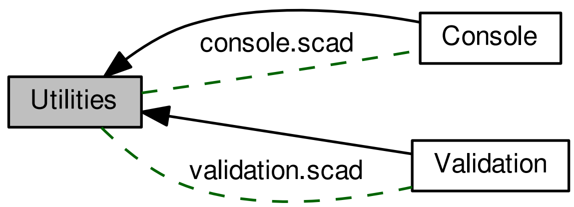
String output modes:

	B3	B2	B1	B0	Description
1	0	0	0	1	list of strings
3	0	0	1	1	text format 1
9	1	0	0	1	html format 1
15	1	1	1	1	custom formating

9.46 Utilities

General utilities.

Collaboration diagram for Utilities:



Modules

- Console

  Console message logging.
- Validation

  Function validation methods.

Files

- file console.scad

  Message logging functions.
- file validation.scad

  Function validation methods.

9.46.1 Detailed Description

General utilities.

9.47 Validation

Function validation methods.

Collaboration diagram for Validation:

Files

- file validation.scad

    Function validation methods.

Functions

- function validate (d, cv, t, ev, p=4, pf=false)

    Compare a computed test value with an known good result.

9.47.1 Detailed Description

Function validation methods.

Example

```
use <validation.scad>;
use <console.scad>;

//
// function to validate
//
function f1( x ) = (x == undef) ? 1 : 2;

farg = undef;      // function test argument
erv1 = 1;          // correct expected function result
erv2 = 3;          // incorrect expected function result

//
// pass test example
//
pass_result = validate("test-a f1(farg)", f1(farg), "equals", erv1);

if ( !validate(cv=f1(farg), t="equals", ev=erv1, pf=true) )
  log_warn( pass_result );
else
  log_info( pass_result );

//
// fail test example
//
fail_result = validate("test-b f1(farg)", f1(farg), "equals", erv2);

if ( !validate(cv=f1(farg), t="equals", ev=erv2, pf=true) )
  log_warn( fail_result );
```

```
else
  log_info( fail_result );

//
// almost equal test example
//
tvae1 = [[90.001], [[45.009], true]];
tvae2 = [[90.002], [[45.010], true]];

log_info( validate("test-c", tvae1, "almost", tvae2, 3) );
log_warn( validate("test-d", tvae1, "almost", tvae2, 4) );
```

Result

```
ECHO: "[ INFO ] root(); passed: 'test-a f1(farg)'"
ECHO:
ECHO: "root()"
ECHO: "##########################################################################"
ECHO: "# [ WARNING ] FAILED: 'test-b f1(farg)'.  Got '1'. Expected to equal '3' #"
ECHO: "##########################################################################"
ECHO: "[ INFO ] root(); passed: 'test-c'"
ECHO:
ECHO: "root()"
ECHO: "#########################################################################################################################################"
ECHO: "# [ WARNING ] FAILED: 'test-d'.  Got '[[90.001], [[45.009], true]]'. Expected to almost equal '[[90.002], [[45.01], true]]' to 4 digits #"
ECHO: "#########################################################################################################################################"
```

9.47.2 Function Documentation

9.47.2.1 function validate (d , cv , t , ev , p = 4, pf = false)

Compare a computed test value with an known good result.

Parameters

d	<string> A description.
cv	<value> A computed value to validate.
t	<string|boolean> The validation type.
ev	<value> The expected good value.
p	<number> A numerical precision for approximate comparisons.
pf	<boolean> Report result as pass or fail boolean value.

Returns

<string|boolean> Validation result indicating if the test passed or failed.

validation types	pass if (else fail)
"almost"	`cv` almost equals `ev`
"equals"	`cv` equals `ev`
"not"	`cv` not equal to `ev`
"true" | **true**	`cv` is **true**
"false" | **false**	`cv` is **false**

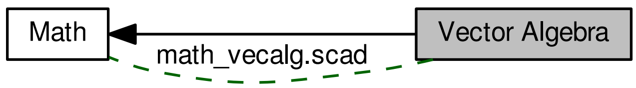
Note

When performing an **"almost"** equal validation, the comparison precision is controlled by `p`. This specifies the number of digits of precision for each numerical comparison. A passing result indicates that `cv` equals `ev` to the number of decimal digits specified by `p`. The comparison is performed by the function almost_equal().

9.48 Vector Algebra

Algebraic operations on Euclidean vectors.

Collaboration diagram for Vector Algebra:



Files

- file math_vecalg.scad

    Vector algebra mathematical functions.

Functions

- function distance_pp (p1, p2)

    Compute the distance between two Euclidean points.
- function is_left_ppp (p1, p2, p3)

    Test if a point is left, on, or right of an infinite line in a Euclidean 2d-space.
- function dimension_2to3_v (v)

    Return 3d vector unchanged or add a zeroed third dimension to 2d vector.
- function get_line_dim (l)

    Return the number of dimensions of a Euclidean line (or vector).
- function get_line_tp (l)

    Return the terminal point of a Euclidean line (or vector).
- function get_line_ip (l)

    Return the initial point of a Euclidean line (or vector).
- function get_line2origin (l)

    Shift a Euclidean line (or vector) to the origin.
- function dot_ll (l1, l2)

    Compute the dot product of two lines (or vectors).
- function cross_ll (l1, l2)

    Compute the cross product of two lines (or vectors) in a Euclidean 3d or 2d-space.
- function striple_lll (l1, l2, l3)

    Compute the scalar triple product of three lines (or vectors) in a Euclidean 3d or 2d-space.
- function angle_ll (l1, l2)

    Compute the angle between two lines (or vectors) in a Euclidean 3d or 2d-space.
- function angle_lll (l1, l2, n)

    Compute the angle between two lines (or vectors) in a Euclidean 3d-space.
- function unit_l (l)

Compute the normalized unit vector of a Euclidean line (or vector).

- function are_coplanar_lll (l1, l2, l3, d=6)

  Test if three lines (or vectors) are coplanar in Euclidean 3d-space.

- function get_pnorm2nv (pn, cw=true)

  Convert a planes' normal specification into a normal vector.

9.48.1 Detailed Description

Algebraic operations on Euclidean vectors.

See validation results.

9.48.2 Function Documentation

9.48.2.1 function angle_ll (l1 , l2)

Compute the angle between two lines (or vectors) in a Euclidean 3d or 2d-space.

Parameters

l1	<line-3d|line-2d> A 3d or 2d line (or vector) 1.
l2	<line-3d|line-2d> A 3d or 2d line (or vector) 2.

Returns

<decimal> The angle between the two lines (or vectors) in degrees. Returns **undef** when lines (or vectors) have different dimensions or when they do not intersect.

See Lines and vectors for argument specification and conventions.

Note

For 3d lines (or vectors), a normal is required to uniquely identify the perpendicular plane and axis of rotation. This function calculates the positive angle, and the plane and axis of rotation will be that which fits this assumed positive angle.

See also

angle_lll().

9.48.2.2 function angle_lll (l1 , l2 , n)

Compute the angle between two lines (or vectors) in a Euclidean 3d-space.

Parameters

l1	<line-3d> A 3d line (or vector) 1.
l2	<line-3d> A 3d line (or vector) 2.
n	<line-3d> A 3d normal line (or vector).

Returns

<decimal> The angle between the two lines (or vectors) in degrees. Returns **undef** when lines (or vectors) have different dimensions or when they do not intersect.

See Lines and vectors for argument specification and conventions.

See also

angle_ll().

9.48.2.3 function are_coplanar_lll (l1 , l2 , l3 , d = 6)

Test if three lines (or vectors) are coplanar in Euclidean 3d-space.

Parameters

l1	<line-3d> A 3d line (or vector) 1.
l2	<line-3d> A 3d line (or vector) 2.
l3	<line-3d> A 3d line (or vector) 3.
d	<integer> The number of decimal places to consider.

Returns

<boolean> **true** when all three lines (or vectors) are coplanar, and **false** otherwise.

See Lines and vectors for argument specification and conventions. See `Wikipedia` for more information.

Note

When lines (or vectors) are specified with start and end points, this function tests if they are in a planes parallel to the coplanar.

9.48.2.4 function cross_ll (l1 , l2)

Compute the cross product of two lines (or vectors) in a Euclidean 3d or 2d-space.

Parameters

l1	<line-3d|line-2d> A 3d or 2d line (or vector) 1.
l2	<line-3d|line-2d> A 3d or 2d line (or vector) 2.

Returns

<decimal\|vector-2d> The cross product of `l1` with `l2`. Returns **undef** when lines (or vectors) have different dimensions.

This function supports the abstraction outlined in Lines and vectors. The built-in operation will be more efficient in situations that do not make use of the aforementioned abstraction.

See Lines and vectors for argument specification and conventions. See Wikipedia `cross` and `determinant` for more information.

Note

This function returns the 2x2 determinant for 2d vectors.

9.48.2.5 function dimension_2to3_v (v)

Return 3d vector unchanged or add a zeroed third dimension to 2d vector.

Parameters

v	<vector-3d|vector-2d> A vector.

Returns

<vector-3d> The 3d vector or the 2d vector converted to 3d with its third dimension assigned zero.

Warning

To reduce overhead, this function assumes any vector that is not 3d to be 2d.

9.48.2.6 function distance_pp (p1 , p2)

Compute the distance between two Euclidean points.

Parameters

p1	<point> A point coordinate 1.
p2	<point> A point coordinate 2.

Returns

<decimal> The distance between the two points. Returns **undef** when points do not have equal dimensions.

When `p2` is not given, it is assumed to be at the origin. This function is similar to `norm`.

9.48.2.7 function dot_ll (l1 , l2)

Compute the dot product of two lines (or vectors).

Parameters

l1	<line> A n-dimensional line (or vector) 1.
l2	<line> A n-dimensional line (or vector) 2.

Returns

<decimal> The dot product of `l1` with `l2`. Returns **undef** when lines (or vectors) have different dimensions.

This function supports the abstraction outlined in Lines and vectors. The built-in operation will be more efficient in situations that do not make use of the aforementioned abstraction.

See Lines and vectors for argument specification and conventions. See `Wikipedia` for more information.

9.48.2.8 function get_line2origin (l)

Shift a Euclidean line (or vector) to the origin.

Parameters

l	<line> A line (or vector).

Returns

<vector> The line (or vector) shifted to the origin.

See Lines and vectors for argument specification and conventions.

9.48.2.9 function get_line_dim (l)

Return the number of dimensions of a Euclidean line (or vector).

Parameters

l	<line> A line (or vector).

Returns

<integer> The number of dimensions for the line (or vector).

See Lines and vectors for argument specification and conventions.

9.48.2.10 function get_line_ip (l)

Return the initial point of a Euclidean line (or vector).

Parameters

l	<line> A line (or vector).

Returns

<point> The initial point of the line (or vector).

See Lines and vectors for argument specification and conventions.

9.48.2.11 function get_line_tp (l)

Return the terminal point of a Euclidean line (or vector).

Parameters

l	<line> A line (or vector).

Returns

<point> The terminal point of the line (or vector).

See Lines and vectors for argument specification and conventions.

9.48.2.12 function get_pnorm2nv (pn, cw = `true`)

Convert a planes' normal specification into a normal vector.

Parameters

pn	<pnorm> A plane normal specification.
cw	<boolean> Point ordering. When the plane specified as non-collinear points, this indicates ordering.

Returns

<normal> A vector-3d normal to the plane.

See Planes' normal for argument specification and conventions.

9.48.2.13 function is_left_ppp (p1, p2, p3)

Test if a point is left, on, or right of an infinite line in a Euclidean 2d-space.

Parameters

p1	<point-2d> A 2d point coordinate 1.
p2	<point-2d> A 2d point coordinate 2.
p3	<point-2d> A 2d point coordinate 3.

Returns

<decimal> (> 0) for `p3` *left* of the line through `p1` and `p2`, (**=** 0) for p3 *on* the line, and (< 0) for p3 right of the line.

Function patterned after `Dan Sunday, 2012`.

9.48.2.14 function striple_lll (l1 , l2 , l3)

Compute the scalar triple product of three lines (or vectors) in a Euclidean 3d or 2d-space.

Parameters

l1	<line-3d|line-2d> A 3d or 2d line (or vector) 1.
l2	<line-3d|line-2d> A 3d or 2d line (or vector) 2.
l3	<line-3d|line-2d> A 3d or 2d line (or vector) 3.

Returns

<decimal|vector-2d> The scalar triple product. Returns **undef** when lines (or vectors) have different dimensions.

[l1, l2, l3] = l1 $*$ (l2 x l3)

See Lines and vectors for argument specification and conventions. See `Wikipedia` for more information.

Warning

Returns a 2d vector result for 2d vectors. The cross product computes the 2x2 determinant of the vectors `(l2 x l3)`, a scalar value, which is then *multiplied* by `l1`.

9.48.2.15 function unit_l (l)

Compute the normalized unit vector of a Euclidean line (or vector).

Parameters

l	<line> A line (or vector).

Returns

<vector> The normalized unit vector.

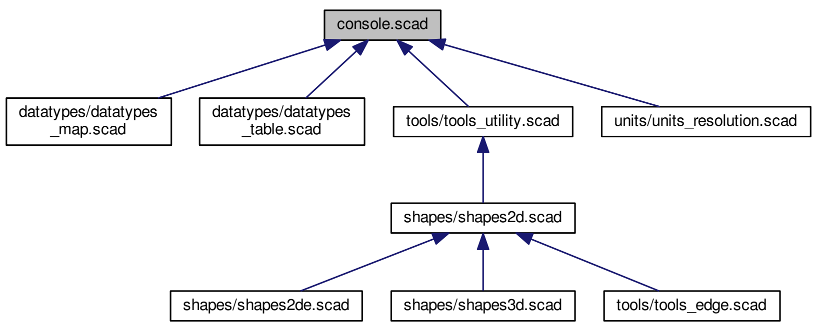
See Lines and vectors for argument specification and conventions.

10 File Documentation

10.1 console.scad File Reference

Message logging functions.

This graph shows which files directly or indirectly include this file:



Functions

- function stack (b=0, t=0)

  Format the function call stack as a string.
- module log_echo (m)

  Output message to console.
- module log_debug (m)

  Output diagnostic message to console.
- module log_info (m)

  Output information message to console.
- module log_warn (m)

  Output warning message to console.
- module log_error (m)

  Output error message to console.

10.1.1 Detailed Description

Message logging functions.

Author

Roy Allen Sutton

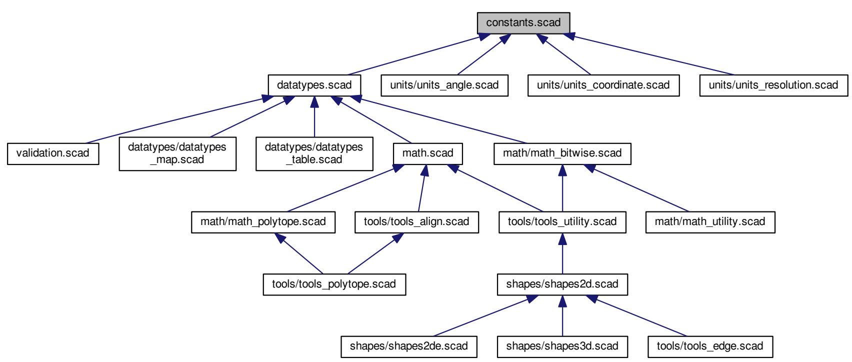
Date

2015-2017

10.2 constants.scad File Reference

Design constant definitions.

This graph shows which files directly or indirectly include this file:



Variables

- aeps = 0.001

  <decimal> Epsilon, small distance to deal with overlapping shapes.
- pi = 3.14159265358979323

  <decimal> The ratio of a circle's circumference to its diameter.
- tau = 2∗pi

  <decimal> The ratio of a circle's circumference to its radius.
- phi = (1 + sqrt(5)) / 2

  <decimal> The golden ratio.
- number_max = 1e308

  <decimal> The largest representable number in OpenSCAD scripts.
- number_min = -1e308

  <decimal> The smallest representable number in OpenSCAD scripts.

- empty_str = ""

  <string> A string with no characters (the empty string).
- empty_lst = [ ]

  <list> A list with no values (the empty list).
- x_axis_ci = 0

  <integer> The coordinate axis index for the Euclidean space x-axis.
- y_axis_ci = 1

  <integer> The coordinate axis index for the Euclidean space y-axis.
- z_axis_ci = 2

  <integer> The coordinate axis index for the Euclidean space z-axis.
- zero2d = [0, 0]

  <decimal-list-2> A 2d zero vector (a list with two zeros).
- origin2d = [0, 0]

  <point-2d> The origin point coordinate in 2d Euclidean space.
- x_axis2d_uv = [1, 0]

  <vector-2d> The unit vector of the positive x-axis in 2d Euclidean space.
- y_axis2d_uv = [0, 1]

  <vector-2d> The unit vector of the positive y-axis in 2d Euclidean space.
- x_axis2d_ul = [-x_axis2d_uv, +x_axis2d_uv]

  <line-2d> A positively-directed unit line centered on the x-axis in 2d Euclidean space.
- y_axis2d_ul = [-y_axis2d_uv, +y_axis2d_uv]

  <line-2d> A positively-directed unit line centered on the y-axis in 2d Euclidean space.
- zero3d = [0, 0, 0]

  <decimal-list-2> A 3d zero vector (a list with three zeros).
- origin3d = [0, 0, 0]

  <point-3d> The origin point coordinate in 3-dimensional Euclidean space.
- x_axis3d_uv = [1, 0, 0]

  <vector-3d> The unit vector of the positive x-axis in 3d Euclidean space.
- y_axis3d_uv = [0, 1, 0]

  <vector-3d> The unit vector of the positive y-axis in 3d Euclidean space.
- z_axis3d_uv = [0, 0, 1]

  <vector-3d> The unit vector of the positive z-axis in 3d Euclidean space.
- x_axis3d_ul = [-x_axis3d_uv, +x_axis3d_uv]

  <line-3d> A positively-directed unit line centered on the x-axis in 3d Euclidean space.
- y_axis3d_ul = [-y_axis3d_uv, +y_axis3d_uv]

  <line-3d> A positively-directed unit line centered on the y-axis in 3d Euclidean space.
- z_axis3d_ul = [-z_axis3d_uv, +z_axis3d_uv]

  <line-3d> A positively-directed unit line centered on the z-axis in 3d Euclidean space.
- xy_plane_on = [origin3d, z_axis3d_uv]

  <plane> The right-handed xy plane centered at the origin with normal vector.
- yz_plane_on = [origin3d, x_axis3d_uv]

  <plane> The right-handed yz plane centered at the origin with normal vector.
- zx_plane_on = [origin3d, y_axis3d_uv]

  <plane> The right-handed zx plane centered at the origin with normal vector.
- xy_plane_os = [origin3d, [for (r=[[1,1],[1,-1],[-1,-1],[-1,1]]) [r[0],r[1],0]]]

  <plane> The right-handed xy plane centered at the origin with coplanar unit square points.
- yz_plane_os = [origin3d, [for (r=[[1,1],[1,-1],[-1,-1],[-1,1]]) [0,r[0],r[1]]]]

  <plane> The right-handed yz plane centered at the origin with coplanar unit square points.
- zx_plane_os = [origin3d, [for (r=[[1,1],[1,-1],[-1,-1],[-1,1]]) [r[1],0,r[0]]]]

  <plane> The right-handed zx plane centered at the origin with coplanar unit square points.

10.2.1 Detailed Description

Design constant definitions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

Include this library file using the **include** statement.

10.3 database/geometry/polyhedra/anti_prisms.scad File Reference

Table of polyhedra data group: `anti_prisms`.

Variables

- dtc_polyhedra_anti_prisms
- dtr_polyhedra_anti_prisms

10.3.1 Detailed Description

Table of polyhedra data group: `anti_prisms`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.4 database/geometry/polyhedra/archimedean.scad File Reference

Table of polyhedra data group: `archimedean`.

Variables

- dtc_polyhedra_archimedean
- dtr_polyhedra_archimedean

10.4.1 Detailed Description

Table of polyhedra data group: `archimedean`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.5 database/geometry/polyhedra/archimedean_duals.scad File Reference

Table of polyhedra data group: `archimedean_duals`.

Variables

- dtc_polyhedra_archimedean_duals
- dtr_polyhedra_archimedean_duals

10.5.1 Detailed Description

Table of polyhedra data group: `archimedean_duals`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.6 database/geometry/polyhedra/cupolas.scad File Reference

Table of polyhedra data group: `cupolas`.

Variables

- dtc_polyhedra_cupolas
- dtr_polyhedra_cupolas

10.6.1 Detailed Description

Table of polyhedra data group: `cupolas`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.7 database/geometry/polyhedra/dipyramids.scad File Reference

Table of polyhedra data group: `dipyramids`.

Variables

- dtc_polyhedra_dipyramids
- dtr_polyhedra_dipyramids

10.7.1 Detailed Description

Table of polyhedra data group: `dipyramids`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.8 database/geometry/polyhedra/johnson.scad File Reference

Table of polyhedra data group: `johnson`.

Variables

- dtc_polyhedra_johnson
- dtr_polyhedra_johnson

10.8.1 Detailed Description

Table of polyhedra data group: `johnson`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.9 database/geometry/polyhedra/platonic.scad File Reference

Table of polyhedra data group: `platonic`.

Variables

- dtc_polyhedra_platonic
- dtr_polyhedra_platonic

10.9.1 Detailed Description

Table of polyhedra data group: `platonic`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.10 database/geometry/polyhedra/polyhedra_all.scad File Reference

Table of polyhedra data group: `polyhedra_all`.

Variables

- dtc_polyhedra_polyhedra_all
- dtr_polyhedra_polyhedra_all

10.10.1 Detailed Description

Table of polyhedra data group: `polyhedra_all`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.11 database/geometry/polyhedra/prisms.scad File Reference

Table of polyhedra data group: `prisms`.

Variables

- dtc_polyhedra_prisms
- dtr_polyhedra_prisms

10.11.1 Detailed Description

Table of polyhedra data group: `prisms`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.12 database/geometry/polyhedra/pyramids.scad File Reference

Table of polyhedra data group: `pyramids`.

Variables

- dtc_polyhedra_pyramids
- dtr_polyhedra_pyramids

10.12.1 Detailed Description

Table of polyhedra data group: `pyramids`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

Include this library file using the **include** statement.

10.13 database/geometry/polyhedra/trapezohedron.scad File Reference

Table of polyhedra data group: `trapezohedron`.

Variables

- dtc_polyhedra_trapezohedron
- dtr_polyhedra_trapezohedron

10.13.1 Detailed Description

Table of polyhedra data group: `trapezohedron`.

Author

Roy Allen Sutton

Date

2017

Copyright

This omdl formatted data table has been assembled using a script that converts the polyhedra data obtained from Anthony Thyssen's Studies into Polyhedra. The vertices are tabulated in both their original Cartesian as well as their converted spherical coordinate form, which is convenient when scaling. The data originates from one of three sources:

- Exact Mathematics as presented by Anthony Thyssen,
- the Polyhedron Database maintained by Netlib, and
- an Encyclopedia of Polyhedra by George W. Hart.

Note

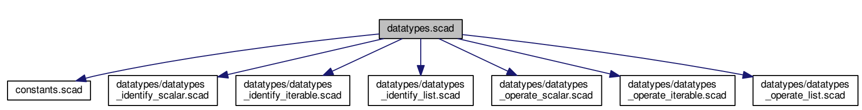
Include this library file using the **include** statement.

10.14 datatypes.scad File Reference

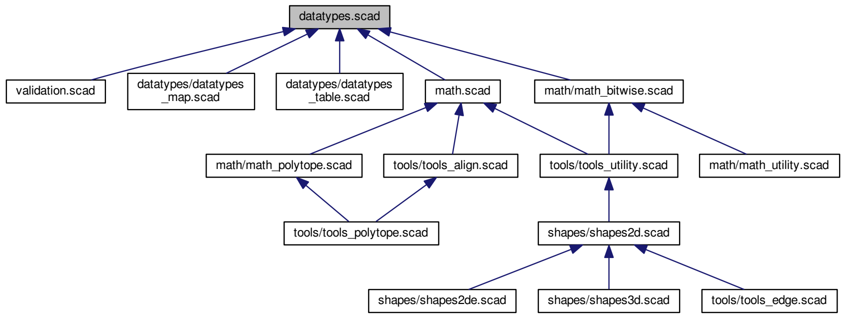
Data type identification and operations.

```
#include <constants.scad>
#include <datatypes/datatypes_identify_scalar.scad>
#include <datatypes/datatypes_identify_iterable.scad>
#include <datatypes/datatypes_identify_list.scad>
#include <datatypes/datatypes_operate_scalar.scad>
#include <datatypes/datatypes_operate_iterable.scad>
#include <datatypes/datatypes_operate_list.scad>
```

Include dependency graph for datatypes.scad:



This graph shows which files directly or indirectly include this file:



10.14.1 Detailed Description

Data type identification and operations.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

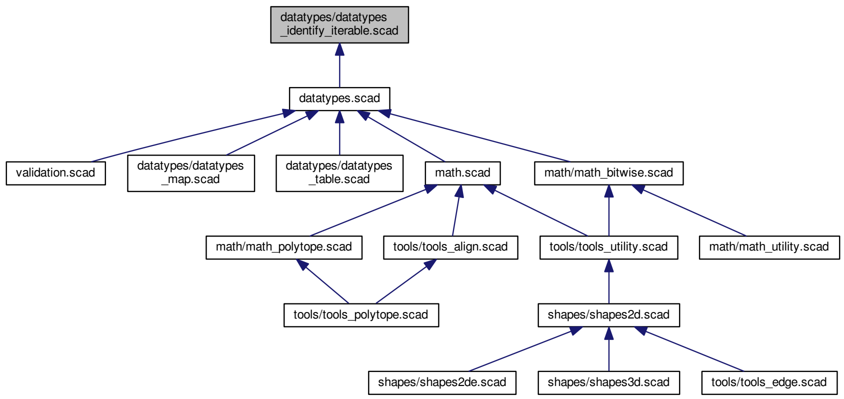
Note

Include this library file using the **include** statement.

10.15 datatypes/datatypes_identify_iterable.scad File Reference

Iterable data type identification.

This graph shows which files directly or indirectly include this file:



Functions

- function all_equal (v, cv)

  Test if a list of values equal a comparison value.
- function any_equal (v, cv)

  Test if any element of a list of values equal a comparison value.
- function all_defined (v)

  Test if no element of a list of values is undefined.
- function any_undefined (v)

  Test if any element of a list of values is undefined.
- function all_scalars (v)

  Test if all elements of a list of values are scalars.
- function all_lists (v)

  Test if all elements of a list of values are lists.
- function all_strings (v)

  Test if all elements of a list of values are strings.
- function all_numbers (v)

  Test if all elements of a list of values are numbers.
- function all_len (v, l)

  Test if all elements of a list of values are lists of a specified length.

10.15.1 Detailed Description

Iterable data type identification.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

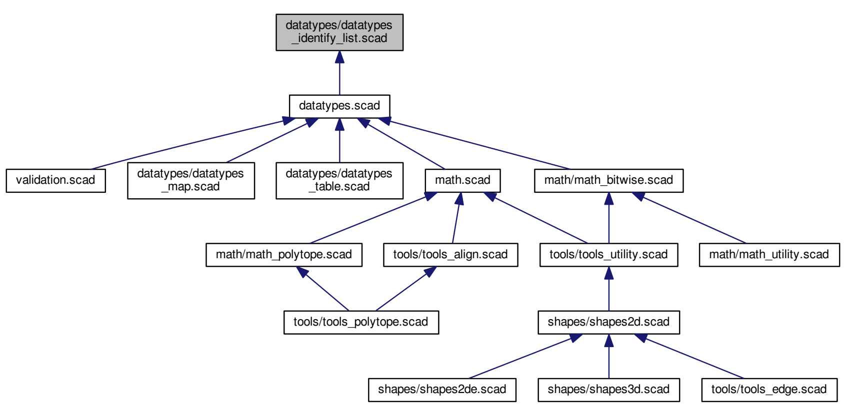
Note

Include this library file using the **include** statement.

10.16 datatypes/datatypes_identify_list.scad File Reference

List data type identification.

This graph shows which files directly or indirectly include this file:



Functions

- function n_almost_equal (v1, v2, p=6)

  Test if all elements of two lists of numbers are sufficiently equal.
- function almost_equal (v1, v2, p=6)

  Test if all numerical elements of two lists of values are sufficiently equal.

- function compare (v1, v2, s=true)

  Order to lists of arbitrary values.

10.16.1 Detailed Description

List data type identification.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

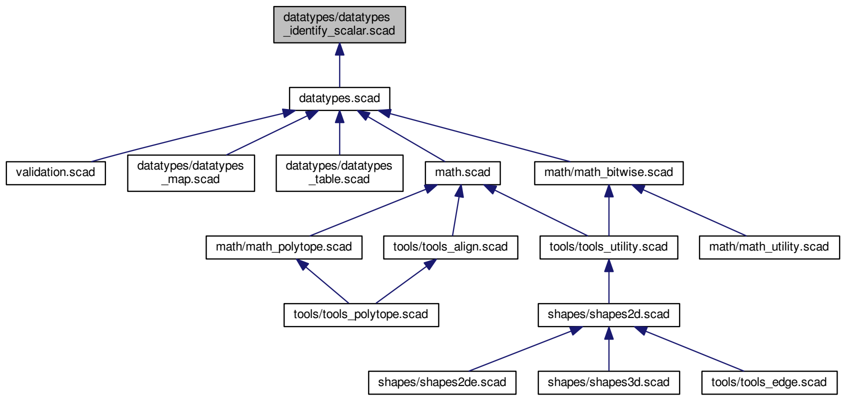
Note

Include this library file using the **include** statement.

10.17 datatypes/datatypes_identify_scalar.scad File Reference

Scalar data type identification.

This graph shows which files directly or indirectly include this file:



Functions

- function is_defined (v)

  Test if a value is defined.
- function not_defined (v)

  Test if a value is not defined.
- function is_nan (v)

  Test if a numerical value is invalid.
- function is_inf (v)

  Test if a numerical value is infinite.
- function is_scalar (v)

  Test if a value is a single non-iterable value.
- function is_iterable (v)

  Test if a value has multiple parts and is iterable.
- function is_empty (v)

  Test if an iterable value is empty.
- function is_number (v)

  Test if a value is a number.
- function is_integer (v)

  Test if a value is an integer.
- function is_decimal (v)

  Test if a value is a decimal.
- function is_boolean (v)

Test if a value is a predefined boolean constant.

- function is_string (v)

    Test if a value is a string.

- function is_list (v)

    Test if a value is an iterable list of values.

- function is_range (v)

    Test if a value is a range definition.

- function is_even (v)

    Test if a numerical value is even.

- function is_odd (v)

    Test if a numerical value is odd.

- function is_between (v, l, u)

    Test if a numerical value is between an upper and lower bounds.

10.17.1 Detailed Description

Scalar data type identification.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

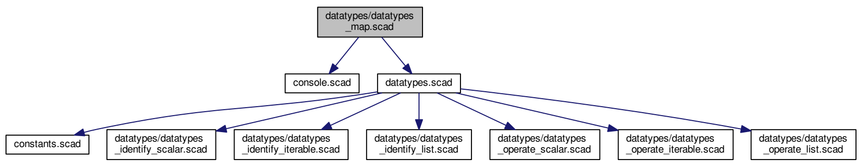
Include this library file using the **include** statement.

10.18 datatypes/datatypes_map.scad File Reference

Map data type operations.

```
#include <console.scad>
#include <datatypes.scad>
```

Include dependency graph for datatypes_map.scad:



Functions

- function get_map_i (m, k)

  Return the index of a map key.
- function map_exists (m, k)

  Test if a key exists.
- function get_map_v (m, k)

  Get the map value associated with a key.
- function get_map_kl (m)

  Get a list of all map keys.
- function get_map_vl (m)

  Get a list of all map values.
- function get_map_size (m)

  Get the number of map entries.
- module map_check (m, verbose=false)

  Perform some basic validation/checks on a map.
- module map_dump (m, sort=true, number=true, p=3)

  Dump each map entry to the console.

10.18.1 Detailed Description

Map data type operations.

Author

Roy Allen Sutton

Date

2015-2017

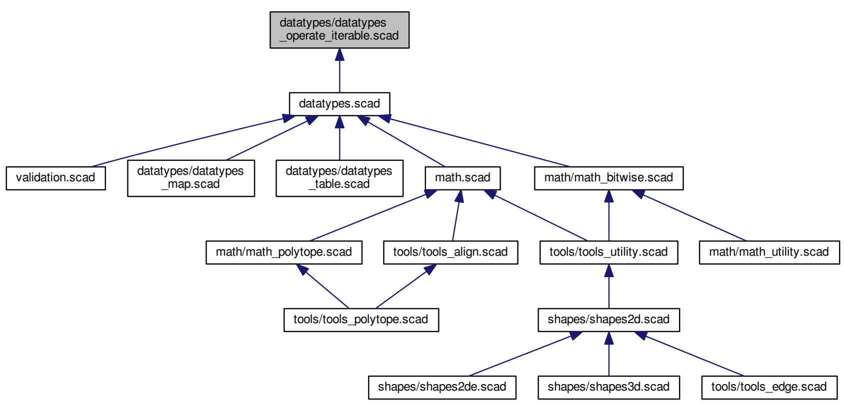
Copyright

Manage a collection of key-value pairs where keys are unique.

10.19 datatypes/datatypes_operate_iterable.scad File Reference

Iterable data type operations.

This graph shows which files directly or indirectly include this file:



Functions

- function edefined_or (v, i, d)

    Return an iterable element when it exists or a default value when it does not.
- function find (mv, v, c=1, i, i1=0, i2)

    Find the occurrences of a match value in an iterable value.
- function count (mv, v, s=true, i)

    Count all occurrences of a match value in an iterable value.
- function exists (mv, v, s=true, i)

    Check for the existence of a match value in an iterable value.
- function first (v)

Return the first element of an iterable value.
- function second (v)

    Return the second element of an iterable value.
- function third (v)

    Return the third element of an iterable value.
- function last (v)

    Return the last element of an iterable value.
- function nfirst (v, n=1)

    Return a list containing the first n elements of an iterable value.
- function nlast (v, n=1)

    Return a list containing the last n elements of an iterable value.
- function nhead (v, n=1)

    Return a list containing all but the last n elements of an iterable value.
- function ntail (v, n=1)

    Return a list containing all but the first n elements of an iterable value.
- function reverse (v)

    Reverse the elements of an iterable value.
- function rselect (v, i)

    Select a range of elements from an iterable value.
- function nssequence (v, n=1, s=1, w=false)

    Return a list of all n-element sequential-subsets of an iterable value.
- function eappend (nv, v, r=true, j=true, l=true)

    Append a value to each element of an iterable value.
- function insert (nv, v, i=0, mv, mi=0, s=true, si)

    Insert a new value into an iterable value.
- function delete (v, i, mv, mc=1, s=true, si)

    Delete elements from an iterable value.
- function strip (v, mv=empty_lst)

    Strip all matching values from an iterable value.
- function unique (v)

    Return the unique elements of an iterable value.

10.19.1 Detailed Description

Iterable data type operations.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

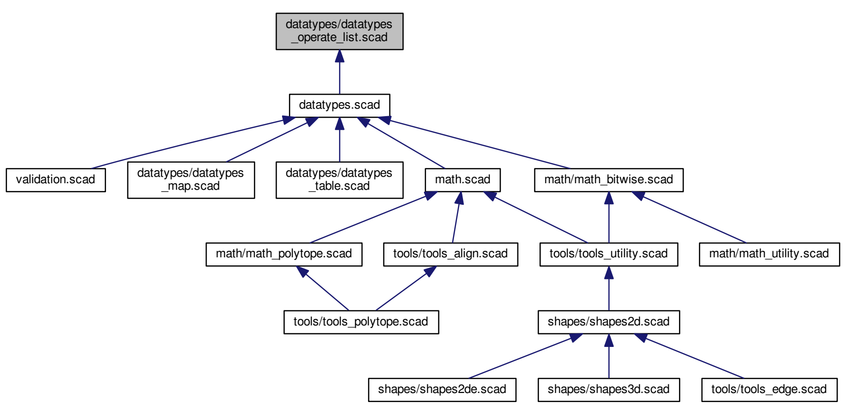
Note

Include this library file using the **include** statement.

10.20 datatypes/datatypes_operate_list.scad File Reference

List data type operations.

This graph shows which files directly or indirectly include this file:



Functions

- function lstr (v)

  Convert a list of values to a concatenated string.
- function lstr_html (v, b, p, a, f, d=false)

  Convert a list of values to a concatenated HTML-formatted string.
- function consts (l, v, u=false)

  Create a sequence of constant or incrementing elements.
- function get_index (l, s=true, rs)

Create a sequence for a list index sequence specification.

- function pad (l, w, p=0, r=true)

  Pad a list to a constant width of elements.
- function dround (v, d=6)

  Round all numerical values of a list to a fixed number of decimal point digits.
- function sround (v, d=6)

  Round all numerical values of a list to a fixed number of significant figures.
- function limit (v, l, u)

  Limit all numerical values of a list between an upper and lower bounds.
- function sum (v, i1, i2)

  Compute the sum of a list of numbers.
- function mean (v)

  Compute the mean/average of a list of numbers.
- function ciselect (v, i)

  Case-like select a value from a list of ordered value options.
- function cmvselect (v, mv)

  Case-like select a value from a list of mapped key-value options.
- function eselect (v, f=true, l=false, i)

  Select a specified element from each iterable value of a list.
- function smerge (v, r=false)

  Serial-merge lists of iterable values.
- function pmerge (v, j=true)

  Parallel-merge lists of iterable values.
- function qsort (v, i, r=false)

  Sort the numeric or string elements of a list using quick sort.
- function qsort2 (v, i, d=0, r=false, s=true)

  Hierarchically sort an arbitrary data list using quick sort.

10.20.1 Detailed Description

List data type operations.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

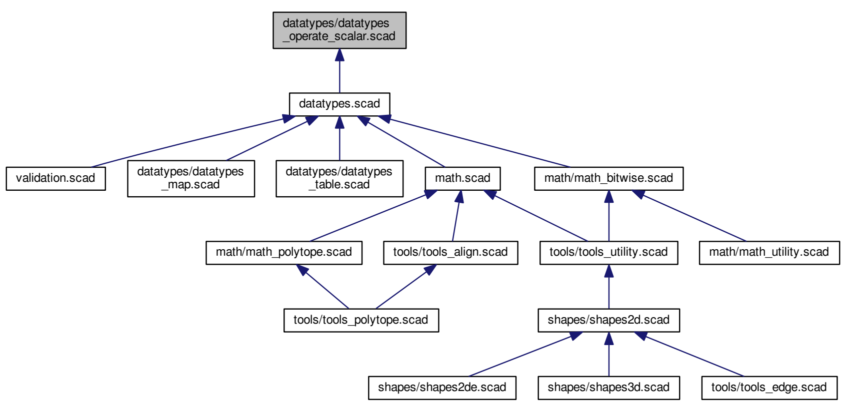
Note

Include this library file using the **include** statement.

10.21 datatypes/datatypes_operate_scalar.scad File Reference

Scalar data type operations.

This graph shows which files directly or indirectly include this file:



Functions

- function defined_or (v, d)

  Return a value when it is defined or a default value when it is not.
- function circular_index (i, l, f=0)

  Map an index position into a circularly indexed list.

10.21.1 Detailed Description

Scalar data type operations.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

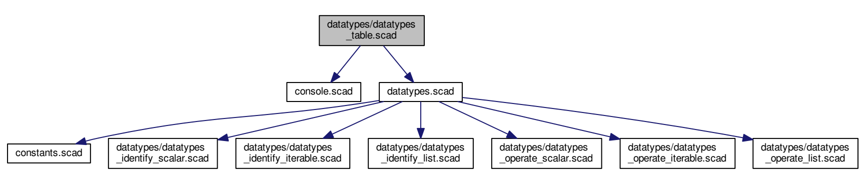
Include this library file using the **include** statement.

10.22 datatypes/datatypes_table.scad File Reference

Table data type operations.

```
#include <console.scad>
#include <datatypes.scad>
```

Include dependency graph for datatypes_table.scad:



Functions

- function get_table_ri (r, ri)

  Get the table row index that matches a table row identifier.
- function get_table_r (r, ri)

  Get the table row that matches a table row identifier.
- function get_table_ci (c, ci)

Get the table column index that matches a table column identifier.

- function get_table_c (c, ci)

    Get the table column that matches a table column identifier.
- function get_table_v (r, c, ri, ci)

    Get the table cell value for a specified row and column identifier.
- function get_table_crl (r, c, ci)

    Form a list of a select column across all table rows.
- function get_table_ridl (r)

    Form a list of all table row identifiers.
- function get_table_cidl (c)

    Form a list of all table column identifiers.
- function table_exists (r, c, ri, ci)

    Test the existence of a table row and column identifier.
- function get_table_size (r, c)

    Get the size of a table.
- function get_table_copy (r, c, rl, cl)

    Create a new matrix from select rows and columns of a table.
- function get_table_sum (r, c, rl, cl)

    Sum select rows and columns of a table.
- module table_check (r, c, verbose=false)

    Perform some basic validation/checks on a table.
- module table_dump (r, c, rl, cl, number=true)

    Dump a table to the console.

10.22.1 Detailed Description

Table data type operations.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

10.23 mainpage.scad File Reference

Documentation main page.

10.23.1 Detailed Description

Documentation main page.

Author

Roy Allen Sutton

Date

2015-2017

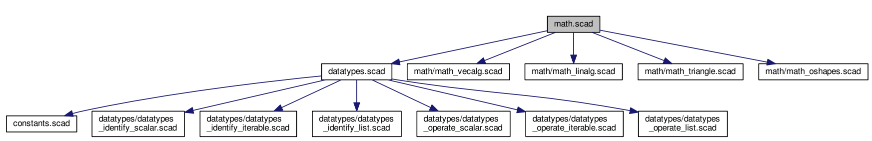
Copyright

10.24 math.scad File Reference

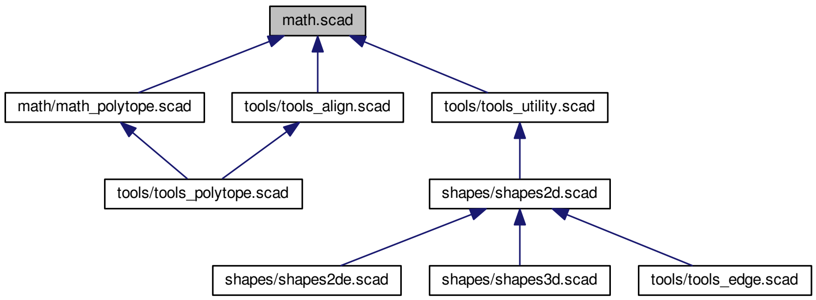
Mathematical function primitives.

```
#include <datatypes.scad>
#include <math/math_vecalg.scad>
#include <math/math_linalg.scad>
#include <math/math_triangle.scad>
#include <math/math_oshapes.scad>
```

Include dependency graph for math.scad:



This graph shows which files directly or indirectly include this file:



10.24.1 Detailed Description

Mathematical function primitives.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

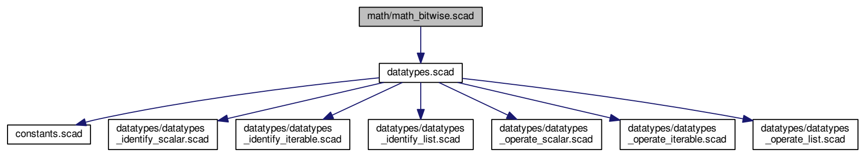
Include this library file using the **include** statement.

10.25 math/math_bitwise.scad File Reference

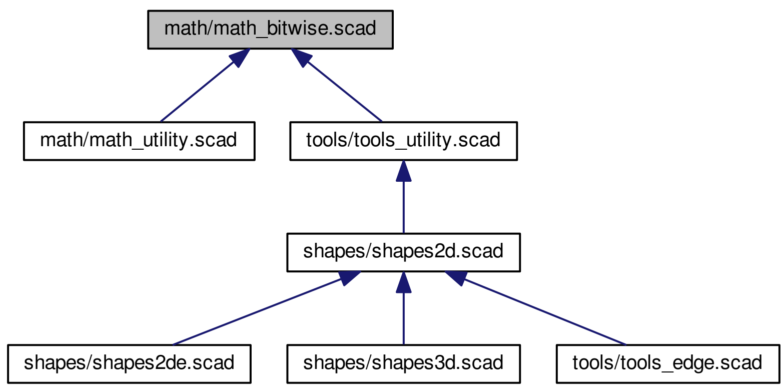
Mathematical base-two bitwise binary functions.

`#include <datatypes.scad>`

Include dependency graph for math_bitwise.scad:



This graph shows which files directly or indirectly include this file:



Functions

- function bitwise_is_equal (v, b, t=1)

    Test if a base-two bit position of an integer value equals a test bit.
- function bitwise_i2v (v, w=1, bv=1)

    Encode an integer value as a base-two list of bits.
- function bitwise_v2i (v)

    Decode a base-two list of bits to an integer value.
- function bitwise_i2s (v, w=1)

    Encode an integer value as a base-two string of bits.
- function bitwise_s2i (v)

    Decode a base-two string of bits to an integer value.
- function bitwise_imi (v, w, s)

    Decode the integer in a value at a shifted base-two bit mask of width-w.
- function bitwise_and (v1, v2, bv=1)

    Base-two bitwise AND operation for integers.
- function bitwise_or (v1, v2, bv=1)

    Base-two bitwise OR operation for integers.

- function bitwise_xor (v1, v2, bv=1)

  Base-two bitwise XOR operation for integers.
- function bitwise_not (v, w=1, bv=1)

  Base-two bitwise NOT operation for an integer.
- function bitwise_lsh (v, s=1, bm=1, bv=1)

  Base-two bitwise left-shift operation for an integer.
- function bitwise_rsh (v, s=1)

  Base-two bitwise right-shift operation for an integer.

10.25.1 Detailed Description

Mathematical base-two bitwise binary functions.

Author

Roy Allen Sutton

Date

2017

Copyright

This file is part of `omdl`, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the `GNU Lesser General Public License` as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see `http://www.gnu.org/licenses/`.

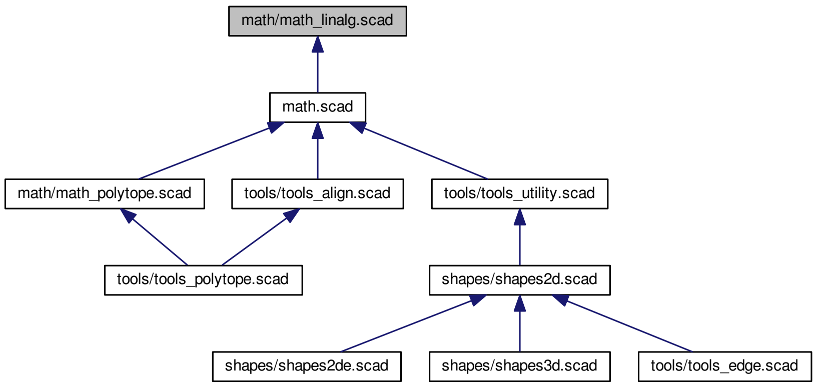
Note

Include this library file using the **include** statement.

10.26 math/math_linalg.scad File Reference

Linear algebra mathematical functions.

This graph shows which files directly or indirectly include this file:



Functions

- function multmatrix_lp (c, m)

  Multiply all coordinates by a 4x4 3d-transformation matrix.
- function translate_lp (c, v)

  Translate all coordinates dimensions.
- function rotate_lp (c, a, v, o=origin3d)

  Rotate all coordinates about one or more axes in Euclidean 2d or 3d space.
- function scale_lp (c, v)

  Scale all coordinates dimensions.
- function resize_lp (c, v)

  Scale all coordinates dimensions proportionately to fit inside a region.

10.26.1 Detailed Description

Linear algebra mathematical functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

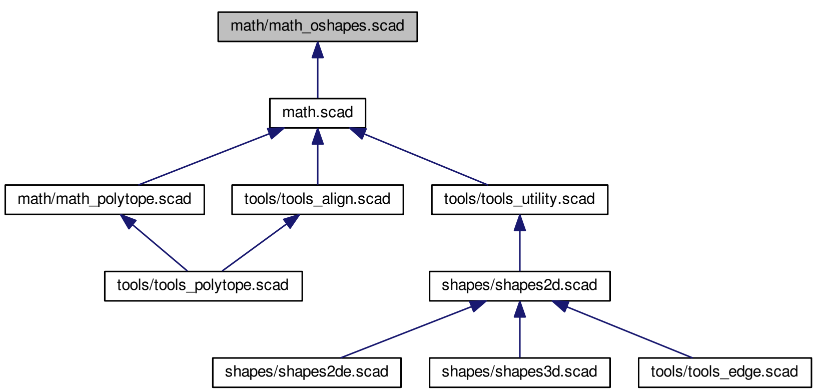
Note

Include this library file using the **include** statement.

10.27 math/math_oshapes.scad File Reference

Other shapes mathematical functions.

This graph shows which files directly or indirectly include this file:



Functions

- function rpolygon_lp (n, r, a, vr, cw=true)

    Compute the coordinates for an n-sided regular polygon.
- function rpolygon_area (n, r, a)

    Compute the area of an n-sided regular polygon.
- function rpolygon_perimeter (n, r, a)

    Compute the perimeter of an n-sided regular polygon.

10.27.1 Detailed Description

Other shapes mathematical functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

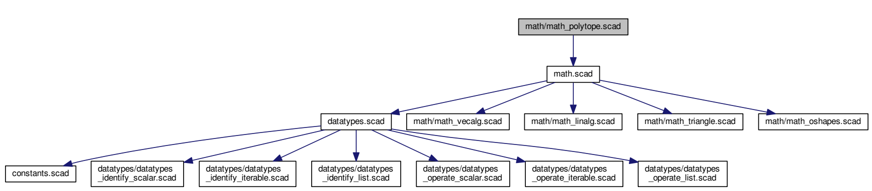
Note

Include this library file using the **include** statement.

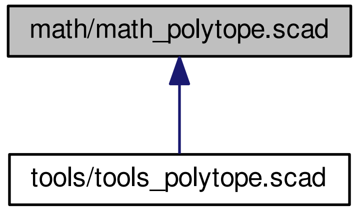
10.28 math/math_polytope.scad File Reference

Polygon and polyhedron mathematical functions.

`#include <math.scad>`
Include dependency graph for math_polytope.scad:



This graph shows which files directly or indirectly include this file:



Functions

- function polytope_faces2edges (f)

  List the edge coordinate index pairs of a polytope.
- function polytope_limits (c, f, a, d=[0:2], s=true)

  Determine the bounding limits of a polytope.
- function polytope_bbox_pf (c, f, a)

  Generate a bounding box polytope for another polytope in 3d or 2d.
- function polytope_line (c, f, e, i, l, r=false)

  Get a line from an edge or any two vetices of a polytope.
- function polytope_vertex_av (f, i)

  List the adjacent vertices for a given polytope vertex.
- function polytope_vertex_af (f, i)

  List the adjacent face indexes for a polytope vertex.
- function polytope_edge_af (f, e, i)

  List the adjacent face indexes for a polytope edge.
- function polytope_vertex_n (c, f, i)

  Get a normal vector for a polytope vertex.
- function polytope_edge_n (c, f, e, i)

  Get a normal vector for a polytope edge.
- function polytope_face_n (c, f, i, l, cw=true)

  Get the normal vector of a polytope face.
- function polytope_face_m (c, f, i, l)

  Get the mean coordinate of all vertices of a polytope face.
- function polytope_face_mn (c, f, i, l, cw=true)

  Get the mean coordinate and normal vector of a polytope face.
- function polytope_plane (c, f, i, l, cw=true)

  Get a plane for a polytope face.
- function polytope_face_vcounts (f)

  List the vertex counts for all polytope faces.
- function polytope_face_angles (c, f)

  List the angles between all adjacent faces of a polyhedron.

- function polytope_edge_lengths (c, e)

    List the edge lengths of a polytope.
- function polytope_edge_angles (c, f)

    List the adjacent edge angles for each polytope vertex.
- function polytope_faces_are_regular (c, f, e, d=6)

    Test if the faces of a polytope are all regular.
- function polytope_triangulate_ft (f)

    Triangulate the faces of a convex polytope using fan triangulation.
- function polygon2d_perimeter (c, p)

    Calculate the perimeter length of a polygon in 2d.
- function polygon2d_area (c, p, s=false)

    Compute the signed area of a polygon in a Euclidean 2d-space.
- function polygon3d_area (c, p, n)

    Compute the area of a polygon in a Euclidean 3d-space.
- function polygon2d_centroid (c, p)

    Compute the center of mass of a polygon in a Euclidean 2d-space.
- function polygon2d_is_cw (c, p)

    Test the vertex ordering of a polygon in a Euclidean 2d-space.
- function polygon2d_is_convex (c, p)

    Test the convexity of a polygon in a Euclidean 2d-space.
- function polygon2d_winding (c, p, t)

    Compute the winding number of a polygon about a point in a Euclidean 2d-space.
- function polygon2d_is_pip_wn (c, p, t)

    Test if a point is inside a polygon in a Euclidean 2d-space using winding number.
- function polygon2d_is_pip_as (c, p, t)

    Test if a point is inside a polygon in a Euclidean 2d-space using angle summation.
- function polyhedron_area (c, f)

    Compute the surface area of a polyhedron in a Euclidean 3d-space.
- function polyhedron_volume_tf (c, f)

    Compute the volume of a triangulated polyhedron in a Euclidean 3d-space.
- function polyhedron_centroid_tf (c, f)

    Compute the center of mass of a triangulated polyhedron in a Euclidean 3d-space.
- function linear_extrude_pp2pf (c, p, h=1, centroid=false, center=false)

    Convert a polygon to a polyhedron by adding a height dimension.

10.28.1 Detailed Description

Polygon and polyhedron mathematical functions.

Author

Roy Allen Sutton

Date

2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

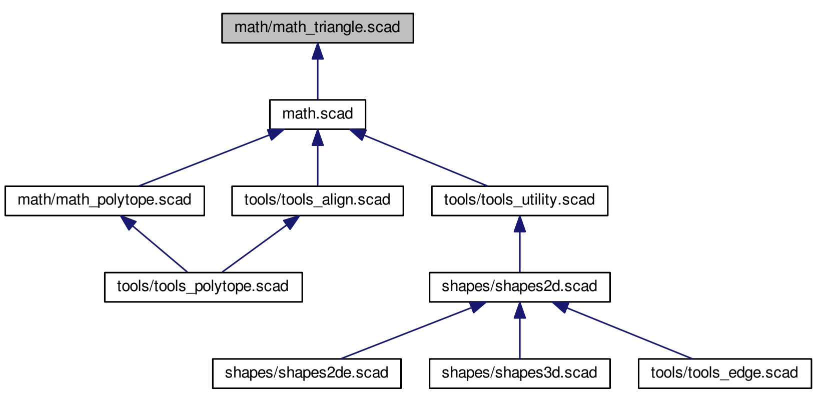
Note

Include this library file using the **include** statement.

10.29 math/math_triangle.scad File Reference

Triangle solutions mathematical functions.

This graph shows which files directly or indirectly include this file:



Functions

- function triangle_sss2lp (s1, s2, s3, cw=true)

  Compute the vertex coordinates of a triangle given its side lengths.
- function triangle_ls2lp (v, cw=true)

  Compute the vertex coordinates of a triangle given its side lengths.
- function triangle_ppp2ls (v1, v2, v3)

  Compute the side lengths of a triangle given its vertex coordinates.
- function triangle_lp2ls (v)

Compute the side lengths of a triangle given its vertex coordinates.

- function triangle_area_ppp (v1, v2, v3, s=false)

  Compute the signed area of a triangle given its vertex coordinates.
- function triangle_area_lp (v, s=false)

  Compute the signed area of a triangle given its vertex coordinates.
- function triangle_centroid_ppp (v1, v2, v3)

  Compute the centroid (geometric center) of a triangle.
- function triangle_centroid_lp (v)

  Compute the centroid (geometric center) of a triangle.
- function triangle_incenter_ppp (v1, v2, v3)

  Compute the coordinate for the triangle's incircle.
- function triangle_incenter_lp (v)

  Compute the coordinate for the triangle's incircle.
- function triangle_inradius_ppp (v1, v2, v3)

  Compute the inradius of a triangle's incircle.
- function triangle_inradius_lp (v)

  Compute the inradius of a triangle's incircle.
- function triangle_is_cw_ppp (v1, v2, v3)

  Test the vertex ordering, or orientation, of a triangle.
- function triangle_is_cw_lp (v)

  Test the vertex ordering, or orientation, of a triangle.
- function triangle_is_pit_ppp (v1, v2, v3, t)

  Test if a point is inside a triangle in a Euclidean 2d-space using Barycentric.
- function triangle_is_pit_lp (v, t)

  Test if a point is inside a triangle in a Euclidean 2d-space using Barycentric.

10.29.1 Detailed Description

Triangle solutions mathematical functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

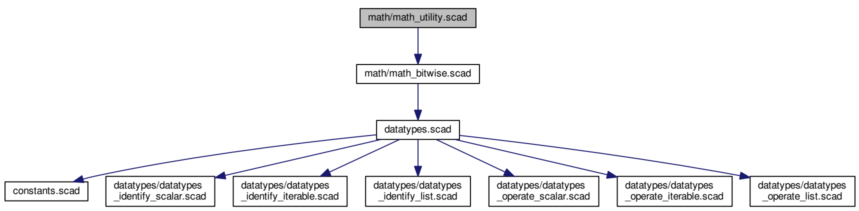
Include this library file using the **include** statement.

10.30 math/math_utility.scad File Reference

Miscellaneous mathematical utilities.

```
#include <math/math_bitwise.scad>
```

Include dependency graph for math_utility.scad:



Functions

- function hist (v, m=0, cs, cb, cp, ca, cf, d=false)

    Generate a histogram for the elements of a list of values.

10.30.1 Detailed Description

Miscellaneous mathematical utilities.

Author

Roy Allen Sutton

Date

2017

Copyright

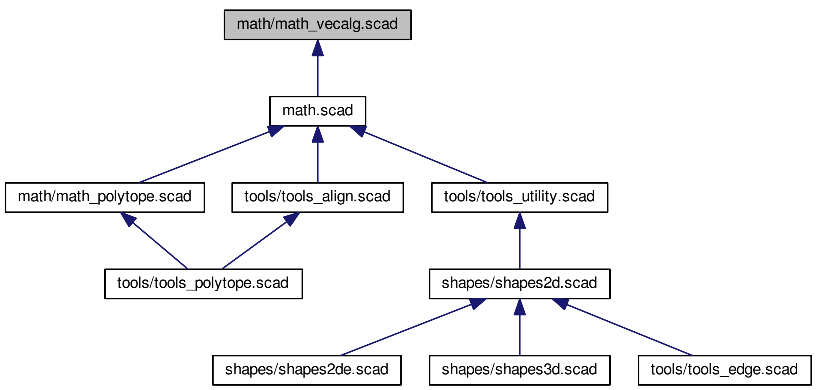
Note

Include this library file using the **include** statement.

10.31 math/math_vecalg.scad File Reference

Vector algebra mathematical functions.

This graph shows which files directly or indirectly include this file:



Functions

- function distance_pp (p1, p2)

  Compute the distance between two Euclidean points.
- function is_left_ppp (p1, p2, p3)

  Test if a point is left, on, or right of an infinite line in a Euclidean 2d-space.
- function dimension_2to3_v (v)

  Return 3d vector unchanged or add a zeroed third dimension to 2d vector.
- function get_line_dim (l)

  Return the number of dimensions of a Euclidean line (or vector).
- function get_line_tp (l)

  Return the terminal point of a Euclidean line (or vector).
- function get_line_ip (l)

  Return the initial point of a Euclidean line (or vector).
- function get_line2origin (l)

  Shift a Euclidean line (or vector) to the origin.

- function dot_ll (l1, l2)

    Compute the dot product of two lines (or vectors).
- function cross_ll (l1, l2)

    Compute the cross product of two lines (or vectors) in a Euclidean 3d or 2d-space.
- function striple_lll (l1, l2, l3)

    Compute the scalar triple product of three lines (or vectors) in a Euclidean 3d or 2d-space.
- function angle_ll (l1, l2)

    Compute the angle between two lines (or vectors) in a Euclidean 3d or 2d-space.
- function angle_lll (l1, l2, n)

    Compute the angle between two lines (or vectors) in a Euclidean 3d-space.
- function unit_l (l)

    Compute the normalized unit vector of a Euclidean line (or vector).
- function are_coplanar_lll (l1, l2, l3, d=6)

    Test if three lines (or vectors) are coplanar in Euclidean 3d-space.
- function get_pnorm2nv (pn, cw=true)

    Convert a planes' normal specification into a normal vector.

10.31.1 Detailed Description

Vector algebra mathematical functions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

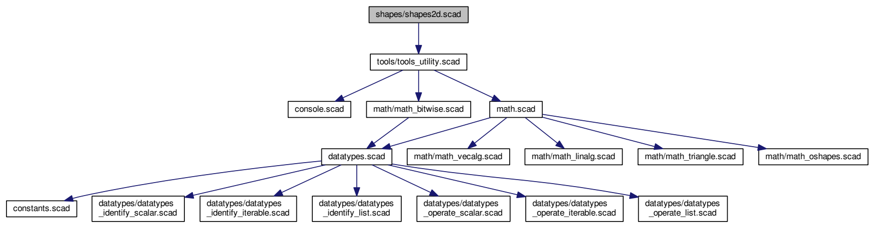
Note

Include this library file using the **include** statement.

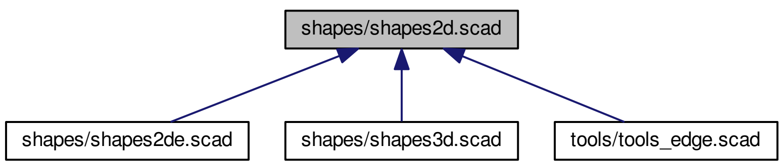
10.32 shapes/shapes2d.scad File Reference

Two-dimensional basic shapes.

`#include <tools/tools_utility.scad>`
Include dependency graph for shapes2d.scad:



This graph shows which files directly or indirectly include this file:



Functions

- module rectangle (size, vr, vrm=0, center=false)

  A rectangle with edge, fillet, and/or chamfer corners.
- module rectangle_c (size, core, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)

  A rectangle with a removed rectangular core.
- module rhombus (size, vr, center=false)

  A rhombus.
- module triangle_ppp (v1, v2, v3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by three vertices.
- module triangle_lp (v, vr, centroid=false, incenter=false)

  A general triangle specified by a list of its three vertices.
- module triangle_sss (s1, s2, s3, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by its three side lengths.
- module triangle_ls (v, vr, centroid=false, incenter=false)

  A general triangle specified by a list of its three side lengths.
- module triangle_ls_c (vs, vc, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false)

  A general triangle specified by its sides with a removed triangular core.
- module triangle_sas (s1, a, s2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

A general triangle specified by two sides and the included angle.

- module triangle_asa (a1, s, a2, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by a side and two adjacent angles.
- module triangle_aas (a1, a2, s, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A general triangle specified by a side, one adjacent angle and the opposite angle.
- module triangle_ss (x, y, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A right-angled triangle specified by its opposite and adjacent side lengths.
- module triangle_sa (x, y, aa, oa, vr, v1r, v2r, v3r, centroid=false, incenter=false)

  A right-angled triangle specified by a side length and an angle.
- module ngon (n, r, vr)

  An n-sided equiangular/equilateral regular polygon.
- module ellipse (size)

  An ellipse.
- module ellipse_c (size, core, t, co, cr=0)

  An ellipse with a removed elliptical core.
- module ellipse_s (size, a1=0, a2=0)

  An ellipse sector.
- module ellipse_cs (size, core, t, a1=0, a2=0, co, cr=0)

  A sector of an ellipse with a removed elliptical core.
- module star2d (size, n=5, vr)

  A two-dimensional star.

10.32.1 Detailed Description

Two-dimensional basic shapes.

Author

Roy Allen Sutton

Date

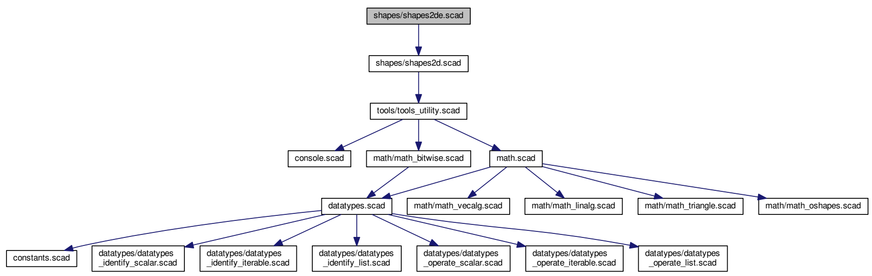
2015-2017

Copyright

10.33 shapes/shapes2de.scad File Reference

Linearly extruded two-dimensional basic shapes.

`#include <shapes/shapes2d.scad>`
Include dependency graph for shapes2de.scad:



Functions

- module erectangle (size, h, vr, vrm=0, center=false)

  An extruded rectangle with edge, fillet, and/or chamfer corners.
- module erectangle_c (size, core, h, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, center=false)

  An extruded rectangle with a removed rectangular core.
- module erhombus (size, h, vr, center=false)

  An extruded rhombus.
- module etriangle_ppp (v1, v2, v3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by three vertices.
- module etriangle_lp (v, h, vr, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a list of its three vertices.
- module etriangle_sss (s1, s2, s3, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by its three side lengths.
- module etriangle_ls (v, h, vr, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a list of its three side lengths.
- module etriangle_ls_c (vs, vc, h, co, cr=0, vr, vr1, vr2, centroid=false, incenter=false, center=false)

  A general triangle specified by its sides with a removed triangular core.
- module etriangle_sas (s1, a, s2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by two sides and the included angle.
- module etriangle_asa (a1, s, a2, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a side and two adjacent angles.
- module etriangle_aas (a1, a2, s, h, x=1, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded general triangle specified by a side, one adjacent angle and the opposite angle.
- module etriangle_ss (x, y, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded right-angled triangle specified by its opposite and adjacent side lengths.
- module etriangle_sa (x, y, aa, oa, h, vr, v1r, v2r, v3r, centroid=false, incenter=false, center=false)

  An extruded right-angled triangle specified by a side length and an angle.
- module engon (n, r, h, vr, center=false)

  An extruded n-sided equiangular/equilateral regular polygon.

- module eellipse (size, h, center=false)

  An extruded ellipse.
- module eellipse_c (size, core, h, t, co, cr=0, center=false)

  An extruded ellipse with a removed elliptical core.
- module eellipse_s (size, h, a1=0, a2=0, center=false)

  An extruded ellipse sector.
- module eellipse_cs (size, core, h, t, a1=0, a2=0, co, cr=0, center=false)

  An extruded sector of an ellipse with a removed elliptical core.
- module estar2d (size, h, n=5, vr, center=false)

  An extruded two-dimensional star.

10.33.1 Detailed Description

Linearly extruded two-dimensional basic shapes.

Author

Roy Allen Sutton

Date

2015-2017

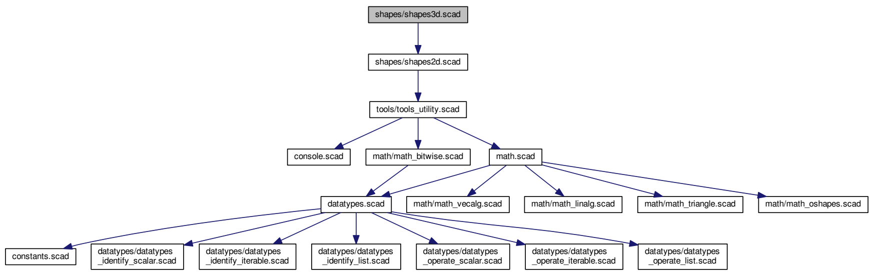
Copyright

10.34 shapes/shapes3d.scad File Reference

Three-dimensional basic shapes.

```
#include <shapes/shapes2d.scad>
```

Include dependency graph for shapes3d.scad:



Functions

- module cone (r, h, d, vr, vr1, vr2)

    A cone.
- module cuboid (size, vr, vrm=0, center=false)

    A cuboid with edge, fillet, or chamfer corners.
- module ellipsoid (size)

    An ellipsoid.
- module ellipsoid_s (size, a1=0, a2=0)

    A sector of an ellipsoid.
- module pyramid_t (size, center=false)

    A pyramid with trilateral base formed by four equilateral triangles.
- module pyramid_q (size, center=false)

    A pyramid with quadrilateral base.
- module star3d (size, n=5, half=false)

    A three-dimensional star.
- module torus_rp (size, core, r, l, t, co, cr=0, vr, vr1, vr2, vrm=0, vrm1, vrm2, pa=0, ra=360, m=255, center=false, profile=false)

    A rectangular cross-sectional profile revolved about the z-axis.
- module torus_tp (size, core, r, l, co, cr=0, vr, vr1, vr2, pa=0, ra=360, m=255, centroid=false, incenter=false, profile=false,)

    A triangular cross-sectional profile revolved about the z-axis.
- module torus_ep (size, core, r, l, t, a1=0, a2=0, co, cr=0, pa=0, ra=360, m=255, profile=false)

    An elliptical cross-sectional profile revolved about the z-axis.

10.34.1 Detailed Description

Three-dimensional basic shapes.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

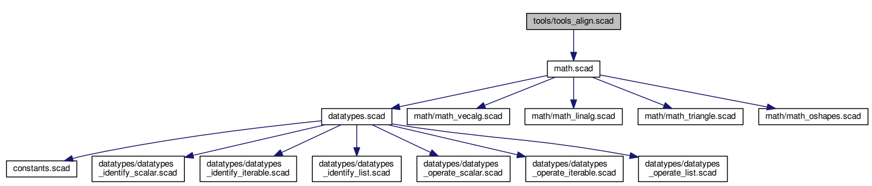
Todo Complete rounded cylinder.

10.35 tools/tools_align.scad File Reference

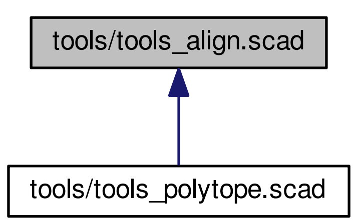
Shape alignment tools.

```
#include <math.scad>
```

Include dependency graph for tools_align.scad:



This graph shows which files directly or indirectly include this file:



Functions

- module orient_ll (l=z_axis3d_ul, rl=z_axis3d_ul, r=0)

  Orient a line or vector to a reference line or vector.
- module align_ll (l=z_axis3d_ul, rl=z_axis3d_ul, ap=0, rp=0, r=0, to=origin3d, ro=zero3d)

  Align a line or vector to a reference line or vector.
- module align_l (rl=z_axis3d_ul, rp=0, r=0, to=origin3d, ro=zero3d, d=z_axis_ci)

  Align a shapes' x, y, or z Cartesian axis to reference line or vector.

10.35.1 Detailed Description

Shape alignment tools.

Author

Roy Allen Sutton

Date

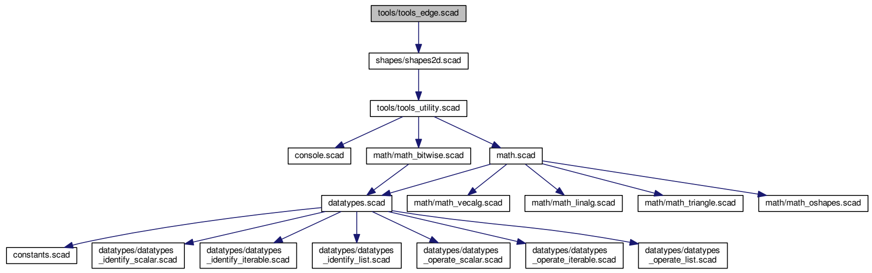
2017

Copyright

10.36 tools/tools_edge.scad File Reference

Shape edge finishing tools.

#include <shapes/shapes2d.scad>
Include dependency graph for tools_edge.scad:



Functions

- module edge_profile_r (r, p=0, f=1, a=90,)

  A 2d edge-finish profile specified by intersection radius.
- module edge_add_r (r, l=1, p=0, f=1, m=3, ba=45, a1=0, a2=90, center=false)

  A 3d edge-finish additive shape specified by intersection radius.

10.36.1 Detailed Description

Shape edge finishing tools.

Author

Roy Allen Sutton

Date

2017

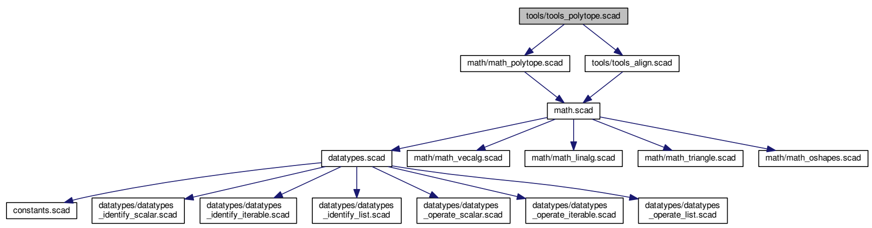
Copyright

10.37 tools/tools_polytope.scad File Reference

Polygon and polyhedron tools.

```
#include <math/math_polytope.scad>
#include <tools/tools_align.scad>
```

Include dependency graph for tools_polytope.scad:



Functions

- module polytope_number (c, f, e, vi=true, fi=true, ei=true, sp=false, ts, th, to, tr=0)

    Label the vertices, paths, and edges of a polytope.
- module polytope_frame (c, f, e, vi=true, fi=true, ei=true, vc=1, fc=2, ec=0)

    Assemble a polytope skeletal frame using child objects.
- module polytope_bbox (c, f, a)

    The 3d or 2d bounding box shape for a polytope.

10.37.1 Detailed Description

Polygon and polyhedron tools.

Author

Roy Allen Sutton

Date

2017

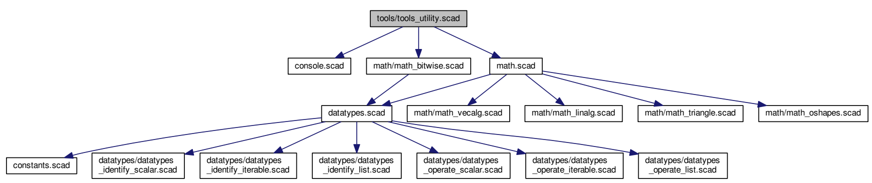
Copyright

10.38 tools/tools_utility.scad File Reference

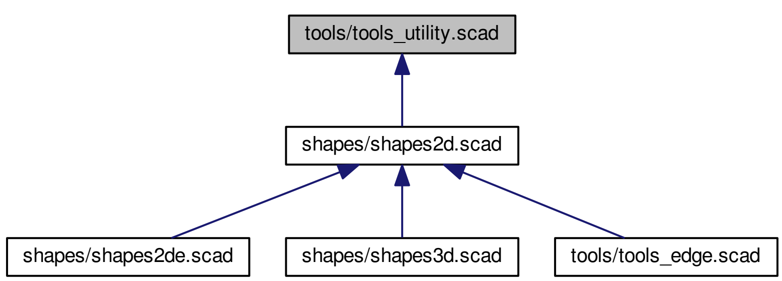
Shape transformation utility tools.

```
#include <console.scad>
#include <math.scad>
#include <math/math_bitwise.scad>
```

Include dependency graph for tools_utility.scad:



This graph shows which files directly or indirectly include this file:



Functions

- module rotate_extrude_tr (r, pa=0, ra=360, profile=false)

  Translate, rotate, and revolve the 2d shape about the z-axis.
- module rotate_extrude_tre (r, l, pa=0, ra=360, m=255, profile=false)

  Translate, rotate, and revolve the 2d shape about the z-axis with linear elongation.
- module linear_extrude_uls (h, center=false)

  Linearly extrude 2d shape with extrusion upper and lower scaling.
- module radial_repeat (n, r=1, angle=true, move=false)

  Distribute copies of a 2d or 3d shape equally about a z-axis radius.
- module grid_repeat (g, i, c=1, center=false)

  Distribute copies of 2d or 3d shapes about Cartesian grid.

10.38.1 Detailed Description

Shape transformation utility tools.

Author

Roy Allen Sutton

Date

2015-2017

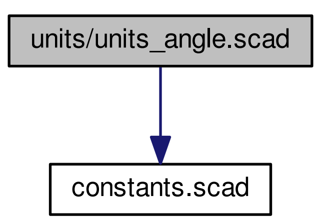
Copyright

10.39 units/units_angle.scad File Reference

Angle units and conversions.

`#include <constants.scad>`

Include dependency graph for units_angle.scad:



Functions

- function unit_angle_name (u=base_unit_angle)

Return the name of an angle unit identifier.

- function convert_angle (a, from=base_unit_angle, to=base_unit_angle)

  Convert an angle from some units to another.

Variables

- base_unit_angle = "d"

  <string> The base units for angle measurements.

10.39.1 Detailed Description

Angle units and conversions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

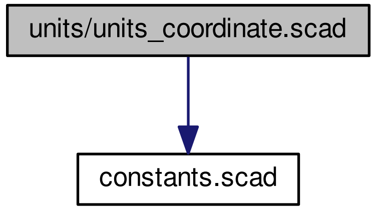
Include this library file using the **include** statement.

10.40 units/units_coordinate.scad File Reference

Coordinate systems and conversions.

```
#include <constants.scad>
```

Include dependency graph for units_coordinate.scad:



Functions

- function coordinates_name (s=base_coordinates)

  Return the name of the given coordinate system identifier.
- function convert_coordinate (c, from=base_coordinates, to=base_coordinates)

  Convert point from one coordinate system to another.
- function coordinates_cpc (c, r, t=false)

  Radially scale a list of 2d cartesian coordinates.
- function coordinates_pc (p, r, t=false)

  Radially scale and convert a list of 2d polar coordinates to cartesian.
- function coordinates_csc (c, r, t=false)

  Radially scale a list of 3d cartesian coordinates.
- function coordinates_sc (s, r, t=false)

  Radially scale and convert a list of 3d spherical coordinates to cartesian.

Variables

- base_coordinates = "c"

  <string> The base coordinate system.
- coordinates_positive_angles = true

  <boolean> When converting to angular measures add 360 to negative angles.

10.40.1 Detailed Description

Coordinate systems and conversions.

Author

Roy Allen Sutton

Date

2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

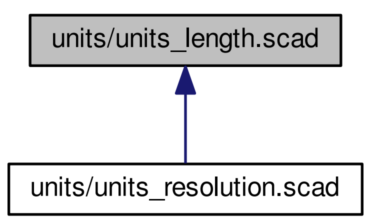
Note

Include this library file using the **include** statement.

10.41 units/units_length.scad File Reference

Length units and conversions.

This graph shows which files directly or indirectly include this file:



Functions

- function unit_length_name (u=base_unit_length, d=1, w=false)

  Return the name for a length unit identifier with dimension.
- function convert_length (v, from=base_unit_length, to=base_unit_length, d=1)

  Convert a value from from one units to another with dimensions.

Variables

- base_unit_length = "mm"

  <string> The base unit for length measurements.

10.41.1 Detailed Description

Length units and conversions.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Note

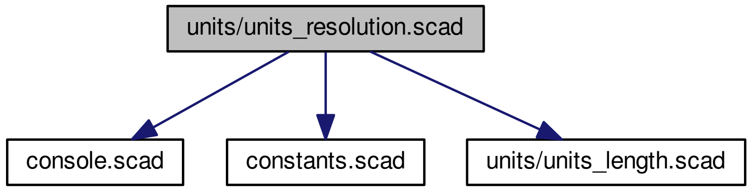
Include this library file using the **include** statement.

10.42 units/units_resolution.scad File Reference

An abstraction for arc rendering resolution control.

```
#include <console.scad>
#include <constants.scad>
#include <units/units_length.scad>
```

Include dependency graph for units_resolution.scad:



Functions

- function resolution_fn (r)

  Return facets number for the given arc radius.
- function resolution_fs ()

  Return minimum facets size.
- function resolution_fa (r)

  Return the minimum facets angle.
- function resolution_reduced ()

  Return the radius at which arc resolution will begin to degrade.
- module resolution_info (r)

  Output resolution information to the console for given radius.
- function resolution_facets (r)

  Return facet count used to render a radius.
- function resolution_facetsv (r)

  Return facet count information list used to render a radius.

Variables

- $resolution_mode = "fast"

  <string> Global special variable that configures the arc resolution mode.
- $resolution_value = 0

  <number> Global special variable for modes that use custom resolutions.

10.42.1 Detailed Description

An abstraction for arc rendering resolution control.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

This file is part of omdl, an OpenSCAD mechanical design library.

The *omdl* is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The *omdl* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with the *omdl*; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA; or see http://www.gnu.org/licenses/.

Note

Include this library file using the **include** statement.

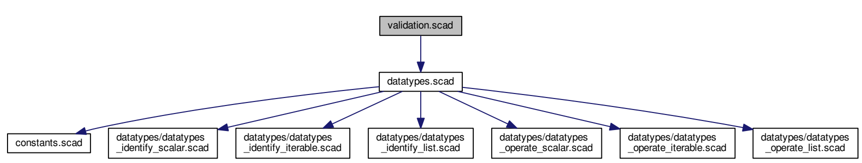
Test Review model for accuracy.

10.43 validation.scad File Reference

Function validation methods.

```
#include <datatypes.scad>
```

Include dependency graph for validation.scad:



Functions

- function validate (d, cv, t, ev, p=4, pf=false)

  Compare a computed test value with an known good result.

10.43.1 Detailed Description

Function validation methods.

Author

Roy Allen Sutton

Date

2015-2017

Copyright

Index