# webcolors Documentation

*Release 1.9.1*

James Bennett

Jun 08, 2019

# Contents

This module provides utility functions for working with the color names and color value formats defined by the HTML and CSS specifications for use in documents on the web.

Support is included for normalizing and converting between the following formats (RGB colorspace only; conversion to/from HSL can be handled by the `colorsys` module in the Python standard library):

- Specification-defined color names
- Six-digit hexadecimal
- Three-digit hexadecimal
- Integer *rgb()* triplet
- Percentage *rgb()* triplet

For example:

```
>>> import webcolors
>>> webcolors.hex_to_name(u'#daa520')
u'goldenrod'
```

Implementations are also provided for the HTML5 color parsing and serialization algorithms. For example, parsing the infamous “chucknorris” string into an rgb() triplet:

```
>>> import webcolors
>>> webcolors.html5_parse_legacy_color(u'chucknorris')
HTML5SimpleColor(red=192, blue=0, green=0)
```

# CHAPTER 1

# Documentation contents

## 1.1 Installation guide

The 1.9.1 version of webcolors is officially tested and supported on the following versions of Python:

- Python 2.7
- Python 3.5
- Python 3.6
- Python 3.7

### 1.1.1 Normal installation

The preferred method of installing webcolors is via *pip*, the standard Python package-installation tool. If you don't have *pip*, instructions are available for how to obtain and install it. If you're using a supported version of Python, *pip* came bundled with your installation of Python.

Once you have *pip*, type:

```
pip install webcolors
```

### 1.1.2 Installing from a source checkout

If you want to work on webcolors, you can obtain a source checkout.

The development repository for webcolors is at <https://github.com/ubernostrum/webcolors>. If you have git installed, you can obtain a copy of the repository by typing:

```
git clone https://github.com/ubernostrum/webcolors.git
```

From there, you can use git commands to check out the specific revision you want, and perform an "editable" install (allowing you to change code as you work on it) by typing:

```
pip install -e .
```

## 1.2 An overview of colors on the web

Colors on the web are typically specified in the sRGB color space, where each color is made up of a red component, a green component and a blue component. This maps to the red, green and blue components of the pixels on a computer display, and to the three sets of cone cells in the human eye, which respond to (roughly) the wavelengths of light associated with red, green and blue.

On the web, sRGB colors are specified in formats which describe the color as a 24-bit integer, where the first 8 bits provide the red value, the second 8 bits the green value and the final 8 bits the blue value. This gives a total space of 256 × 256 × 256 or 16,777,216 unique colors, though due to differences in display technology not all of these colors may be clearly distinguishable on any given physical display.

### 1.2.1 HTML 4

HTML 4 defined two ways to specify sRGB colors:

- The character # followed by three pairs of hexadecimal digits, specifying values for red, green and blue components in that order; for example, *#0099cc*.
- A set of predefined color names which correspond to specific hexadecimal values; for example, *blue*. HTML 4 defines sixteen such colors.

### 1.2.2 CSS1

In its description of color units, CSS1 added three new ways to specify sRGB colors:

- The character # followed by three hexadecimal digits, which is expanded into three hexadecimal pairs by repeating each digit; thus *#09c* is equivalent to *#0099cc*.
- The string *rgb*, followed by parentheses, between which are three base-10 integers in the range 0..255, which are taken to be the values of the red, green and blue components in that order; for example, *rgb(0, 153, 204)*.
- The same as above, except using percentages instead of numeric values; for example, *rgb(0%, 60%, 80%)*.

CSS1 also suggested a set of sixteen color names. These names were identical to the set defined in HTML 4, but CSS1 did not provide definitions of their values and stated that they were taken from “the Windows VGA palette”.

### 1.2.3 CSS2

In its section on colors, CSS2 allowed the same methods of specifying colors as CSS1, and defined and provided values for sixteen named colors, identical to the set found in HTML 4.

CSS2 also specified a list of names of system colors. These had no fixed color values, but would take on values from the operating system or other user interface, and allowed elements to be styled using the same colors as the surrounding user interface. These names are deprecated as of CSS3.

The CSS2.1 revision did not add any new methods of specifying sRGB colors, but did define one additional named color: *orange*.

### 1.2.4 CSS3

The CSS3 color module adds one new way to specify colors:

- A hue-saturation-lightness triplet (HSL), using the construct *hsl()*.

CSS3 also adds support for variable opacity of colors, by allowing the specification of alpha-channel information through the *rgba()* and *hsla()* constructs. These are used similarly to the *rgb()* and *hsl()* constructs, except a fourth value is supplied indicating the level of opacity from *0.0* (completely transparent) to *1.0* (completely opaque). Though not technically a color, the keyword *transparent* is also made available in lieu of a color value, and corresponds to *rgba(0,0,0,0)*.

CSS3 also defines a new set of 147 color names. This set is taken directly from the named colors defined for SVG (Scalable Vector Graphics) markup, and is a superset of the named colors defined in CSS2.1.

### 1.2.5 HTML5

HTML5 exists in two forms: a living document maintained by WHATWG, and a W3C Recommendation. The two HTML5 documents, as of this writing, share a common definition of color values and parsing, and formalize the parsing and serialization of colors according to prior standards and real-world implementations in web browsers.

HTML5 does not introduce any new methods of specifying colors, but does simplify the description of colors and introduce useful terminology.

- A set of three 8-bit numbers representing the red, blue and green components of an sRGB color is termed a "simple color".
- A seven-character string which begins with the character #, followed by six ASCII hex digits (i.e., *A-Fa-f0-9*), representing the red, green and blue components of an sRGB color, is a "valid simple color".
- A valid simple color expressed with only lowercase ASCII hex digits (i.e., *a-f0-9*) is a "valid lowercase simple color".

HTML5 provides three algorithms related to colors:

1. An algorithm for parsing simple color values, which works on any string that is a valid simple color as defined above.
2. An algorithm for serializing simple color values, which will always produce a valid lowercase simple color.
3. A legacy color-parsing algorithm, which will yield a simple color from a variety of inputs, including inputs which are valid simple colors, inputs which are valid for formats from other standards, and certain types of "junk" inputs which were common in real-world documents.

The HTML5 legacy parsing algorithm does not support the non-color keyword *transparent* from CSS3 and will produce an error for that input. It also does not recognize the CSS2 "system color" keywords; it will convert each such keyword to a simple color, consistently, but in a way which does not follow CSS2's definitions of these keywords (which itself was system- and configuration-dependent).

The implementations in this module are based on the definitions and algorithms of the W3C HTML5 Recommendation's section on colors.

### 1.2.6 What this module supports

The webcolors module supports the following methods of specifying sRGB colors, and conversions between them:

- Six-digit hexadecimal
- Three-digit hexadecimal

- Integer *rgb()* triplet
- Percentage *rgb()* triplet
- The defined named colors of HTML 4, CSS2, CSS2.1, and CSS3

The webcolors module **does not support**:

- The CSS1 named colors, which did not have defined values
- The CSS2 system colors, which did not have fixed values
- The *transparent* keyword, which denotes an effective lack of color
- Opacity/alpha-channel information specified via the *rgba()* construct
- Colors specified in the HSL color space, via *hsl()* or *hsla()* constructs

If you need to convert between sRGB-specified colors and HSL-specified colors, or colors specified via other means, consult the `colorsys` module in the Python standard library, which can perform conversions amongst several common color systems.

## 1.3 Normalization and conventions

Since the various formats used to specify colors in web documents do not always map cleanly to Python data types, and some variation is permitted in how to use each format in a web document, webcolors applies a set of conventions for representing color names and values, and for normalizing them.

### 1.3.1 Python string types

The webcolors module is written to be compatible with both Python 2 and Python 3, which have different approaches to strings:

- On Python 2, a sequence of bytes in a particular encoding (a "byte string") is represented by the type `str` , and Unicode strings are represented by the type `unicode`. Mixing of `str` and `unicode` is possible in Python 2, but not recommended as it is a frequent source of bugs.
- On Python 3, a sequence of bytes in a particular encoding is represented by the type `bytes`, and Unicode strings are represented by the type `str`. Mixing of `str` and `bytes` is not permitted in Python 3, and will usually raise exceptions.

The approach to string types in webcolors is as follows:

- On Python 3, use of Unicode strings – `str` – is mandatory for all string arguments to functions in webcolors. Use of `bytes` values is forbidden and will result in errors.
- All mappings from color names to hexadecimal values (and vice versa) are dictionaries whose keys and values are Unicode strings (`str` on Python 3 and `unicode` on Python 2). This permits use of byte strings on Python 2, but ensures that results will be Unicode strings.
- All functions whose return values include strings will use Unicode strings (`unicode` on Python 2 and `str` on Python 3).
- All functions whose arguments include string values, *except* for the HTML5 color algorithms (see below), will accept a sequence of bytes (`str`) on Python 2, but will convert to Unicode strings (`unicode`) for output.

Because the HTML5 Recommendation specifies its color algorithms only in terms of Unicode strings (and in some cases, requires exact identification of Unicode code points to determine behavior), the following constraint applies to the functions implementing these algorithms:

- Any string arguments *must* be Unicode strings (`unicode` on Python 2 or `str` on Python 3). Use of `str` on Python 2 or `bytes` on Python 3 will raise a `ValueError`.

Use of Unicode strings is strongly preferred on Python 2. To encourage this, all documentation for webcolors uses the *u* prefix for string literals. Use of the *u* prefix is required on Python 2 to mark a string literal as Unicode; on Python 3.3 and later, use of this prefix is permitted but not necessary (as all unprefixed string literals on Python 3 are Unicode strings).

When relevant, this documentation will refer to the Unicode string type as `six.text_type`; the `six` library provides helpers for writing code compatible with both Python 2 and 3, and `six.text_type` always resolves to the Unicode string type of the Python version in use.

### 1.3.2 Hexadecimal color values

For colors specified via hexadecimal values, webcolors will accept strings in the following formats:

- The character # followed by three hexadecimal digits, where digits A-F may be upper- or lowercase.
- The character # followed by six hexadecimal digits, where digits A-F may be upper- or lowercase (i.e., what HTML5 designates a "valid simple color" when all digits are uppercase, and a "valid lowercase simple color" when all digits are lowercase).

For output which consists of a color specified via hexadecimal values, and for functions which perform intermediate conversion to hexadecimal before returning a result in another format, webcolors always normalizes such values to a string in the following format:

- The character # followed by six hexadecimal digits, with digits A-F forced to lowercase (what HTML5 designates a "valid lowercase simple color").

The function *`normalize_hex()`* can be used to perform this normalization manually if desired.

### 1.3.3 Integer and percentage *rgb()* triplets

Functions which work with integer *rgb()* triplets accept them as a 3-`tuple` of Python `int`. Functions which work with percentage *rgb()* triplets accept them as 3-`tuple` of Python strings (see *above regarding Python string types*).

Plain tuples are accepted by all functions which deal with integer or percentage *rgb()* triplets, but three types of `namedtuple()` are also provided to represent these values: *`IntegerRGB`* for integer triplets, *`PercentRGB`* for percentage triplets, and *`HTML5SimpleColor`* for an HTML5 simple color. Functions which return an integer or percentage *rgb()* triplet, or an HTML5 simple color, will return values of these types.

Internally, Python `float` is used in some conversions to and from the triplet representations; for each function which may have the precision of its results affected by this, a note is provided in the documentation.

For colors specified via *rgb()* triplets, values contained in the triplets will be normalized in accordance with CSS clipping rules:

- Integer values less than 0 will be normalized to 0, and percentage values less than 0% will be normalized to 0%.
- Integer values greater than 255 will be normalized to 255, and percentage values greater than 100% will be normalized to 100%.
- The "negative zero" values -0 and -0% will be normalized to 0 and 0%, respectively.

The functions *`normalize_integer_triplet()`* and *`normalize_percent_triplet()`* can be used to perform this normalization manually if desired.

### 1.3.4 Color names

For colors specified via predefined names, webcolors will accept strings containing names case-insensitively, so long as they contain no spaces or non-alphabetic characters. Thus, for example, *u'AliceBlue'* and *u'aliceblue'* are both accepted, and both will refer to the same color: *rgb(240, 248, 255)*.

For output which consists of a color name, and for functions which perform intermediate conversion to a predefined name before returning a result in another format, webcolors always normalizes such values to be entirely lowercase.

#### Identifying sets of named colors

For purposes of identifying the specification from which to draw the selection of defined color names, webcolors uses strings naming the specifications, and provides *a set of constants containing the correct values*.

Note that the CSS1 named colors are not supported here, as CSS1 merely "suggested" a set of color names, and declined to provide values for them. The CSS2 "system colors" are also not represented here, as they had no fixed defined values and are now deprecated.

## 1.4 Module contents

The contents of the webcolors module fall into five categories:

1. A set of (optional) data types for representing color values.
2. Constants for several purposes.
3. Normalization functions which sanitize input in various formats prior to conversion or output.
4. Conversion functions between each method of specifying colors.
5. Implementations of the color parsing and serialization algorithms in HTML5.

See *the documentation regarding conventions* for information regarding the types and representation of various color formats in webcolors.

All conversion functions which involve color names take an optional argument to determine the specification from which to draw color names. See *the set of specification identifiers* for valid values.

All conversion functions, when faced with identifiably invalid hexadecimal color values, or with a request to name a color which has no name in the requested specification, or with an invalid specification identifier, will raise `ValueError`.

In the documentation below, `six.text_type` means the Unicode string type of the Python version being used; on Python 3 this is `str` and on Python 2 it is `unicode`. See *the documentation on use of Python string types* for details.

### 1.4.1 Data types

Integer and percentage *rgb()* triplets, and HTML5 simple colors, can be passed to functions in webcolors as plain 3-`tuple` of the appropriate data type. But the following `namedtuple()` instances are also provided to represent these types more richly, and functions in webcolors which return triplets or simple colors will return instances of these:

**class** `webcolors.`**`IntegerRGB`**

A `namedtuple()` representing an integer RGB triplet. Has three fields, each of type `int` and in the range 0-255 inclusive:

**red**
The red portion of the color value.

**green**
The red portion of the color value.

**blue**
The red portion of the color value.

**class** webcolors.**PercentRGB**
A `namedtuple()` representing a percentage RGB triplet. Has three fields, each of type `six.text_type` and representing a percentage value in the range 0%-100% inclusive:

**red**
The red portion of the color value.

**green**
The red portion of the color value.

**blue**
The red portion of the color value.

**class** webcolors.**HTML5SimpleColor**
A `namedtuple()` representing an HTML5 simple color. Has three fields, each of type `int` and in the range 0-255 inclusive:

**red**
The red portion of the color value.

**green**
The red portion of the color value.

**blue**
The red portion of the color value.

## 1.4.2 Constants

Several sets of constants are provided in webcolors, for use when converting or identifying colors or specifications.

### Specification identifiers

The following constants are available for indicating the specification from which to draw color name choices, in functions which can work with multiple specifications.

webcolors.**CSS2**
Represents the CSS2 specification. Value is *u'css2'*.

webcolors.**CSS21**
Represents the CSS2.1 specification. Value is *u'css21'*.

webcolors.**CSS3**
Represents the CSS3 specification. Value is *u'css3'*.

webcolors.**HTML4**
Represents the HTML 4 specification. Value is *u'html4'*.

### Color mappings

The following constants are available for direct use in mapping from color names to values, although it is strongly recommended to use one of the normalizing conversion functions instead.

#### Mappings from names to hexadecimal values

`webcolors.`**`HTML4_NAMES_TO_HEX`**

A `dict` whose keys are the normalized names of the sixteen named HTML 4 colors, and whose values are the normalized hexadecimal values of those colors.

`webcolors.`**`CSS2_NAMES_TO_HEX`**

An alias for *`HTML4_NAMES_TO_HEX`*, as CSS2 defined the same set of colors.

`webcolors.`**`CSS21_NAMES_TO_HEX`**

A `dict` whose keys are the normalized names of the seventeen named CSS2.1 colors, and whose values are the normalized hexadecimal values of those colors (sixteen of these are identical to HTML 4 and CSS2; the seventeenth color is *orange*, added in CSS2.1).

`webcolors.`**`CSS3_NAMES_TO_HEX`**

A `dict` whose keys are the normalized names of the 147 named CSS3 colors, and whose values are the normalized hexadecimal values of those colors. These colors are also identical to the 147 named colors of SVG.

#### Mappings from hexadecimal values to names

`webcolors.`**`HTML4_HEX_TO_NAMES`**

A `dict` whose keys are the normalized hexadecimal values of the sixteen named HTML 4 colors, and whose values are the corresponding normalized names.

`webcolors.`**`CSS2_HEX_TO_NAMES`**

An alias for *`HTML4_HEX_TO_NAMES`*, as CSS2 defined the same set of colors.

`webcolors.`**`CSS21_HEX_TO_NAMES`**

A `dict` whose keys are the normalized hexadecimal values of the seventeen named CSS2.1 colors, and whose values are the corresponding normalized names (sixteen of these are identical to HTML 4 and CSS2; the seventeenth color is *orange*, added in CSS2.1).

`webcolors.`**`CSS3_HEX_TO_NAMES`**

A `dict` whose keys are the normalized hexadecimal values of the 147 names CSS3 colors, and whose values are the corresponding normalized names. These colors are also identical to the 147 named colors of SVG.

**Note: Spelling variants**

CSS3 provides two names – *gray* and *grey* – which each map to the hexadecimal value *#808080*. Reversing from the hexadecimal value to a name requires picking one and only one of these, and webcolors chooses *gray* as it was the spelling used by HTML 4, CSS1, and CSS2.

## 1.4.3 Normalization functions

`webcolors.`**`normalize_hex`**`(`*`hex_value`*`)`

Normalize a hexadecimal color value to a string consisting of the character # followed by six lowercase hexadecimal digits (what HTML5 terms a “valid lowercase simple color”).

If the supplied value cannot be interpreted as a hexadecimal color value, `ValueError` is raised. See *the conventions used by this module* for information on acceptable formats for hexadecimal values.

Examples:

```
>>> normalize_hex(u'#0099cc')
'#0099cc'
>>> normalize_hex(u'#0099CC')
'#0099cc'
>>> normalize_hex(u'#09c')
'#0099cc'
>>> normalize_hex(u'#09C')
'#0099cc'
>>> normalize_hex(u'#0099gg')
Traceback (most recent call last):
    ...
ValueError: '#0099gg' is not a valid hexadecimal color value.
>>> normalize_hex(u'0099cc')
Traceback (most recent call last):
    ...
ValueError: '0099cc' is not a valid hexadecimal color value.
```

**Parameters** **`hex_value`** (*`str`*) – The hexadecimal color value to normalize.

**Return type** `six.text_type`

**Raises** **`ValueError`** – when the input is not a valid hexadecimal color value.

`webcolors.`**`normalize_integer_triplet`**(*rgb_triplet*)

Normalize an integer *rgb()* triplet so that all values are within the range 0..255.

Examples:

```
>>> normalize_integer_triplet((128, 128, 128))
IntegerRGB(red=128, green=128, blue=128)
>>> normalize_integer_triplet((0, 0, 0))
IntegerRGB(red=0, green=0, blue=0)
>>> normalize_integer_triplet((255, 255, 255))
IntegerRGB(red=255, green=255, blue=255)
>>> normalize_integer_triplet((270, -20, -0))
IntegerRGB(red=255, green=0, blue=0)
```

**Parameters** **`rgb_triplet`** (*`tuple`*) – The integer *rgb()* triplet to normalize.

**Return type** *IntegerRGB*

`webcolors.`**`normalize_percent_triplet`**(*rgb_triplet*)

Normalize a percentage *rgb()* triplet so that all values are within the range 0%..100%.

Examples:

```
>>> normalize_percent_triplet((u'50%', u'50%', u'50%'))
PercentRGB(red=u'50%', green=u'50%', blue=u'50%')
>>> normalize_percent_triplet((u'0%', u'100%', u'0%'))
PercentRGB(red=u'0%', green=u'100%', blue=u'0%')
>>> normalize_percent_triplet((u'-10%', u'-0%', u'500%'))
PercentRGB(red=u'0%', green=u'0%', blue=u'100%')
```

**Parameters** **`rgb_triplet`** (*`tuple`*) – The percentage *rgb()* triplet to normalize.

**Return type** *PercentRGB*

## 1.4.4 Conversions from color names to other formats

`webcolors.`**`name_to_hex`**`(`*name*, *spec=CSS3*`)`

Convert a color name to a normalized hexadecimal color value.

The color name will be normalized to lower-case before being looked up.

Examples:

```
>>> name_to_hex(u'white')
u'#ffffff'
>>> name_to_hex(u'navy')
u'#000080'
>>> name_to_hex(u'goldenrod')
u'#daa520'
>>> name_to_hex(u'goldenrod', spec=HTML4)
Traceback (most recent call last):
    ...
ValueError: 'goldenrod' is not defined as a named color in html4.
```

**Parameters**

- **`name`** (*`str`*) – The color name to convert.
- **`spec`** (*`str`*) – The specification from which to draw the list of color names. Default is *`CSS3`*.

**Return type** `six.text_type`

**Raises** **`ValueError`** – when the given name has no definition in the given spec.

`webcolors.`**`name_to_rgb`**`(`*name*, *spec=CSS3*`)`

Convert a color name to a 3-`tuple` of `int` suitable for use in an *rgb()* triplet specifying that color.

The color name will be normalized to lower-case before being looked up.

Examples:

```
>>> name_to_rgb(u'white')
IntegerRGB(red=255, green=255, blue=255)
>>> name_to_rgb(u'navy')
IntegerRGB(red=0, green=0, blue=128)
>>> name_to_rgb(u'goldenrod')
IntegerRGB(red=218, green=165, blue=32)
```

**Parameters**

- **`name`** (*`str`*) – The color name to convert.
- **`spec`** (*`str`*) – The specification from which to draw the list of color names. Default is `CSS3`.

**Return type** *IntegerRGB*

**Raises** **`ValueError`** – when the given name has no definition in the given spec.

`webcolors.`**`name_to_rgb_percent`**`(`*name*, *spec=CSS3*`)`

Convert a color name to a 3-`tuple` of percentages suitable for use in an *rgb()* triplet specifying that color.

The color name will be normalized to lower-case before being looked up.

Examples:

```
>>> name_to_rgb_percent(u'white')
PercentRGB(red=u'100%', green=u'100%', blue=u'100%')
>>> name_to_rgb_percent(u'navy')
PercentRGB(red=u'0%', green=u'0%', blue=u'50%')
>>> name_to_rgb_percent(u'goldenrod')
PercentRGB(red=u'85.49%', green=u'64.71%', blue=u'12.5%')
```

**Parameters**

- **`name`** (`str`) – The color name to convert.
- **`spec`** (`str`) – The specification from which to draw the list of color names. Default is `CSS3`.

**Return type** *PercentRGB*

**Raises** **`ValueError`** – when the given name has no definition in the given spec.

## Conversion from hexadecimal color values to other formats

`webcolors.`**`hex_to_name`**`(`*hex_value*, *spec=CSS3*`)`

Convert a hexadecimal color value to its corresponding normalized color name, if any such name exists.

The hexadecimal value will be normalized before being looked up.

**Note: Spelling variants**

One hexadecimal value – *#808080* – can map to either of two names in CSS3, because it supports both *gray* and *grey* as color names. This function will return *u'gray'* for the input *u'#808080'*. See also the note on `CSS3_HEX_TO_NAMES`.

Examples:

```
>>> hex_to_name(u'#ffffff')
u'white'
>>> hex_to_name(u'#fff')
u'white'
>>> hex_to_name(u'#000080')
u'navy'
>>> hex_to_name(u'#daa520')
u'goldenrod'
>>> hex_to_name(u'#daa520', spec=HTML4)
Traceback (most recent call last):
    ...
ValueError: '#daa520' has no defined color name in html4.
```

**Parameters**

- **`hex_value`** (`str`) – The hexadecimal color value to convert.

- **spec** (*str*) – The specification from which to draw the list of color names. Default is *CSS3*.

**Return type** six.text_type

**Raises** **ValueError** – when the given color has no name in the given spec, or when the supplied hex value is invalid.

webcolors.**hex_to_rgb**(*hex_value*)

Convert a hexadecimal color value to a 3-tuple of int suitable for use in an *rgb()* triplet specifying that color.

The hexadecimal value will be normalized before being converted.

Examples:

```
>>> hex_to_rgb(u'#fff')
IntegerRGB(red=255, green=255, blue=255)
>>> hex_to_rgb(u'#000080')
IntegerRGB(red=0, green=0, blue=128)
```

**Parameters** **hex_value** (*str*) – The hexadecimal color value to convert.

**Return type** *IntegerRGB*

**Raises** **ValueError** – when the supplied hex value is invalid.

webcolors.**hex_to_rgb_percent**(*hex_value*)

Convert a hexadecimal color value to a 3-tuple of percentages suitable for use in an *rgb()* triplet representing that color.

The hexadecimal value will be normalized before being converted.

Examples:

```
>>> hex_to_rgb_percent(u'#ffffff')
PercentRGB(red=u'100%', green=u'100%', blue=u'100%')
>>> hex_to_rgb_percent(u'#000080')
PercentRGB(red=u'0%', green=u'0%', blue=u'50%')
```

**Parameters** **hex_value** (*str*) – The hexadecimal color value to convert.

**Return type** *PercentRGB*

**Raises** **ValueError** – when the supplied hex value is invalid.

### 1.4.5 Conversions from integer *rgb()* triplets to other formats

webcolors.**rgb_to_name**(*rgb_triplet*, *spec=CSS3*)

Convert a 3-tuple of int, suitable for use in an *rgb()* color triplet, to its corresponding normalized color name, if any such name exists.

To determine the name, the triplet will be converted to a normalized hexadecimal value.

**Note: Spelling variants**

One *rgb()* value – *(128, 128, 128)* – can map to either of two names in CSS3, because it supports both *gray* and *grey* as color names. This function will return *u'gray'* for the input *(128, 128, 128)*. See also the note on *CSS3_HEX_TO_NAMES*.

Examples:

```
>>> rgb_to_name((255, 255, 255))
u'white'
>>> rgb_to_name((0, 0, 128))
u'navy'
```

**Parameters**

- **`rgb_triplet`** (*`typing.Union[IntegerRGB, Tuple[int, int, int]]`*) – The *rgb()* triplet
- **`spec`** (*`str`*) – The specification from which to draw the list of color names. Default is *`CSS3`*.

**Return type** `six.text_type`

**Raises** **`ValueError`** – when the given color has no name in the given spec.

`webcolors.`**`rgb_to_hex`**(*rgb_triplet*)

Convert a 3-`tuple` of `int`, suitable for use in an *rgb()* color triplet, to a normalized hexadecimal value for that color.

Examples:

```
>>> rgb_to_hex((255, 255, 255))
u'#ffffff'
>>> rgb_to_hex((0, 0, 128))
u'#000080'
```

**Parameters** **`rgb_triplet`** (*`typing.Union[IntegerRGB, Tuple[int, int, int]]`*) – The *rgb()* triplet.

**Return type** `six.text_type`

`webcolors.`**`rgb_to_rgb_percent`**(*rgb_triplet*)

Convert a 3-`tuple` of `int`, suitable for use in an *rgb()* color triplet, to a 3-`tuple` of percentages suitable for use in representing that color.

**Note: Floating-point precision**

This function makes some trade-offs in terms of the accuracy of the final representation; for some common integer values, special-case logic is used to ensure a precise result (e.g., integer 128 will always convert to *u'50%'*, integer 32 will always convert to *u'12.5%'*), but for all other values a standard Python `float` is used and rounded to two decimal places, which may result in a loss of precision for some values due to the inherent imprecision of IEEE floating-point numbers.

Examples:

```
>>> rgb_to_rgb_percent((255, 255, 255))
PercentRGB(red=u'100%', green=u'100%', blue=u'100%')
>>> rgb_to_rgb_percent((0, 0, 128))
PercentRGB(red=u'0%', green=u'0%', blue=u'50%')
>>> rgb_to_rgb_percent((218, 165, 32))
PercentRGB(red=u'85.49%', green=u'64.71%', blue=u'12.5%')
```

**Parameters** **rgb_triplet** (*typing.Union[IntegerRGB, Tuple[int, int, int]]*) – The *rgb()* triplet.

**Return type** *PercentRGB*

## 1.4.6 Conversions from percentage *rgb()* triplets to other formats

webcolors.**rgb_percent_to_name**(*rgb_percent_triplet*, *spec=CSS3*)

Convert a 3-tuple of percentages, suitable for use in an *rgb()* color triplet, to its corresponding normalized color name, if any such name exists.

To determine the name, the triplet will be converted to a normalized hexadecimal value.

**Note: Spelling variants**

One value – *(50%, 50%, 50%)* – can map to either of two names in CSS3, because it supports both *gray* and *grey* as color names. This function will return *u'gray'* for the input *(u'50%', u'50%', u'50%')*. See also the note on *CSS3_HEX_TO_NAMES*.

Examples:

```
>>> rgb_percent_to_name((u'100%', u'100%', u'100%'))
u'white'
>>> rgb_percent_to_name((u'0%', u'0%', u'50%'))
u'navy'
>>> rgb_percent_to_name((u'85.49%', u'64.71%', u'12.5%'))
u'goldenrod'
```

**Parameters**

- **rgb_percent_triplet** (*typing.Union[PercentRGB, Tuple[str, str, str]]*) – The *rgb()* triplet.
- **spec** (*str*) – The specification from which to draw the list of color names. Default is *CSS3*.

**Return type** six.text_type

**Raises** **ValueError** – when the given color has no name in the given spec.

webcolors.**rgb_percent_to_hex**(*rgb_percent_triplet*)

Convert a 3-tuple of percentages, suitable for use in an *rgb()* color triplet, to a normalized hexadecimal color value for that color.

Examples:

```
>>> rgb_percent_to_hex((u'100%', u'100%', u'0%'))
u'#ffff00'
>>> rgb_percent_to_hex((u'0%', u'0%', u'50%'))
u'#000080'
>>> rgb_percent_to_hex((u'85.49%', u'64.71%', u'12.5%'))
u'#daa520'
```

**Parameters** **rgb_percent_triplet** (*typing.Union[PercentRGB, Tuple[str, str, str]]*) – The *rgb()* triplet.

**Return type** *str*

webcolors.**rgb_percent_to_rgb**(*rgb_percent_triplet*)

Convert a 3-`tuple` of percentages, suitable for use in an *rgb()* color triplet, to a 3-`tuple` of `int` suitable for use in representing that color.

Some precision may be lost in this conversion. See the note regarding precision for *`rgb_to_rgb_percent()`* for details.

Examples:

```
>>> rgb_percent_to_rgb((u'100%', u'100%', u'100%'))
IntegerRGB(red=255, green=255, blue=255)
>>> rgb_percent_to_rgb((u'0%', u'0%', u'50%'))
IntegerRGB(red=0, green=0, blue=128)
>>> rgb_percent_to_rgb((u'85.49%', u'64.71%', u'12.5%'))
IntegerRGB(red=218, green=165, blue=32)
```

**Parameters** **`rgb_percent_triplet`** (*`typing.Union[PercentRGB, Tuple[str, str, str]]`*) – The *rgb()* triplet.

**Return type** *IntegerRGB*

## 1.4.7 HTML5 color algorithms

**Warning:** There are two versions of the HTML5 standard. Although they have common origins and are extremely similar, one is a living document (maintained by WHATWG) and the other is a W3C Recommendation. The functions documented below implement the HTML5 color algorithms as given in section 2.4.6 of the W3C HTML5 Recommendation.

webcolors.**html5_parse_simple_color**(*input*)

Apply the HTML5 simple color parsing algorithm.

Note that *input* **must** be a Unicode string – on Python 2, byte strings will not be accepted.

Examples:

```
>>> html5_parse_simple_color(u'#ffffff')
HTML5SimpleColor(red=255, green=255, blue=255)
>>> html5_parse_simple_color(u'#fff')
Traceback (most recent call last):
    ...
ValueError: An HTML5 simple color must be a string exactly seven characters long.
```

**Parameters** **`input`** (`six.text_type`, which must consist of exactly the character '#' followed by six hexadecimal digits) – The color to parse.

**Return type** *HTML5SimpleColor*

**Raises** **`ValueError`** – when the given input is not a Unicode string of length 7, consisting of exactly the character # followed by six hexadecimal digits.

webcolors.**html5_serialize_simple_color**(*simple_color*)

Apply the HTML5 simple color serialization algorithm.

Examples:

```
>>> html5_serialize_simple_color((0, 0, 0))
u'#000000'
>>> html5_serialize_simple_color((255, 255, 255))
u'#ffffff'
```

**Parameters** **simple_color** (*typing.Union[IntegerRGB, HTML5SimpleColor, Tuple[int, int, int]], all values in the range 0..255 inclusive*) – The color to serialize.

**Return type** A valid lowercase simple color, which is a Unicode string exactly seven characters long, beginning with # and followed by six lowercase hexadecimal digits.

webcolors.**html5_parse_legacy_color**(*input*)

Apply the HTML5 legacy color parsing algorithm.

Note that, since this algorithm is intended to handle many types of malformed color values present in real-world Web documents, it is *extremely* forgiving of input, but the results of parsing inputs with high levels of "junk" (i.e., text other than a color value) may be surprising.

Note also that *input* **must** be a Unicode string – on Python 2, byte strings will not be accepted.

Examples:

```
>>> html5_parse_legacy_color(u'black')
HTML5SimpleColor(red=0, green=0, blue=0)
>>> html5_parse_legacy_color(u'chucknorris')
HTML5SimpleColor(red=192, green=0, blue=0)
>>> html5_parse_legacy_color(u'Window')
HTML5SimpleColor(red=0, green=13, blue=0)
```

**Parameters** **input** (six.text_type) – The color to parse.

**Return type** *HTML5SimpleColor*

**Raises** **ValueError** – when the given input is not a Unicode string, or when it is precisely the string *u'transparent'*.

## 1.5 Conformance and testing

Much of the behavior of webcolors is dictated by the relevant web standards, which define the acceptable color formats, how to determine valid values for each format and the values corresponding to defined color names. Maintaining correct conversions and conformance to those standards is crucial.

The source distribution of webcolors (the *.tar.gz* file you can download from the Python Package Index) includes a *tests/* directory containing a normal test suite as well as supplemental test files which perform more comprehensive verification.

### 1.5.1 The normal test suite

The normal test suite for webcolors aims for 100% code coverage, but does *not* aim for 100% coverage of possible color value inputs and outputs. Instead, it uses a small number of test values to routinely exercise various functions.

The test values used in most test functions are chosen to provide, where applicable, at least one of each of the following types of values:

- An endpoint of the acceptable range of values (i.e., *#ffffff* and/or *#000000* for hexadecimal).
- A value beyond the high end of the acceptable range (i.e., greater than 255 in an integer triplet, or greater than 100% for a percentage triplet).
- A value beyond the low end of the acceptable range (i.e., less than 0 in an integer triplet, or less than 0% for a percentage triplet).
- A "negative zero" value (-0 in an integer triplet, or -0% in a percentage triplet).
- An arbitrary value not from an endpoint of the acceptable range (usually *#000080*, chosen because the author likes navy blue).
- A value which corresponds to a named color in CSS3/SVG but not in earlier standards (usually *#daa520*, which is *goldenrod* in CSS3/SVG).

Since this covers the cases most likely to produce problems, this test suite provides good basic confidence in the correctness of the tested functions. It runs on every commit to the repository, and on every release tag. You can see the results of test runs online at Travis CI.

However, the normal test suite cannot guarantee that the color definitions included in webcolors correspond to those in the relevant standards, and cannot provide guarantees of correct conversions for all possible values. For that, additional tests are required.

### 1.5.2 Full verification tests

These tests are contained in two files which are not executed during normal test runs: *tests/definitions.py* and *tests/full_colors.py*. They are not run as part of the normal test suite, but are run prior to each release of webcolors.

#### Verifying color definitions

The *definitions.py* test file verifies that the color definitions in webcolors are correct. It does this by downloading the relevant standards documents as HTML, parsing out the color definitions in them, and comparing them to the definitions in webcolors. That consists of:

- Parsing out the names and hexadecimal values of the 16 named colors in the HTML 4 standard, and checking that the names and values in `HTML4_NAMES_TO_HEX` match.
- Parsing out the names and hexadecimal values of the 17 named colors in the CSS2.1 standard, and checking that the names and values in `CSS21_NAMES_TO_HEX` match.
- Parsing out the names and hexadecimal and integer values of the 147 named colors in the CSS3 color module (although the color set is taken from SVG, CSS3 provides both hexadecimal and integer values for them, while the SVG standard provides only integer values), and checking that the names and values in `CSS3_NAMES_TO_HEX` match, and that `name_to_rgb()` returns the correct integer values.

#### Fully verifying correctness of conversions

The *full_colors.py* test file exercises `hex_to_rgb()`, `rgb_to_hex()`, `rgb_to_rgb_percent()` and `rgb_percent_to_rgb()` as fully as is practical.

For conversions between hexadecimal and integer *rgb()*, it generates all 16,777,216 possible color values for each format in order (starting at *#000000* and *(0, 0, 0)* and incrementing), and verifies that each one converts to the corresponding value in the other format. Thus, it is possible to be confident that webcolors provides correct conversions between all possible color values in those formats.

Testing the correctness of conversion to and from percentage *rgb()*, however, is more difficult, and a full test is not provided, for two reasons:

1. Because percentage *rgb()* values can make use of floating-point values, and because standard floating-point types in most common programming languages (Python included) are inherently imprecise, exact verification is not possible.
2. The only rigorous definition of the format of a percentage value is in CSS2, which declares a percentage to be "a <*number*> immediately followed by '%'". The CSS2 definition of a number places no limit on the length past the decimal point, and appears to be declaring any real number as a valid value, though percentage triplets clip their inputs to the range 0.0-100.0. As the subset of reals in the range 0.0 to 100.0 is uncountably infinite, testing all legal values is not possible on current hardware in any reasonable amount of time.

Since precise correctness and completeness are not achievable, webcolors instead aims to achieve *consistency* in conversions. Specifically, the *full_colors.py* test generates all 16,777,216 integer *rgb()* triplets, and for each such triplet *t* verifies that the following assertion holds:

```
t == rgb_percent_to_rgb(rgb_to_rgb_percent(t))
```

### 1.5.3 Running the tests

The standard test runner for webcolors is tox, which supports testing against multiple Python versions and executing a variety of different test tasks. The source distribution of webcolors includes its *tox.ini* file. To run the tests, install tox (*pip install tox*), then download and unpack a source distribution of webcolors from the Python Package Index.

To run the normal test suite against the complete set of supported Python versions:

```
$ tox
```

This requires that you have each supported version of Python (for webcolors 1.9.1, this is 2.7, 3.5, 3.6, and 3.7) available. To test only against a specific version of Python, use the *-e* flag and pass the version to test. For example, to test on Python 3.7:

```
$ tox -e py37
```

To run the full verification tests for definition correctness and conversions, specify the "release" test environment instead (so named because these tests are usually run only prior to a new release of webcolors):

```
$ tox -e release
```

Note that this requires an internet connection, and is CPU-intensive.

## 1.6 Change log

This document lists the changes between each release of webcolors.

### 1.6.1 Version 1.9.1, released 2019-06-07

#### Bugs fixed

- The *__version__* attribute of the installed webcolors module, although not documented or referenced anywhere, was accidentally not updated in the 1.9 release. It has now been updated (and now indicates 1.9.1).

### 1.6.2 Version 1.9, released 2019-06-01

No bug fixes.

#### New features

- Added *a set of constants to use when referring to specifications that define color names*.

#### Other changes

- When asked to provide a color name, using the CSS3/SVG set of names, for the hexadecimal value *#808080*, the integer triplet *rgb(128, 128, 128)*, or the percentage triplet *rgb(50%, 50%, 50%)*, webcolors now always returns *u'gray'*, never *u'grey'*. Previously, the behavior could be inconsistent as it depended on the Python version in use; *u'gray'* was picked because it was the spelling variant used in HTML 4, CSS1, and CSS2.

### 1.6.3 Version 1.8.1, released 2018-02-12

The 1.8.1 release is a repackaging of 1.8 to produce both source (.tar.gz) and binary (.whl) package formats, following reports that the source-package-only release of 1.8 was causing installation issues for some users. See issue 6 in the repository for details.

### 1.6.4 Version 1.8, released 2018-02-08

No bug fixes.

#### New features

- Added the `IntegerRGB`, `PercentRGB`, and `HTML5SimpleColor` named tuples.

#### Other changes

- Drop support for Python 3.3 (Python core team no longer maintains 3.3).
- Mark support for Python 3.6.
- *The full verification tests* now run correctly on Python 3.

### 1.6.5 Version 1.7, released 2016-11-25

No new features or bugfixes.

#### Other changes

- Drop support for Python 2.6 (Python core team no longer maintains 2.6).
- Mark support for Python 3.4.
- On Python 3, the use of `str` for all functions which take string arguments is now mandatory. Attempted use of `bytes` will raise an exception. On Python 2, use of bytestrings is still permitted.

### 1.6.6 Version 1.5.1, released 2015-11-23

No new features.

#### Bug fixes

- Corrected multiple typos in documentation.

### 1.6.7 Version 1.5, released 2015-03-07

No bug fixes.

#### New features

- Python 3 support: webcolors now supports Python 3.3.
- Added *HTML5 color algorithms*.

#### Other changes

- Packaging improvements.

### 1.6.8 Version 1.4, released 2012-02-10

No new features.

#### Bugs fixed

- Integer and percentage *rgb()* triplets now normalized in accordance with CSS clipping rules.

#### Other changes

- Packaging fixes.
- Preparatory work for Python 3 support.

### 1.6.9 Version 1.3.1, released 2009-10-24

No new features or bugfixes.

#### Other changes

- Documentation expanded.
- Documentation now maintained using Sphinx.

### 1.6.10 Version 1.3, released 2009-05-08

No new features or bugfixes.

#### Other changes

- Documentation expanded.

### 1.6.11 Version 1.2, 2009-03-01

#### Bugs fixed:

- Corrected the download URL in the *setup.py* script.

### 1.6.12 Version 1.1, released 2008-12-19

No new features or bugfixes.

#### Other changes

- Documentation expanded.

### 1.6.13 Version 1.0, released 2008-10-28

Initial stable release of webcolors.

## 1.7 Frequently asked questions

The following notes answer common questions, and may be useful to you when using webcolors.

### 1.7.1 What versions of Python are supported?

On Python 2, webcolors supports and is tested on Python 2.7. On Python 3, webcolors supports and is tested on Python 3.5, 3.6, and 3.7. These are the only versions of Python currently receiving upstream security support from the Python core team.

### 1.7.2 How closely does this module follow the standards?

As closely as is practical (see below regarding floating-point values), within *the supported formats*; the webcolors module was written with the relevant standards documents close at hand. See *the conformance documentation* for details.

### 1.7.3 Why aren't *rgb_to_rgb_percent()* and *rgb_percent_to_rgb()* precise?

This is due to limitations in the representation of floating-point numbers in programming languages. Python, like many programming languages, uses IEEE floating-point, which is inherently imprecise for some values. This imprecision only appears when converting between integer and percentage *rgb()* triplets, as in `rgb_to_rgb_percent()` and `rgb_percent_to_rgb()`.

To work around this, some common values (255, 128, 64, 32, 16 and 0) are handled as special cases, with hard-coded precise results. For all other values, conversion to percentage *rgb()* triplet uses a standard Python `float`, rounding the result to two decimal places.

See *the conformance documentation* for details on how this affects testing.

### 1.7.4 Why does webcolors prefer American spellings?

In CSS3, two names – *gray* and *grey* – each map to the same color: *rgb(128, 128, 128)*. Using any of the conversions from names to other formats (`name_to_hex()`, `name_to_rgb()`, or `name_to_rgb_percent()`) will accept either name provided the *spec* argument is `CSS3`.

However, converting from other formats to a name requires picking one of these spellings. Since webcolors uses a Python `dict` to store its *name-to-value mappings*, simply reversing those mappings risks inconsistency: swapping the keys and values of a `dict` in Python depends on the key order, which varies from one version of Python to another and in several supported Python versions is not guaranteed to be consistent and/or is documented as an implementation detail not to be relied on. So webcolors must manually pick a spelling to normalize to, and chooses *gray*. This choice was made for consistency with HTML 4, CSS1, and CSS2, each of which only allowed *gray*.

As a result, the following functions each return *u'gray'* for the following inputs, and never *u'grey'*, even when the *spec* argument is `CSS3`:

- `hex_to_name()` for input *u'#808080'*
- `rgb_to_name()` for input *(128, 128, 128)*
- `rgb_percent_to_name()` for input *(u'50%', u'50%', u'50%')*

### 1.7.5 Why aren't HSL values supported?

In the author's experience, actual use of HSL values on the Web is extremely rare; the overwhelming majority of all colors used on the Web are specified using sRGB, through hexadecimal color values or through integer or percentage *rgb()* triplets. This decreases the importance of supporting the *hsl()* construct.

Additionally, Python already has the `colorsys` module in the standard library, which offers functions for converting between RGB, HSL, HSV and YIQ color systems. If you need conversion to/from HSL or another color system, use `colorsys`.

### 1.7.6 Why not use a more object-oriented design with classes for the colors?

Representing color values with Python classes would introduce overhead for no real gain. Real-world use cases tend to involve working directly with the actual values, so settling on conventions for how to represent them as Python types, and then offering a function-based interface, accomplishes everything needed without the additional indirection layer of having to instantiate and serialize a color-wrapping object.

Keeping a function-based interface also maintains consistency with Python's built-in `colorsys` module which has the same style of interface for converting amongst color spaces.

Note that if an object-oriented interface is desired, the third-party colormath module does have a class-based interface (and rightly so, as it offers a wider range of color representation and manipulation options than webcolors).

### 1.7.7 How am I allowed to use this module?

The webcolors module is distributed under a three-clause BSD license. This is an open-source license which grants you broad freedom to use, redistribute, modify and distribute modified versions of webcolors. For details, see the file *LICENSE* in the source distribution of webcolors.

### 1.7.8 I found a bug or want to make an improvement!

The canonical development repository for webcolors is online at <https://github.com/ubernostrum/webcolors>. Issues and pull requests can both be filed there.

**See also:**

- The sRGB color space
- HTML 4: Colors
- CSS 1: Color units
- CSS 2: Colors
- CSS 3 color module
- HTML5: Colors

# Python Module Index

# Index