# SolidPython Documentation

***Release 0.1.2***

**Evan Jones**

**Jun 11, 2017**

# Contents

Contents:

# CHAPTER 1

# SolidPython



**Table of Contents** generated with DocToc

CHAPTER 2

# SolidPython: OpenSCAD for Python

SolidPython is a generalization of Phillip Tiefenbacher's openscad module, found on Thingiverse. It generates valid OpenSCAD code from Python code with minimal overhead. Here's a simple example:

This Python code:

```
from solid import *
d = difference()(
    cube(10),
    sphere(15)
)
print(scad_render(d))
```

Generates this OpenSCAD code:

```
difference(){
    cube(10);
    sphere(15);
}
```

That doesn't seem like such a savings, but the following SolidPython code is a lot shorter (and I think clearer) than the SCAD code it compiles to:

```
from solid import *
from solid.utils import *
d = cube(5) + right(5)(sphere(5)) - cylinder(r=2, h=6)
```

Generates this OpenSCAD code:

```
difference(){
    union(){
        cube(5);
        translate( [5, 0,0]){
            sphere(5);
        }
    }
```

```
    cylinder(r=2, h=6);
}
```

CHAPTER 3

# Advantages

Because you're using Python, a lot of things are easy that would be hard or impossible in pure OpenSCAD. Among these are:

- built-in dictionary types
- mutable, slice-able list and string types
- recursion
- external libraries (images! 3D geometry! web-scraping! ...)

# CHAPTER 4

# Installing SolidPython

- Install via PyPI:

```
pip install solidpython
```

  (You may need to use `sudo pip install solidpython`, depending on your environment. This is commonly discouraged though.)

- **OR:** Download SolidPython (Click here to download directly, or use git to pull it all down)

  (Note that SolidPython also depends on the PyEuclid Vector math library, installable via `pip install euclid3`)

  - Unzip the file, probably in ~/Downloads/SolidPython-master
  - In a terminal, cd to location of file:

```
cd ~/Downloads/SolidPython-master
```

  - Run the install script:

```
python setup.py install
```

# CHAPTER 5

# Using SolidPython

- Include SolidPython at the top of your Python file:

```
from solid import *
from solid.utils import *  # Not required, but the utils module is useful
```

- To include other scad code, call `use("/path/to/scadfile.scad")` or `include("/path/to/scadfile.scad")`. This is identical to what you would do in OpenSCAD.
- OpenSCAD uses curly-brace blocks ({}) to create its tree. SolidPython uses parentheses with comma-delimited lists. **OpenSCAD:**

```
difference(){
    cube(10);
    sphere(15);
}
```

**SolidPython:**

```
d = difference()(
    cube(10),  # Note the comma between each element!
    sphere(15)
)
```

- Call `scad_render(py_scad_obj)` to generate SCAD code. This returns a string of valid OpenSCAD code.
- *or*: call `scad_render_to_file(py_scad_obj, filepath)` to store that code in a file.
- If 'filepath' is open in the OpenSCAD IDE and Design => 'Automatic Reload and Compile' is checked (in the OpenSCAD IDE), calling `scad_render_to_file()` from Python will load the object in the IDE.
- Alternately, you could call OpenSCAD's command line and render straight to STL.

CHAPTER 6

# Example Code

The best way to learn how SolidPython works is to look at the included example code. If you've installed SolidPython, the following line of Python will print(the location of ) the examples directory:

```
import os, solid; print(os.path.dirname(solid.__file__) + '/examples')
```

Or browse the example code on Github here

Adding your own code to the example file solid/examples/solidpython_template.py will make some of the setup easier.

CHAPTER 7

# Extra syntactic sugar

## Basic operators

Following Elmo Mäntynen's suggestion, SCAD objects override the basic operators + (union), - (difference), and * (intersection). So

```
c = cylinder(r=10, h=5) + cylinder(r=2, h=30)
```

is the same as:

```
c = union()(
    cylinder(r=10, h=5),
    cylinder(r=2, h=30)
)
```

Likewise:

```
c = cylinder(r=10, h=5)
c -= cylinder(r=2, h=30)
```

is the same as:

```
c = difference()(
    cylinder(r=10, h=5),
    cylinder(r=2, h=30)
)
```

## First-class Negative Space (Holes)

OpenSCAD requires you to be very careful with the order in which you add or subtract objects. SolidPython's `hole()` function makes this process easier.

Consider making a joint where two pipes come together. In OpenSCAD you need to make two cylinders, union them, then make two smaller cylinders, union them, then subtract the smaller from the larger.

Using hole(), you can make a pipe, specify that its center should remain open, and then add two pipes together knowing that the central void area will stay empty no matter what other objects are added to that structure.

Example:

```
outer = cylinder(r=pipe_od, h=seg_length)
inner = cylinder(r=pipe_id, h=seg_length)
pipe_a = outer - hole()(inner)
```

Once you've made something a hole, eventually you'll want to put something, like a bolt, into it. To do this, we need to specify that there's a given 'part' with a hole and that other parts may occupy the space in that hole. This is done with the `part()` function.

See solid/examples/hole_example.py for the complete picture.

## Animation

OpenSCAD has a special variable, `$t`, that can be used to animate motion. SolidPython can do this, too, using the special function `scad_render_animated_file()`.

See solid/examples/animation_example.py for more details.

CHAPTER 8

# solid.utils

SolidPython includes a number of useful functions in solid/utils.py. Currently these include:

## Directions: (up, down, left, right, forward, back) for arranging things:

```
up(10)(
    cylinder()
)
```

seems a lot clearer to me than:

```
translate( [0,0,10])(
    cylinder()
)
```

I took this from someone's SCAD work and have lost track of the original author.
My apologies.

## Arcs

I've found this useful for fillets and rounds.

```
arc(rad=10, start_degrees=90, end_degrees=210)
```

draws an arc of radius 10 counterclockwise from 90 to 210 degrees.

```
arc_inverted(rad=10, start_degrees=0, end_degrees=90)
```

draws the portion of a 10x10 square NOT in a 90 degree circle of radius 10. This is the shape you need to add to make fillets or remove to make rounds.

## Offsets

To offset a set of points in one direction or another (inside or outside a closed figure, for example) use `solid.utils.offset_points(point_arr, offset, inside=True)`

Note that, for a non-convex figure, inside and outside may be non-intuitive. The simple solution is to manually check that your offset is going in the direction you intend, and change the boolean value of `inside` if you're not happy.

See the code for futher explanation. Improvements on the inside/outside algorithm would be welcome.

## Extrude Along Path

`solid.utils.extrude_along_path(shape_pts, path_pts, scale_factors=None)`

See solid/examples/path_extrude_example.py for use.

## Basic color library

You can change an object's color by using the OpenSCAD `color([rgba_array])` function:

```
transparent_blue = color([0,0,1, 0.5])(cube(10))   # Specify with RGB[A]
red_obj = color(Red)(cube(10))                     # Or use predefined colors
```

These colors are pre-defined in solid.utils:

| | | |
|---|---|---|
| Red | Green | Blue |
| Cyan | Magenta | Yellow |
| Black | White | Transparent |
| Oak | Pine | Birch |
| Iron | Steel | Stainless |
| Aluminum | Brass | BlackPaint |
| FiberBoard | | |

They're a conversion of the materials in the MCAD OpenSCAD library, as seen [here] (https://github.com/openscad/MCAD/blob/master/materials.scad).

## Bill Of Materials

Put `@bom_part()` before any method that defines a part, then call `bill_of_materials()` after the program is run, and all parts will be counted, priced and reported.

The example file solid/examples/bom_scad.py illustrates this. Check it out.

## solid.screw_thread

solid.screw_thread includes a method, thread() that makes internal and external screw threads.

See solid/examples/screw_thread_example.py for more details.

CHAPTER 9

# Contact

Enjoy, and please send any questions or bug reports to me at `evan_t_jones@mac.com`.

Cheers!

Evan

CHAPTER 10

# License

This library 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.

This library 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.

Full text of the license.

Some class docstrings are derived from the OpenSCAD User Manual, so are available under the Creative Commons Attribution-ShareAlike License.

# CHAPTER 11

# Library Reference

solid.solidpython.**calling_module**(*stack_depth=2*)

Returns the module *2* back in the frame stack. That means: code in module A calls code in module B, which asks calling_module() for module A.

This means that we have to know exactly how far back in the stack our desired module is; if code in module B calls another function in module B, we have to increase the stack_depth argument to account for this.

Got that?

solid.utils.**frange**(*[start], end[, step[, mode]]*) → generator

A float range generator. If not specified, the default start is 0.0 and the default step is 1.0.

Optional argument mode sets whether frange outputs an open or closed interval. mode must be an int. Bit zero of mode controls whether start is included (on) or excluded (off); bit one does the same for end. Hence:

> 0 -> open interval (start and end both excluded) 1 -> half-open (start included, end excluded) 2 -> half open (start excluded, end included) 3 -> closed (start and end both included)

By default, mode=1 and only start is included in the output.

CHAPTER 12

# Indices and tables

- genindex
- modindex
- **search**

  **members**

# Python Module Index

## S

# Index

## C



## F



## S