

---

# **pyScss Documentation**

*Release 1.2.0*

**German M. Bravo (Kronuz)**

October 08, 2013



# CONTENTS



pyScss compiles Scss (Sass), a superset of CSS that is more powerful, elegant and easier to maintain than plain-vanilla CSS. The library acts as a CSS source code preprocessor which allows you to use variables, nested rules, mixins, and have inheritance of rules, all with a CSS-compatible syntax which the preprocessor then compiles to standard CSS.

Scss, as an extension of CSS, helps keep large stylesheets well-organized. It borrows concepts and functionality from projects such as OOCSS and other similar frameworks like as Sass. It's build on top of the original PHP xCSS codebase structure but it's been completely rewritten, many bugs have been fixed and it has been extensively extended to support almost the full range of Sass' Scss syntax and functionality. Contents:



# INSTALLATION AND USAGE

## 1.1 Installation

pyScss requires only Python 2.5 or later, including Python 3.x. PyPy is also known to work. Install with pip:

```
pip install pyScss
```

Its lone dependency is the `six` library, which pip should install for you.

## 1.2 Usage

Run from the command line by using `-m`:

```
python -mscss < file.scss
```

Specify directories to search for imports with `-I`. See `python -mscss --help` for more options.

---

**Note:** `-mscss` will only work in Python 2.7 and above. For Python 2.5 and 2.6, `-m` doesn't work with packages, and you need to invoke:

```
python -mscss.tool
```

---

## 1.3 Interactive mode

To get a REPL:

```
python -mscss --interactive
```

Example session:

```
$ python scss.py --interactive
>>> @import "compass/css3"
>>> show()
['functions', 'mixins', 'options', 'vars']
>>> show(mixins)
['apply-origin',
 'apply-transform',
 ...]
```

```
'transparent']
>>> show(mixins, transparent)
@mixin transparent() {
  @include opacity(0);
}
>>> 1px + 5px
6px
>>> _
```

## 1.4 Compass example

With `--load-path` or `scss.config.LOAD_PATHS` set to Compass and Blueprint roots, you can compile with Compass like with the following:

```
@option compress: no;

$blueprint-grid-columns : 24;
$blueprint-grid-width   : 30px;
$blueprint-grid-margin  : 10px;
$font-color             : #333;

@import "compass/reset";
@import "compass/utilities";
@import "blueprint";

// your code...
```

# PYTHON API

## 2.1 Compiling files

Very basic usage is simple enough:

```
from scss import Scss
css = Scss()
css.compile("a { color: red + green; }")
```

## 2.2 Configuration

There are several configuration variables in the `scss.config` module that you may wish to change.

**PROJECT\_ROOT:** Root of your entire project. Used only to construct defaults for other variables. Defaults to the root of the pyScss installation, which is probably not what you want.

**LOAD\_PATHS:** An iterable of paths to search when using “@import”.

**STATIC\_ROOT:** Used for finding sprite files. Defaults to `$PROJECT_ROOT/static`.

**ASSETS\_ROOT:** Generated sprites are saved here. Defaults to `$STATIC_ROOT/assets`.

**CACHE\_ROOT:** Used for storing cached sprite information. Defaults to `ASSETS_ROOT`.

**STATIC\_URL:** URL equivalent to `STATIC_ROOT`. Defaults to `static/`.

**ASSETS\_URL:** URL equivalent to `ASSETS_ROOT`. Defaults to `static/assets/`.

**SPRTE\_MAP\_DIRECTION:** Direction in which to arrange sprites in a spritesheet. Defaults to `vertical`; may be changed to `horizontal`, `diagonal`, or `smart`.

**VERBOSITY:** Increase spew from the compiler. Defaults to 1.

**DEBUG:** Set to true to make parse errors fatal. Defaults to false.

**Warning:** Configuration via monkeypatching is fraught with issues. If you don’t need the Compass sprite functionality, stick with passing `search_paths` to the `Scss` constructor, and don’t touch these variables at all. The current plan is to introduce a new mechanism for Compass configuration in 1.3 with deprecation warnings, and remove `scss.config` entirely in 2.0.

## 2.3 Django example

A rough example of using pyScss with Django:

```
import os
import fnmatch

import scss

from django.conf import settings
from django.utils.datastructures import SortedDict
from django.contrib.staticfiles import finders

def finder(glob):
    """
    Finds all files in the django finders for a given glob,
    returns the file path, if available, and the django storage object.
    storage objects must implement the File storage API:
    https://docs.djangoproject.com/en/dev/ref/files/storage/
    """
    for finder in finders.get_finders():
        for path, storage in finder.list([]):
            if fnmatch.fnmatchcase(path, glob):
                yield path, storage

# STATIC_ROOT is where pyScss looks for images and static data.
# STATIC_ROOT can be either a fully qualified path name or a "finder"
# iterable function that receives a filename or glob and returns a tuple
# of the file found and its file storage object for each matching file.
# (https://docs.djangoproject.com/en/dev/ref/files/storage/)
scss.config.STATIC_ROOT = finder
scss.config.STATIC_URL = settings.STATIC_URL

# ASSETS_ROOT is where the pyScss outputs the generated files such as spritemaps
# and compile cache:
scss.config.ASSETS_ROOT = os.path.join(settings.MEDIA_ROOT, 'assets/')
scss.config.ASSETS_URL = settings.MEDIA_URL + 'assets/'

# These are the paths pyScss will look ".scss" files on. This can be the path to
# the compass framework or blueprint or compass-recepies, etc.
scss.config.LOAD_PATHS = [
    '/usr/local/www/sass/frameworks/',
    '/Library/Ruby/Gems/1.8/gems/compass-0.11.5/frameworks/compass/stylesheets/',
    '/Library/Ruby/Gems/1.8/gems/compass-0.11.5/frameworks/blueprint/stylesheets/',
]

# This creates the Scss object used to compile SCSS code. In this example,
# _scss_vars will hold the context variables:
_scss_vars = {}
_scss = scss.Scss(
    scss_vars=_scss_vars,
    scss_opts={
        'compress': True,
        'debug_info': True,
    }
)
```

```
# 1. Compile from a string:
compiled_css_from_string = _scss.compile('@import "file2"; a {color: red + green; }')

# 2. Compile from a file:
compiled_css_from_file = _scss.compile(scss_file='file1.scss')

# 3. Compile from a set of files (use SortedDict or collections.OrderedDict to
# maintain the compile order):
_scscss._scss_files = SortedDict((
    ('file2.scss', open('file2.scss').read()),
    ('file3.scss', open('file3.scss').read()),
    ('file4.scss', open('file4.scss').read()),
))
compiled_css_from_files = _scss.compile()
```

---

**Note:** The API here is likely to be improved in 1.3, to avoid the need for calling underscored functions.

---

## 2.4 Extending pyScss

There is some support for adding custom functions from Python, but the API is explicitly undocumented and subject to change. Watch this space.



# PYSCSS SYNTAX

## 3.1 Supported Sass features

pyScss is mostly compatible with Sass 3.2 and has partial support for the upcoming Sass 3.3. The canonical syntax reference is in the Sass documentation: [http://sass-lang.com/docs/yardoc/file.SASS\\_REFERENCE.html](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html)

### 3.1.1 Both syntaxes

SCSS (CSS3 superset) is the primary syntax, but there's experimental support for the SASS (YAML-like) syntax.

### 3.1.2 Built-in functions

All of the Sass 3.2 functions described in the Sass documentation are supported.

### 3.1.3 Rule nesting

Rule/selector nesting and the & parent-reference selector are both supported.

Example:

```
.selector {  
  a {  
    display: block;  
  }  
  strong {  
    color: blue;  
  }  
}
```

Produces:

```
.selector a {  
  display: block;  
}  
.selector strong {  
  color: blue;  
}
```

### 3.1.4 Variables, data types

Variables are supported. All of the Sass data types—strings, numbers, booleans, colors, lists, maps, and null—are supported.

Example:

```
$main-color: #ce4dd6;
$style: solid;
$side: bottom;
#navbar {
  border-#{$side}: {
    color: $main-color;
    style: $style;
  }
}
```

Produces:

```
#navbar {
  border-bottom-color: #ce4dd6;
  border-bottom-style: solid;
}
```

### 3.1.5 Functions and mixins

@function, @mixin, and @include (optionally with @content) are supported.

Named arguments (foo(\$name: value)) and slurpy arguments (foo(\$args...)) are also supported.

Example:

```
@mixin rounded($side, $radius: 10px) {
  border-#{$side}-radius: $radius;
  -moz-border-radius-#{$side}: $radius;
  -webkit-border-#{$side}-radius: $radius;
}
#navbar li { @include rounded(top); }
#footer { @include rounded(top, 5px); }
#sidebar { @include rounded(left, 8px); }
```

Produces:

```
#navbar li {
  border-top-radius: 10px;
  -moz-border-radius-top: 10px;
  -webkit-border-top-radius: 10px;
}
#footer {
  border-top-radius: 5px;
  -moz-border-radius-top: 5px;
  -webkit-border-top-radius: 5px;
}
#sidebar {
  border-left-radius: 8px;
  -moz-border-radius-left: 8px;
  -webkit-border-left-radius: 8px;
}
```

### 3.1.6 Rule extension

@extend is supported, though some particularly thorny edge cases may not produce output identical to the reference compiler.

Example:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.error.intrusion {
  background-image: url("/image/hacked.png");
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
```

Produces:

```
.error,
.seriousError {
  border: 1px red;
  background-color: #fdd;
}
.error.intrusion,
.seriousError.intrusion {
  background-image: url("/image/hacked.png");
}
.seriousError {
  border-width: 3px;
}
```

### 3.1.7 Conditions

@if, @else if, and @else are supported.

### 3.1.8 Loops

Both types of iteration are supported:

```
@for $n from 1 through 9 {
  .span-#{$n} { width: $n * 10%; }
}

@each $color in red, blue, yellow {
  .button-#{$color} {
    background-color: $color;
  }
}
```

Additionally, the unpacking-iteration syntax in Sass trunk is supposed; see *Maps*.

### 3.1.9 Maps

pyScss has experimental support for maps, a data type recently added to Sass trunk. Maps are defined with colons inside parentheses:

```
$colors: (  
  text: black,  
  background: white  
);
```

Keys may be any Sass expression, not just strings.

Maps are manipulated with a handful of map functions:

```
a {  
  color: map-get($colors, text);  
  background-color: map-get($colors, background);  
}
```

A map is semantically equivalent to a list of 2-lists, stored in the order they appeared when the map was defined. Any list operation will work on a map:

```
div {  
  // I don't know why you'd do this :)  
  margin: nth($colors, 1); // => text, black  
}
```

Maps may be iterated over with `@each`, of course, but each item will be a somewhat clumsy 2-list. Instead, you can give multiple variables to do an unpacking iteration:

```
@each $key, $value in $colors {  
  // I don't know why you'd do this either!  
  [data-style=$key] {  
    color: $value;  
  }  
}
```

This syntax works on any list-of-lists.

### 3.1.10 Everything is a list

Another change borrowed from Sass trunk: any scalar type (string, number, boolean, etc.) will also act as a list of one element when used where a list is expected. This is most useful when writing Python extensions, but may also save you from checking `type-of` in a complex API.

## 3.2 Compass support

An arbitrary cross-section of Compass 0.11 is supported:

- **Math functions:** `sin`, `cos`, `tan`, `round`, `ceil`, `floor`, `pi`, `e`
- **Images:** `image-url`, `image-width`, `image-height`...
- **Embedded (inline) images:** `inline-image`

**Note:** Currently, Compass support is provided by default, which has led to some surprising behavior since parts of Compass conflict with parts of CSS3. In the future, Compass will become an extension like it is for Ruby, and you will have to opt in.

### 3.2.1 Sprites

Example:

```
$icons: sprite-map("sociable/*.png"); // contains sociable/facebook.png among others.
div {
  background: $icons;
}
@each $icon in sprites($icons) {
  div .#{$icon} {
    width: image-width(sprite-file($icons, $icon));
    height: image-height(sprite-file($icons, $icon));
    background-position: sprite-position($icons, $icon);
  }
}
```

...generates a new sprite file and produces something like:

```
div {
  background: url("/static/assets/u8Y7yEQL0UffAVw5rX7yhw.png?_=1298240989") 0px 0px no-repeat;
}
div .facebook {
  width: 32px;
  height: 32px;
  background-position: 0px 0px;
}
div .twitter {
  width: 32px;
  height: 32px;
  background-position: 0px -32px;
}
...
```

## 3.3 pyScss-specific extensions

pyScss supports some constructs that upstream Sass does not, for various reasons. Listed here are “blessed” features in no danger of being removed, though you should avoid them if you’re at all interested in working with the reference compiler.

There are also some deviations that only exist for backwards compatibility; you should **not** rely on them, they will start spewing warnings at some point in the future, and eventually they will disappear. They are listed separately in *Deprecated features*.

### 3.3.1 @option

Compiler options may be toggled at runtime with `@option`. At the moment the only supported option is `compress`, to control whether the output is compressed:

```
@option compress: true;
```

### 3.3.2 Multiplying strings by numbers

Much like in Python, this works:

```
content: "foo" * 3; // => "foofoofoo"
```

This is a runtime error in the reference compiler.

## 3.4 Deprecated features

### 3.4.1 Brackets to delimit expressions

In an expression, square brackets are equivalent to parentheses:

```
margin-top: [1px + 2px] * 3; // => 9px
```

This is a holdover from xCSS and will be removed in the future.

### 3.4.2 extends

There's an alternative syntax for @extend:

```
a extends b {  
    ...  
}
```

This is identical to:

```
a {  
    @extend b;  
    ...  
}
```

This is a holdover from xCSS and will be removed in the future.

### 3.4.3 self selector

self is an alias for &:

```
a {  
    self:hover {  
        text-decoration: underline;  
    }  
}
```

This is a holdover from xCSS and will be removed in the future.

### 3.4.4 @variables block

Variables may be declared in a dedicated block:

```
@variables {
  $color: red;
}
```

@vars is an alias for @variables.

This is a holdover from xCSS and will be removed in the future.

### 3.4.5 +foo to include a mixin

This:

```
div {
  +border-radius 3px;
}
```

Is equivalent to this:

```
div {
  @include border-radius (3px);
}
```

This is the same as the Sass syntax, but causes some parsing ambiguity, since +foo with a block could be either a nested CSS block with a sibling selector or a mixin call. Its future is uncertain, but you should probably avoid using it in SCSS files.

### 3.4.6 Soft errors

pyScss is much more liberal in what it accepts than the reference compiler; for example, rules at the top level and missing closing braces are accepted without complaint, and attempting to use a non-existent mixin only results in a warning.

pyScss 2.0 is likely to be much stricter; don't rely on any particular abuse of syntax to work in the future.

### 3.4.7 Operations on lists

Binary operations with a list on the left-hand side are performed element-wise:

```
p { margin: (1em 0 3em) * 0.5; // => 0.5em 0 1.5em
}
```

Given that future versions of the reference compiler are likely to introduce built-in list operations, the future of this feature is unclear.

### 3.4.8 Mixin “injection”

A mixin defined like this:

```
@mixin foo(...) { // ...
}
```

will accept **any** keyword arguments, which will be available as variables within the mixin.

This behavior exists for historical reasons and due to the lack of a `**kwargs` equivalent within Sass. Its usage makes mixin behavior harder to understand and you should not use it.

## 3.5 Unsupported Sass features

Some Sass features are not supported or have some gaps. Each of these may be considered a bug.

### 3.5.1 CLI

pyScss's command-line arguments are not entirely compatible with those of the reference compiler.

### 3.5.2 Sass 3.3

The following Sass 3.3 improvements are not yet implemented, but are planned for the near future:

- Use of `&` in expressions.
- `@at-root`
- Source map support.
- Using `...` multiple times in a function call, or passing a map of arguments with `...`. Likewise, `keywords()` is not implemented.
- `unique-id()`, `call()`, and the various `*-exists()` functions are not implemented.

# BACK MATTER

## 4.1 Reporting bugs

If you have any suggestions, bug reports, or minor annoyances, please report them to the issue tracker on GitHub: <http://github.com/Kronuz/pyScss/issues>

## 4.2 Contributing

Please send us pull requests on GitHub! <https://github.com/Kronuz/pyScss>

## 4.3 Running the test suite

The test suite is built atop the excellent `py.test` library, and can be run with:

```
py.test
```

from the root of a source checkout.

Most of the tests are pairs of input/output files in `scss/tests/files`; the test suite scans for these, compiles all the `.scss` files, and compares the output with the `.css` file of the same name. You can limit which file tests run:

```
py.test --test-file-filter=REGEX,REGEX,REGEX...
```

There are also several tests borrowed from the Ruby and C implementations. Many of these don't work (due to missing features, different error messages, slightly different formatting, etc.), so to reduce the useless noise produced by a test run, you must explicitly opt into them with `--include-ruby`, even when using a file filter. These files are in the `from-ruby/` and `from-sassc/` subdirectories.

Additionally, test files in the `xfail/` subdirectory are assumed to fail. Other than these cases, the directory names are arbitrary.

## 4.4 License and copyright

Copyright © 2012 German M. Bravo (Kronuz), with additional heavy contributions by Eevee (Alex Munroe). Licensed under the [MIT license](#).

`pyScss` is inspired by and partially derived from various projects:

- Compass © 2009 Christopher M. Eppstein
- Sass © 2006-2009 Hampton Catlin and Nathan Weizenbaum
- xCSS © 2010 Anton Pawlik

Special thanks to Yelp for allowing Eevee to contribute to pyScss during working hours. Yelp does not claim copyright.

## 4.5 Changelog

### 4.5.1 1.2.0 (Oct 8, 2013)

This is a significant release that greatly increases compatibility with the reference compiler; in particular, the Sass port of Bootstrap now compiles.

There are a lot of changes here, so please feel free to report any bugs you see! The goal is 100% compatibility with the Ruby project.

#### Missing Sass features

- Dashes and underscores are treated as interchangeable in variable, function, and mixin names.
- Rule blocks in the form `background: red { ... }` are now supported.
- Colors are output as their shortest representation, and never as `hsl()`. The separate compiler options for compressing colors have been removed.
- The color modification functions (`adjust-color`, etc.) now work reliably.
- `transparent` is recognized as a color.
- Unrecognized units are now supported and treated as opaque.
- Arbitrary combinations of units (e.g., `px * px`) are supported for intermediate values. Unit cancellation now works reliably.
- Comparison and addition are now more in line with the Ruby behavior.
- `/` is now left untouched when it appears between literals, as in `font: 0 / 0`.
- `null` is supported.
- `zip()` is supported.
- `grayscale()` now knows it's also a CSS3 filter function, and won't be evaluated if its argument is a number.
- Slurpy arguments (`some-function($args...)`) are supported.
- `@extend` has been greatly improved: it eliminates common ancestors and works in many complex cases that used to produce strange results.
- Several Compass functions now adhere more closely to Compass's behavior. `linear-gradient()` is less likely to wreck valid CSS3 syntax.
- Compass's `e()`, `pow()`, `log()`, and `sqrt()` are now supported.

## Bug fixes

- Interactive mode works. Again.
- Color names in strings and selectors are no longer replaced with hex equivalents.
- Unrecognized @-rule blocks such as @keyframes are left alone, rather than being treated like selectors.
- @media blocks aren't repeated for every rule inside.
- Pound-interpolation always drops quotes on strings.
- Single quoted strings no longer lose their quotes when rendered.
- `+ foo { ... }` is now recognized as a nested block, not an include.
- `color-stop()` and several proposed CSS4 functions no longer produce "unrecognized function" warnings.
- Several obscure bugs with variable scoping have been fixed, though a couple others remain.
- Several bugfixes to the C speedups module to bring it in line with the behavior of the pure-Python scanner.

## New features

- Python 3 support. As a result, Python 2.5 no longer works; whether this is a bug or a feature is not yet clear.
- It's possible to write custom Sass functions in Python, though the API for this is not final.
- Experimental support for the map type and destructuring @each, both unreleased additions to the Ruby project.
- Support for the new string and list functions in Sass 3.3.
- Added `background-brushed`.

## Backwards-incompatible changes

- Configuration via monkeypatching the `scss` module no longer works. Monkeypatch `scss.config` instead.
- `em` and `px` are no longer compatible.
- Unrecognized variable names are now a fatal error.

## Internals

- No longer a single 5000-line file!
- Vastly expanded test suite, including some experimental tests borrowed from the Ruby and C implementations.
- Parser now produces an AST rather than evaluating expressions during the parse, which allows for heavier caching and fixes some existing cache bugs.
- The type system has been virtually rewritten; types now act much less like Python types, and compilation uses Sass types throughout rather than mixing Python types with Sass types.

### 4.5.2 1.1.5 (Feb 15, 2013)

- `debug_info` now properly produces rules that can be used by FireSass and Google Chrome SASS Source Maps.
- Improved memory usage for large sets of files to be used as sprites.

- Warns about IE 4095 maximum number of selectors.
- `debug_info` prints info as comments if specified as `comments`.
- Better handling of undefined variables.
- Added CSS filter functions and `skewX skewY`.
- Command line tool and entry point fixed.
- Fix cache buster URLs when paths already include queries or fragments.
- Hashable Values.

### 4.5.3 1.1.4 (Aug 8, 2012)

- Added `--debug-info` command line option (for *FireSass* output).
- Added compass helper function `reject()`.
- Added `undefined` keyword for undefined variables.

### 4.5.4 1.1.3 (Jan 9, 2012)

- Support for the new Sass 3.2.0 features (`@content` and placeholder selectors)
- Fixed bug with line numbers throwing an exception.

### 4.5.5 1.1.2 (Jan 3, 2012)

- Regression bug fixed from 1.1.1

### 4.5.6 1.1.1 (Jan 2, 2012)

- Added optional C speedup module for an amazing boost in scanning speed!
- Added `headings`, `stylesheet-url`, `font-url`, `font-files`, `inline-font-files` and `sprite-names`.

### 4.5.7 1.1.0 (Dec 22, 2011)

- Added `min()` and `max()` for lists.
- Removed exception raise.

### 4.5.8 1.0.9 (Dec 22, 2011)

- Optimizations in the scanner.
- Added `background-noise()` for compass-recipes support.
- `enumerate()` and `range()` can go backwards. Ex.: `range(3, 0)` goes from 3 to 0.
- Added line numbers and files for errors.
- Added support for *Firebug* with *FireSass*.

- `nth(n)` is round (returns the `nth mod len` item of the list).
- `--watch` added to the command line.
- Several bugs fixed.

#### 4.5.9 1.0.8 (May 13, 2011)

- Changed source color (`$src-color`) default to black.
- Moved the module filename to `__init__.py` and module renamed back to `scss`.



# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*