TinyColor

JavaScript color tooling

TinyColor is a small, fast library for color manipulation and conversion in JavaScript. It allows many forms of input, while providing color conversions and other color utility functions. It has no dependencies.

Including in a browser

Include the file `tinycolor.js` in the page in a `script` tag:

```
<script type='text/javascript' src='tinycolor.js'></script>
<script type='text/javascript'>
var color = tinycolor("red");
</script>
```

Usage

Call `tinycolor(input)` or `new tinycolor(input)`, and you will have an object with the following properties. See Accepted String Input and Accepted Object Input below for more information about what is accepted.

Accepted String Input

The string parsing is very permissive. It is meant to make typing a color as input as easy as possible. All commas, percentages, parenthesis are optional, and most input allow either 0-1, 0%-100%, or 0-n (where n is either 100, 255, or 360 depending on the value).

HSL and HSV both require either 0%-100% or 0-1 for the `S`/`L`/`V` properties. The `H` (hue) can have values between 0%-100% or 0-360.

RGB input requires either 0-255 or 0%-100%.

If you call `tinycolor.fromRatio`, RGB and Hue input can also accept 0-1.

Here are some examples of string input:

Hex, 8-digit (RGBA) Hex

```
tinycolor("#000");
tinycolor("000");
tinycolor("#369C");
tinycolor("369C");
tinycolor("#f0f0f6");
tinycolor("f0f0f6");
tinycolor("#f0f0f688");
tinycolor("f0f0f688");
```

RGB, RGBA

```
tinycolor("rgb (255, 0, 0)");
tinycolor("rgb 255 0 0");
tinycolor("rgba (255, 0, 0, .5)");
tinycolor({ r: 255, g: 0, b: 0 });
tinycolor.fromRatio({ r: 1, g: 0, b: 0 });
tinycolor.fromRatio({ r: .5, g: .5, b: .5 });
```

HSL, HSLA

```
tinycolor("hsl(0, 100%, 50%)");
tinycolor("hsla(0, 100%, 50%, .5)");
tinycolor("hsl(0, 100%, 50%)");
tinycolor("hsl 0 1.0 0.5");
tinycolor({ h: 0, s: 1, l: .5 });
tinycolor.fromRatio({ h: 1, s: 0, l: 0 });
tinycolor.fromRatio({ h: .5, s: .5, l: .5 });
```

HSV, HSVA

```
tinycolor("hsv(0, 100%, 100%)");
tinycolor("hsva(0, 100%, 100%, .5)");
tinycolor("hsv (0 100% 100%)");
tinycolor("hsv 0 1 1");
tinycolor({ h: 0, s: 100, v: 100 });
tinycolor.fromRatio({ h: 1, s: 0, v: 0 });
tinycolor.fromRatio({ h: .5, s: .5, v: .5 });
```

Named

```
tinycolor("RED");
tinycolor("blanchedalmond");
tinycolor("darkblue");
```

Accepted Object Input

If you are calling this from code, you may want to use object input. Here are some examples of the different types of accepted object inputs:

```
{ r: 255, g: 0, b: 0 }
{ r: 255, g: 0, b: 0, a: .5 }
{ h: 0, s: 100, l: 50 }
{ h: 0, s: 100, v: 100 }
```

Methods

getFormat

Returns the format used to create the tinycolor instance

```
var color = tinycolor("red");
color.getFormat(); // "name"
color = tinycolor({r:255, g:255, b:255});
color.getFormat(); // "rgb"
```

getOriginalInput

Returns the input passed into the constructer used to create the tinycolor instance

```
var color = tinycolor("red");
color.getOriginalInput(); // "red"
color = tinycolor({r:255, g:255, b:255});
color.getOriginalInput(); // "{r: 255, g: 255, b: 255}"
```

isValid

Return a boolean indicating whether the color was successfully parsed. Note: if the color is not valid then it will act like `black` when being used with other methods.

```
var color1 = tinycolor("red");
color1.isValid(); // true
color1.toHexString(); // "#ff0000"

var color2 = tinycolor("not a color");
color2.isValid(); // false
color2.toString(); // "#000000"
```

getBrightness

Returns the perceived brightness of a color, from `0-255`, as defined by Web Content Accessibility Guidelines (Version 1.0).

```
var color1 = tinycolor("#fff");
color1.getBrightness(); // 255

var color2 = tinycolor("#000");
color2.getBrightness(); // 0
```

isLight

Return a boolean indicating whether the color's perceived brightness is light.

```
var color1 = tinycolor("#fff");
color1.isLight(); // true

var color2 = tinycolor("#000");
color2.isLight(); // false
```

isDark

Return a boolean indicating whether the color's perceived brightness is dark.

```
var color1 = tinycolor("#fff");
color1.isDark(); // false

var color2 = tinycolor("#000");
color2.isDark(); // true
```

getLuminance

Returns the perceived luminance of a color, from `0-1` as defined by Web Content Accessibility Guidelines (Version 2.0).

```
var color1 = tinycolor("#fff");
color1.getLuminance(); // 1

var color2 = tinycolor("#000");
color2.getLuminance(); // 0
```

getAlpha

Returns the alpha value of a color, from `0-1`.

```
var color1 = tinycolor("rgba(255, 0, 0, .5)");
color1.getAlpha(); // 0.5

var color2 = tinycolor("rgb(255, 0, 0)");
color2.getAlpha(); // 1

var color3 = tinycolor("transparent");
color3.getAlpha(); // 0
```

setAlpha

Sets the alpha value on a current color. Accepted range is in between `0-1`.

```
var color = tinycolor("red");
color.getAlpha(); // 1
color.setAlpha(.5);
color.getAlpha(); // .5
color.toRgbString(); // "rgba(255, 0, 0, .5)"
```

String Representations

The following methods will return a property for the `alpha` value, which can be ignored: `toHsv`, `toHsl`, `toRgb`

toHsv

```
var color = tinycolor("red");
color.toHsv(); // { h: 0, s: 1, v: 1, a: 1 }
```

toHsvString

```
var color = tinycolor("red");
color.toHsvString(); // "hsv(0, 100%, 100%)"
color.setAlpha(0.5);
color.toHsvString(); // "hsva(0, 100%, 100%, 0.5)"
```

toHsl

```
var color = tinycolor("red");
color.toHsl(); // { h: 0, s: 1, l: 0.5, a: 1 }
```

toHslString

```
var color = tinycolor("red");
color.toHslString(); // "hsl(0, 100%, 50%)"
color.setAlpha(0.5);
color.toHslString(); // "hsla(0, 100%, 50%, 0.5)"
```

toHex

```
var color = tinycolor("red");
color.toHex(); // "ff0000"
```

toHexString

```
var color = tinycolor("red");
color.toHexString(); // "#ff0000"
```

toHex8

```
var color = tinycolor("red");
color.toHex8(); // "ff0000ff"
```

toHex8String

```
var color = tinycolor("red");
color.toHex8String(); // "#ff0000ff"
```

toRgb

```
var color = tinycolor("red");
color.toRgb(); // { r: 255, g: 0, b: 0, a: 1 }
```

toRgbString

```
var color = tinycolor("red");
color.toRgbString(); // "rgb(255, 0, 0)"
color.setAlpha(0.5);
color.toRgbString(); // "rgba(255, 0, 0, 0.5)"
```

toPercentageRgb

```
var color = tinycolor("red");
color.toPercentageRgb() // { r: "100%", g: "0%", b: "0%", a: 1 }
```

toPercentageRgbString

```
var color = tinycolor("red");
color.toPercentageRgbString(); // "rgb(100%, 0%, 0%)"
color.setAlpha(0.5);
color.toPercentageRgbString(); // "rgba(100%, 0%, 0%, 0.5)"
```

toName

```
var color = tinycolor("red");
color.toName(); // "red"
```

toFilter

```
var color = tinycolor("red");
color.toFilter(); //
"progid:DXImageTransform.Microsoft.gradient(startColorstr=#ffff0000,endColorstr=
#ffff0000)"
```

toString

Print to a string, depending on the input format. You can also override this by passing one of `"rgb", "prgb", "hex6", "hex3", "hex8", "name", "hsl", "hsv"` into the function.

```
var color1 = tinycolor("red");
color1.toString(); // "red"
color1.toString("hsv"); // "hsv(0, 100%, 100%)"

var color2 = tinycolor("rgb(255, 0, 0)");
color2.toString(); // "rgb(255, 0, 0)"
color2.setAlpha(.5);
color2.toString(); // "rgba(255, 0, 0, 0.5)"
```

Color Modification

These methods manipulate the current color, and return it for chaining. For instance:

```
tinycolor("red").lighten().desaturate().toHexString() // "#f53d3d"
```

lighten

`lighten: function(amount = 10) -> TinyColor`. Lighten the color a given amount, from 0 to 100. Providing 100 will always return white.

```
tinycolor("#f00").lighten().toString(); // "#ff3333"
tinycolor("#f00").lighten(100).toString(); // "#ffffff"
```

brighten

`brighten: function(amount = 10) -> TinyColor`. Brighten the color a given amount, from 0 to 100.

```
tinycolor("#f00").brighten().toString(); // "#ff1919"
```

darken

`darken: function(amount = 10) -> TinyColor`. Darken the color a given amount, from 0 to 100. Providing 100 will always return black.

```
tinycolor("#f00").darken().toString(); // "#cc0000"
tinycolor("#f00").darken(100).toString(); // "#000000"
```

desaturate

`desaturate: function(amount = 10) -> TinyColor`. Desaturate the color a given amount, from 0 to 100. Providing 100 will is the same as calling `greyscale`.

```
tinycolor("#f00").desaturate().toString(); // "#f20d0d"
tinycolor("#f00").desaturate(100).toString(); // "#808080"
```

saturate

`saturate: function(amount = 10) -> TinyColor`. Saturate the color a given amount, from 0 to 100.

```
tinycolor("hsl(0, 10%, 50%)").saturate().toString(); // "hsl(0, 20%, 50%)"
```

greyscale

`greyscale: function() -> TinyColor`. Completely desaturates a color into greyscale. Same as calling `desaturate(100)`.

```
tinycolor("#f00").greyscale().toString(); // "#808080"
```

spin

`spin: function(amount = 0) -> TinyColor`. Spin the hue a given amount, from -360 to 360. Calling with 0, 360, or -360 will do nothing (since it sets the hue back to what it was before).

```
tinycolor("#f00").spin(180).toString(); // "#00ffff"
tinycolor("#f00").spin(-90).toString(); // "#7f00ff"
tinycolor("#f00").spin(90).toString(); // "#80ff00"

// spin(0) and spin(360) do nothing
tinycolor("#f00").spin(0).toString(); // "#ff0000"
tinycolor("#f00").spin(360).toString(); // "#ff0000"
```

Color Combinations

Combination functions return an array of TinyColor objects unless otherwise noted.

analogous

`analogous: function(, results = 6, slices = 30) -> array<TinyColor>`.

```
var colors = tinycolor("#f00").analogous();

colors.map(function(t) { return t.toHexString(); }); // [ "#ff0000", "#ff0066",
"#ff0033", "#ff0000", "#ff3300", "#ff6600" ]
```

monochromatic

`monochromatic: function(, results = 6) -> array<TinyColor>`.

```
var colors = tinycolor("#f00").monochromatic();

colors.map(function(t) { return t.toHexString(); }); // [ "#ff0000", "#2a0000",
"#550000", "#800000", "#aa0000", "#d40000" ]
```

splitcomplement

`splitcomplement: function() -> array<TinyColor>`.

```
var colors = tinycolor("#f00").splitcomplement();

colors.map(function(t) { return t.toHexString(); }); // [ "#ff0000", "#ccff00",
"#0066ff" ]
```

triad

`triad: function() -> array<TinyColor>`.

```
var colors = tinycolor("#f00").triad();

colors.map(function(t) { return t.toHexString(); }); // [ "#ff0000", "#00ff00",
"#0000ff" ]
```

tetrad

`tetrad: function() -> array<TinyColor>`.

```
var colors = tinycolor("#f00").tetrad();

colors.map(function(t) { return t.toHexString(); }); // [ "#ff0000", "#80ff00",
"#00ffff", "#7f00ff" ]
```

complement

`complement: function() -> TinyColor`.

```
tinycolor("#f00").complement().toHexString(); // "#00ffff"
```

Color Utilities

```
tinycolor.equals(color1, color2)
tinycolor.mix(color1, color2, amount = 50)
```

random

Returns a random color.

```
var color = tinycolor.random();
color.toRgb(); // "{r: 145, g: 40, b: 198, a: 1}"
```

Readability

TinyColor assesses readability based on the Web Content Accessibility Guidelines (Version 2.0).

readability

`readability: function(TinyColor, TinyColor) -> Object`. Returns the contrast ratio between two colors.

```
tinycolor.readability("#000", "#000"); // 1
tinycolor.readability("#000", "#111"); // 1.1121078324840545
tinycolor.readability("#000", "#fff"); // 21
```

Use the values in your own calculations, or use one of the convenience functions below.

isReadable

`isReadable: function(TinyColor, TinyColor, Object) -> Boolean`. Ensure that foreground and background color combinations meet WCAG guidelines. `Object` is optional, defaulting to `{level: "AA",size: "small"}`. `level` can be `"AA"` or "AAA" and `size` can be `"small"` or `"large"`.

Here are links to read more about the AA and AAA requirements.

```
tinycolor.isReadable("#000", "#111", {}); // false
tinycolor.isReadable("#ff0088", "#5c1a72",{level:"AA",size:"small"}); //false
tinycolor.isReadable("#ff0088", "#5c1a72",{level:"AA",size:"large"}), //true
```

mostReadable

`mostReadable: function(TinyColor, [TinyColor, Tinycolor ...], Object) -> Boolean`. Given a base color and a list of possible foreground or background colors for that base, returns the most readable color. If none of the colors in the list is readable, `mostReadable` will return the better of black or white if `includeFallbackColors:true`.

```
tinycolor.mostReadable("#000", ["#f00", "#0f0", "#00f"]).toHexString(); //
"#00ff00"
tinycolor.mostReadable("#123", ["#124", "#125"],
{includeFallbackColors:false}).toHexString(); // "#112255"
tinycolor.mostReadable("#123", ["#124", "#125"],
{includeFallbackColors:true}).toHexString();  // "#ffffff"
tinycolor.mostReadable("#ff0088", ["#2e0c3a"],
{includeFallbackColors:true,level:"AAA",size:"large"}).toHexString()   //
"#2e0c3a",
tinycolor.mostReadable("#ff0088", ["#2e0c3a"],
{includeFallbackColors:true,level:"AAA",size:"small"}).toHexString()   //
"#000000",
```

See index.html in the project for a demo.

Common operations

clone

`clone: function() -> TinyColor`. Instantiate a new TinyColor object with the same color. Any changes to the new one won't affect the old one.

```
var color1 = tinycolor("#F00");
var color2 = color1.clone();
color2.setAlpha(.5);

color1.toString(); // "#ff0000"
color2.toString(); // "rgba(255, 0, 0, 0.5)"
```