CSS Data Types

Classifying Data Used as CSS Values

R. Scott Granneman & Jans Carton

Notes & URLs for this presentation can be found…

- » underneath the link to this slide show on granneman.com
- » at files.granneman.com/presentations/webdev/CSS-Data-Types.txt

W3C Working Draft

CSS Values and Units Module Level 4

W3C Working Draft, 18 December 2023

▼ More details about this document

This version:

https://www.w3.org/TR/2023/WD-css-values-4-20231218/

Latest published version:

https://www.w3.org/TR/css-values-4/

Editor's Draft:

https://drafts.csswg.org/css-values-4/

History:

https://www.w3.org/standards/history/css-values-4/

Feedback:

CSSWG Issues Repository

Inline In Spec

Editors:

Tab Atkins (Google)

Elika J. Etemad / fantasai (Apple)

Suggest an Edit for this Spec:

GitHub Editor

Test Suite:

https://wpt.fyi/results/css/

For more on `<data types>` & other units...

Defined

What's a *data type*?

A way to classify various kinds of data that are allowed as values for CSS properties

In other words, you group the following values into a specific data type

- `<length>`: `10px`, `1em`, `1.5rem`, `12pt`, & `1.2ch`
- `<color>`: `red`, `#FF0000`, `rgb(255 0 0)`, & `hsl(360deg 100% 50%)`
- `<number>`: `7`, `7.7`, `+7.7`, `-7.7`, `0`, `0.77`

<alpha-value> • <angle-percentage> • <angle> • <calc-constant> • <calc-product> • <calc-sum> • <calc-value> • <color-stop-list> • <color> • <custom-ident> • <dashed-ident> • <dimension> • <ending-shape> • <flex> • <frequency-percentage> • <frequency> • <hue> • <image> • <integer> • <length-percentage> • <length> • <linear-color-hint> • <linear-color-stop> • <number> • <percentage> • <position> • <ratio> • <resolution> • <size> • <string> • <time-percentage> • <time> • <url>

Let’s focus on these basic data types

Text: `<string>` & `<url>`

Distance: `<length>`

Images: `<color>`

Basic Data Types

<string>

<url>

<length>

<color>

<string>

`<string>`

Represents a quoted string of Unicode characters

Quotation marks can be either `"Cthulhu"` or `'Cthulhu'`

`<string>` examples

```
"Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl
fhtagn!"

'Phony Stark is a 💩 social media company
owner'

"2023-04-04 15:22:35 CDT"
```

<url>

`<url>`

Represents a *pointer to a web resource* (e.g., image or font)

Was `<url>`*, then `<uri>`**, but now back to `<url>`*** (to stay, hopefully!)

* CSS 1
** CSS 2.1
*** CSS Values and Units Module 3

`<url>` expressed in 2 ways:

» absolute or relative URL
» data URL

Absolute or Relative

Absolute	`http://baz.com/foo.png`	Complete URL
Site root-relative	`/foo.htm` `/bar/foo.png`	Web server fills in protocol & domain
Document-relative	`foo.png`	Same directory
	`bar/foo.png`	Down into sub-directory
	`../bar/foo.png`	Up into parent directory

Data URL

Data URL

Resource inlining: instead of referencing an external image (via an absolute or relative URL), you instead embed the images directly into the style sheet

```
background-image: url(
R0lGODlhEAAQAMQAAORHHOVSKu…)
```

Why? To save HTTP requests to the web server

The downside: it makes your style sheets bigger


```

nhmeqaJfzxyvjciPPz8/358fbWqezJtuymLv79/frmpNVsEsNcKfTQW/PMqOmVHfTNbO2kSky9LS0u2tNvXh1Nt4Fu7u7vbYgvHx
8frorPG6PfG7Me6wLf301fTNUtirlvj4+NeHVvnmquWJGeugJOB8F+qSMvPFX/G9ReykIu+5Zf766uqaHvrkneqdI+uZNuzs7PG7
dPnhnfX19erq6v312NXV1fLBTeiLG+ymPvXVtdhzFNl2JeSYVe2rNMrKyvrrvdJmEvfajOugIOOGGfrqu8JQEuyiP+uhKPvt3fbT
df323e+1Nfvs1fniwe2lJOubIsRSD+mTHPnevvnoqfjdj/fp4vT09Pvp1efn5+Tk5PTJTuLi4vz38vbUeeWGItnZ2fvo0Pngvs5g
Efjhl+bm5u2uL+Dg4Nvb2/vsuueGGOiQHfzy0f316eGDGPzuweylLcxeEfbUgeucJMNWHOqbIPvsv/bTY//9+vfXcumXH+iYHvjd
jP317P305fvz7u2rMuynL/rps+yeQfjfjeeVHe6pMu+yMe6tJ/ror/njpffadPCyL+mYJOqZIdZvE+2nI+2pJPbUe+yiIeucH+qY
Hfjbh+6sJPbYgfvqsfv7++ukK/TNa+mQG+60POGtkuuiIdR5RPzwx8pcD83NzfTLj+yjJu6vJe2vTvbPnt2hgvjej/TKlvLFfv3y
5/334u6xOPjdlvLYyfTNbvrpsfvqsvjdwvzyzvPd0fC4N+vp5/fUdOmZKuykLOKhd/HSu/bavfngoPXRePbRftuRYOCNRvHOtvbR
c/fZe+Szm/TMWPG0OPfXeeiLKspiJdF0O+qdMOJ/G9LBueSCHvTHXfLCWOnn5fHFnvrw6868tPbPcfjn2vrn08dcHPbQcN+IPumR
IOqRHuqSH/TIYu/AleqgNeOogOu9n8hWD++3OPG/eu+0M/nkmOaKKe6pM////yH5BAEAAP8ALAAAAAAwADAAAAj/AP8JHEiwoEGB
jhwdXMiwocN/vXo9nEjRYBcnTrpU3DhRxwIcOjiKXOjFDh1GkLyMXCkQEa41RVocwYWB5cgKdjJQoMDPTgWbHCkk0bZ1x44WLpII
AFqx1qUUR4vsyMCiFtOJjsAdKrIlQoQtLTjU63PVYRAuqX5FkCUrwi8yl/SVZdgFC4AifAzpNcSnxSFmGucaNJCMWJE8eQgQQFwk
RTIEggvKsbWJz59UmFMVK/bnGABbciJjiFPhigpiEVKoXq06tYorFdDUXCkgRK4gIOAN+rICW4RUGYILD55K1qYVXyDAAxEkV4il
DfuM4YVgEjwwXzgtcAGA0oHFPcKL/x8PhcABSgBcLOD0BQy8SQh4eWk0cIyBH/Kw79ueqMyBTFBAUQUBKfTwwAN0JEjHgQj2kAkB
AVaRwnmJqHdJe/H8gMAYAoCQwxoAMNKDDIpJMqEQA+yySxkstujii2UE00MGKUiimAwPMALAGjkYsBQFk0TCTQo4DsCIBKYkaYoE
SCq5JJNJQgnlk0wyMsADRC4TCQgUDNTID5a4oUkwJ5SJCiplmnkmmmm2meaZbp4QjCZaWDIJfQT5YoMTWkQjhSiAiuKGG4EGSiig
gyJaqKCBShHNNZbY4MtBWXTjxDXBCHLNppx2ek0AmwYgqqekChKMO/7YkAVD7LzhhDpSSP/zway01srKrKzkemuttUojxQf+9IOG
QyGMY8k8ggyTwLLMNuvss8wOMw05kYwTwkTQhAPLLVzkowIOR5Rzw7jkl1tuOUfgoAIXl9wyRTjGVKREDbCcYu+9p2DCwr4sbMMF
vyxggu+9U9SgBEcdzABLKAw37PDDEDcMCxUdjEQPFaBkrPHGHHesMRXnsNREJTmUbPLJKKecQyVN2PTOIqPELPPMNNc8igfV2MRE
IVP07PPPQAc9BQ9MsNSGBqQkrfTSTDetdBSIrESCBjz4wMPVWGetddaFWK3BOitZ80khNPBQytlon2122mmTzcMnz6ykwBKeYKGI
B3jnrYgPWGD/McgieeuNhSef3LOSMyWAsYoYjDNe9yqrDOID5IM3LoYHq4BRgj0rxVADFoC0IronzURywSBi+OGHGINcEEkznYje
ygVY1EDNSt/4cUEze0AAiROWQLBHCXPEEsscJfT+e/B7XHCBH7Gs1M4MI1QPjBMQILHELDEII5AwMcyyBBIQOAFM9SNQ4c1Ij4SB
hCuujNDJDBpY4QAJBpHggBWfzNDJCPBDghWgsxFk4AMeIyiAH6JghQaYoSFmaIAVouCHAoxgD/jAH0csEAg91CAKhNDFIyiCDl0Q
Igo10MMsLCCSV8whEBt4RRtE0oZXbGAOc8hGQzCgChjAYAhnmAAKjRywgRcowwRIRMEEhjAEHzqRiUJEIhCIQEVzvGADDUCBEpsI
A1XUpIdCBEIa1MAAPIjDCCJIowiMgIc7qIEISIwjEdLABgbAQQS0eIIen0ALEaQDDgxgQx2AoMQu8hAGYSSCGthwBwY4Eg+OZEMa
4KjECVjSBGJcpB2NwEkjwAGQbkwDEEwwAUNGZiQBAQA7
```

`base64`

Encodes binary file into text

Convert any image format: JPEG, GIF, PNG, SVG

On macOS & Linux

```
$ base64 star.gif
```

On Windows

```
Base64.exe -e star.gif
```

(see support.microsoft.com/kb/191239)

address, find IP address of a domain name

Convert IP adddress to different formats

Convert Unicode characters to HTML code numbers and vice versa

Convert Unicode characters to Unicode escape sequences and vice versa

Coordinate converter and show map

Create self-signed SSL certificates online

CSV to XML converter

CVS pserver password decoder and encoder

Decode Certificate Signing Request (CSR)

Decode SSL certificate

Electronic business card vCard generator

European clothing standard EN 13402 pictogram generator

Favicon generator

Find the BIC numbers for Dutch IBAN numbers

Free game sound effects

Free game textures

Free online practice exams

Free online SEPA XML

Input base64 encoder and decoder:

Enter source data *:

-- Or --

Upload source file *:
Max 100 KB.

Choose File no file selected

Conversion method *: Encode to Base64 string

Select output *: Output in textbox

Max characters per line *: 75 Allowed values 0 - 999. Use 0 for unlimited characters.

To prevent automated submissions an Access Code has been implemented for this tool.

jH8

Please enter the Access Code as displayed above*:

* = required

Convert Clear

Output base64 encoder and decoder:

Select all Clear

Used in many CSS properties, such as `background-image`, `cursor`, & `list-style`

<length>

`<length>`

Represents *distance measurements*: a `<number>` immediately followed by a unit

Units

» Absolute

» Relative (viewport-percentage or font-relative)

`<length>` is used with properties like...

- `width`
- `height`
- `margin`
- `padding`
- `border-width`
- `font-size`
- `text-shadow`

Note: `<percentage>` is not a `<length>`, even those some of these properties also accept it

Absolute `<length>`

px

cm

mm

Q

in

pc

pt

Of these, the foundation is `px`, because every `<length>` is eventually calculated as pixels

Pixel means
picture element

Pixels are not
just for computers!

Georges Seurat's *A Sunday Afternoon on the Island of La Grande Jatte* (1884)

Resolution is a measurement of the amount of effort it takes to see (or *resolve*) individual pixels

The Roman mosaic is low resolution

The Seurat painting is higher resolution

An iPhone is very high resolution

The basic unit of an electronic display is a *pixel*

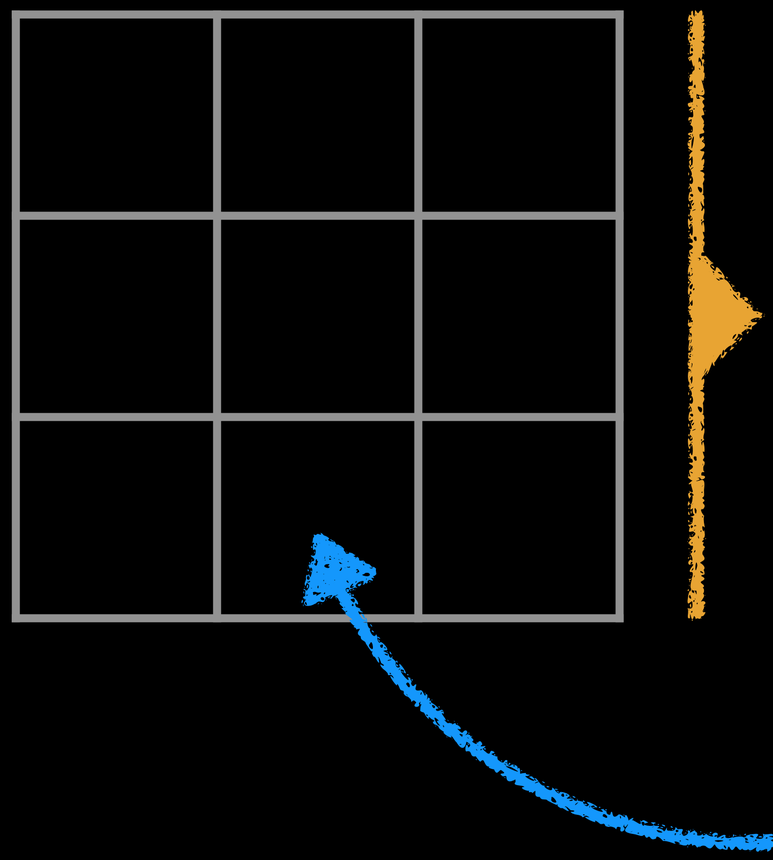
All length measurements eventually get counted in pixels

For years, computer pixels were all `1/72` of an inch

Apple's iPhone 4 introduced the first high resolution devices that have much smaller pixels (300+ per inch — & my 2019 iPhone 11 Pro Max has 458 ppi)

The size of pixels in modern high resolution devices varies hugely

Thanks to these 2 changes, the size of a pixel in CSS had to be redefined



1 CSS pixel can approximate multiple device pixels

For good old “standard resolution” displays, a pixel is (still a device pixel, which is) `1/72` inch

For high resolution displays (& other high resolution media like print) a CSS pixel is now about `1/96` inch

So now it may take several device pixels to equal 1 CSS pixel

`px`	96 pixels per inch
`cm`	1 centimeter = 96px/2.54
`mm`	1 millimeter = 1/10 of 1cm
`Q`	1 quarter-millimeter = 1/40 of 1cm
`in`	1 inch = 2.54 cm = 96px
`pc`	1 pica = 1/6 of 1in = 12 pt
`pt`	1 point = 1/72 of 1in

Relative `<length>`

Relative lengths are measurements derived from some other distance

Depending on the unit, this can be based on…

» size of a specific character
» line height
» viewport size

Units are either *viewport-percentage* or *font-relative*

Font-Relative <length>

ch

ex

em & rem

These values are relative to the size of the font

Of these, the most common you will see are `em` & `rem`

We will cover those & more in *CSS Typography*

`em`

Originally width of capital M

Now the *calculated font-size* of an element, in other words, the total height of a font from descender to ascender, *inherited from the parent element*

1 `em` is the height of a font, so if your font is set to be 16px tall, then `1em = 16px`

HTML

```
<ul class="parent-relative">
  <li>Foo</li>
  <li>Bar
    <ul>
      <li>Baz
        <ul>
          <li>Qaz</li>
          <li>Tub</li>
        </ul>
      </li> <!-- /Baz -->
      <li>Alf</li>
    </ul>
  </li> <!-- /Bar -->
  <li>Pru</li>
</ul>
```

CSS

```
.parent-relative li {
  font-size: .8em;
}
```

JS

em units are parent-relative

- Foo
- Bar
  - Baz
    - Qaz
    - Tub
  - Alf
- Pru

`em` sizing is compounded!

`rem`

Always based on the `font-size` of the *root element* (`<html>`), which defaults to `16px` in all Web browsers

Great for creating perfectly scalable layouts

HTML

```
<ul class="root-relative">
  <li>Foo</li>
  <li>Bar
    <ul>
      <li>Baz
        <ul>
          <li>Qaz</li>
          <li>Tub</li>
        </ul>
      </li> <!-- /Baz -->
      <li>Alf</li>
    </ul>
  </li> <!-- /Bar -->
  <li>Pru</li>
</ul>
```

CSS

```
.root-relative li {
  font-size: .8rem;
}
```

JS

em units are parent-relative

- Foo
- Bar
  - Baz
    - Qaz
    - Tub
  - Alf
- Pru

rem units are root-relative

- Foo
- Bar
  - Baz
    - Qaz
    - Tub
  - Alf
- Pru

`rem` sizing is consistent

					iOS		
`ch`	9*	12	2	7	7	27	4.4
`ex`	6	12	2	3.1	3.2	4	2.1
`em`	Y	Y	Y	Y	Y	Y	Y
`rem`	9† 11	12	3.6	5	4‡	4	2.1

* Width of `ch` is that of the `0` glyph, not its surrounding space
† Doesn't work with font or pseudo elements
‡ iOS Safari 5.0-5.1 doesn't support rem with media queries

Viewport Percentage <length>

`vh`	1% of the viewport's height
`vw`	1% of the viewport's width
`vmin`	1% of `vh` or `vw`—whichever is smaller
`vmax`	1% of `vh` or `vw`—whichever is larger

`vi`	1% of the initial containing block, in the direction of the root element's *inline* axis
`vb`	1% of the initial containing block, in the direction of the root element's *block* axis

	IE	Edge	Firefox	Safari	iOS	Chrome	Android
vh	10	12	19	6.1	8	26	4.4
vw	10	12	19	6.1	6.1	26	4.4
vmin	10	12	19	6.1	6.1	26	4.4
vmax	–	–	19	6.1	8	26	4.4
vi	–	–	–	–	–	–	–
vb	–	–	–	–	–	–	–

`calc()`

`calc()`

CSS function that lets you perform calculations when specifying values for CSS properties

Can be used anywhere you could use a `<length>`, `<frequency>`, `<angle>`, `<time>`, `<percentage>`, `<number>`, or `<integer>`

Most commonly used with `<length>`

Can use `+`, `−`, `×`, & `/` operators

- `×` & `/` require at least 1 `<number>`
- `/` requires that `<number>` is on the right
- `+` & `−` must be surrounded by space (`×` & `/` do not require space, but use it for consistency)
- `/` by `0` gives an error
- Be careful using `calc()` with `<percentage>`s for `width` & `height` on tables

HTML

```
<p>
  Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!
</p>
```

CSS (SCSS)

```
html {
  font-size: calc(1em + 1vw)
}
```

JS

Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!

HTML

```
<p>
  Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!
</p>
```

CSS (SCSS)

```
html {
  font-size: calc(1em + 1vw);
}

```

JS

Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!

HTML

```
<p>
  Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!
</p>
```

CSS (SCSS)

```
html {
  font-size: calc(1em + 1vw);
}

```

JS

Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!

HTML

```
<p>
  Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!
</p>
```

CSS (SCSS)

```
html {
  font-size: calc(1em + 1vw);
}
```

JS

Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!

HTML

```html
<div></div>
```

CSS (SCSS)

```scss
div {
  width: calc(100vw - 60px);
}
```

JS

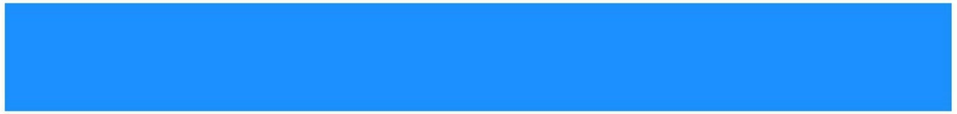
HTML

```
<div></div>
```

CSS (SCSS)

```
div {
  width: calc(100vw - 60px);
}
```

JS



HTML

```html
<div></div>
```

CSS (SCSS)

```scss
div {
  width: calc(100vw - 60px);
}
```

JS

					iOS		
`calc()`	9 	12	16	6.1	7	26	67
`<color>` value	–	–	59	6	6	–	–
Nested `calc()`	–	16	48	11	11	51	Y
`<number>` value	9	12	48	6	6	31	4.4
`<gradient>` color stops	9	12	19	6	6	19	Y
`@media` expressions	–	79	59	12	12	66	Y

<color>

`<color>`

Represents a *color* in the sRGB color space

Named & Hexadecimal

Descriptor	Value
Named colors	`magenta`
6-digit RGB hexadecimal	`#RRGGBB`
8-digit RGB hexadecimal	`#RRGGBBAA`
3-digit RGB hexadecimal	`#RGB`
4-digit RGB hexadecimal	`#RGBA`

Specifications			Color	Keyword	RGB cubic coordinates	Live Example
CSS3	CSS2	CSS1		black	=rgb(0, 0, 0)	
				silver	=rgb(192, 192, 192)	
				gray[*]	=rgb(128, 128, 128)	
				white	=rgb(255, 255, 255)	
				maroon	=rgb(128, 0, 0)	
				red	=rgb(255, 0, 0)	
				purple	=rgb(128, 0, 128)	
				fuchsia	=rgb(255, 0, 255)	
				green	=rgb(0, 128, 0)	
				lime	=rgb(0, 255, 0)	
				olive	=rgb(128, 128, 0)	
				yellow	=rgb(255, 255, 0)	
				navy	=rgb(0, 0, 128)	
				blue	=rgb(0, 0, 255)	
				teal	=rgb(0, 128, 128)	
				aqua	=rgb(0, 255, 255)	
				orange	=rgb(255, 165, 0)	

Color keywords

CSS 1: 16

CSS 2: 1

CSS 3: 128

Rebecca Meyer (2008–2014), namesake for `rebeccapurple`

In 2024 the CSS-Next Community Group introduced the first official CSS logo

The background color is `rebeccapurple`

Numeral systems

Humans use *base 10*: `0`–`9`

Computers use *base 2* (*binary*): `0` & `1`

Colors can be expressed with *hexadecimal*: `0`–`F` (`0`–`9`, then `A`–`F`)

Hexadecimal notation

`#RRGGBB`

`00`: no color at all

`FF`: full color

`#000000`: no red, no green, no blue, so black

`#FFFFFF`: full red, full green, full blue, so white

`#FF0000` (or `#ff0000`): full red, no green, no blue

`#00FF00`: no red, full green, no blue

`#0000FF`: no red, no green, full blue

`#FFFF00`: full red, full green, no blue

`#00FFFF`: no red, full green, full blue

`#E19427`: almost full red, a bit over halfway green, low blue

Quick — what does `#FFA07A` look like?

How about `#32CD32`? `#8A2BE2`? `#DA70D6`?

The cryptic nature of hexadecimal color values are a problem!

Quick — what does `#FFA07A` look like?

How about `#32CD32`? `#8A2BE2`? `#DA70D6`?

The cryptic nature of hexadecimal color values are a problem!

Roses are #FF0000,
violets are #0000FF

✏️ SIDE NOTE

Hexademical numbers can use either capital or lowercase letters, so these are exactly the same:

```
#0455BF
#0455bf

#F3F112
#f3f112
```

#FFFFFF	
#111111	#EEEEEE
#222222	#DDDDDD
#333333	#CCCCCC
#444444	#BBBBBB
#555555	#AAAAAA
#666666	#999999
#777777	#888888

#000000	
#111111	#EEEEEE
#222222	#DDDDDD
#333333	#CCCCCC
#444444	#BBBBBB
#555555	#AAAAAA
#666666	#999999
#777777	#888888

Gradations of gray

Hexadecimal notation may add a 4th pair of digits for transparency: the *alpha channel*

`00` represents a fully transparent color & `FF` represent a fully opaque color

`#0000FF66`: 66 (40%) alpha channel
`#F3F112AA`: AA (67%) alpha channel

Hexadecimal values & their alpha channel (transparency) equivalents

- `00` is 0% — completely transparent
- `33` is 20%
- `66` is 40%
- `99` is 60%
- `CC` is 80%
- `FF` is 100% — completely opaque

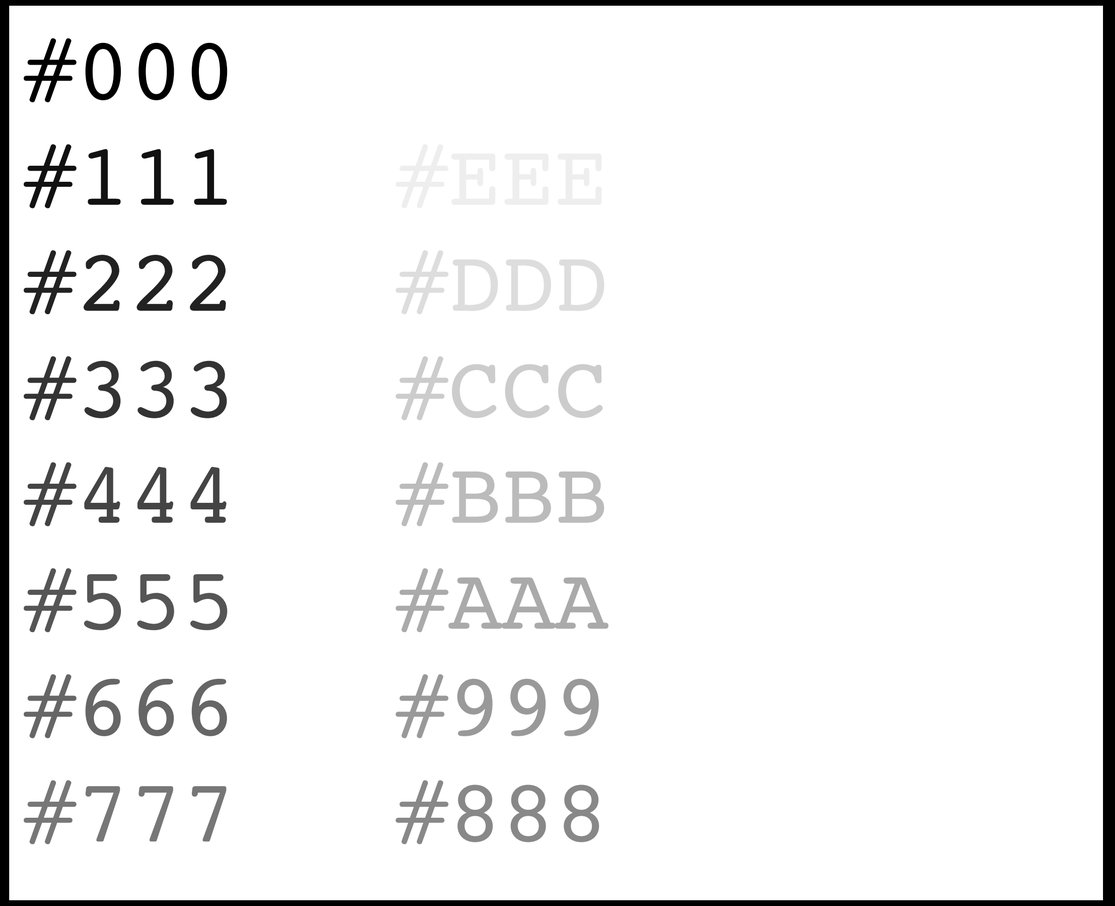
3-digit hexadecimal

`#RGB`

If all three color components (the R, G, & B) are matching pairs, you can abbreviate the hexadecimal notation

`#FF0000` → `#F00`
`#33AA77` → `#3A7`
`#BBFF33` → `#BF3`
`#DDEEFF` → `#DEF`

#FFF
#111 #EEE
#222 #DDD
#333 #CCC
#444 #BBB
#555 #AAA
#666 #999
#777 #888



Gradations of gray

4-digit hexadecimal

#RGBA

Hexadecimal notation may add a 4th hexadecimal number for the alpha channel, if the alpha channel could be expressed by 2 matching numbers

#FF0000 → #F00
#33AA77 → #3A7
#BBFF33 → #BF3
#DDEEFF → #DEF

					iOS		
Named colors	3	12	1	1	1	1	1
`rebeccapurple`	11	12	33	9	8	38	Y
`#RRGGBB`	3	12	1	1	1	1	1
`#RRGGBBAA`	–	79	49	10	10	62	Y
`#RGB`	3	12	1	1	1	1	1
`#RGBA`	–	79	49	10	10	62	Y

`rgb()` & `hsl()`

Descriptor	Value
RGB `<number>`	`rgb(255 0 51)`
RGB `<percentage>`	`rgb(100% 0% 20%)`
RGB + Alpha	`rgb(255 0 51 / .7)`
HSL	`hsl(348 100% 50%)`
HSL + Alpha	`hsl(348 100% 50% / .7)`

The old method used commas, e.g., `hsl(348,100%,50%)`, & specific alpha functions, e.g., `hsla(348,100%,50%,.5)`

RGB functional notation

```
rgb(red green blue)
```

Expressed as either:

» `<integer>`s between `0` (black) & `255` (white)
» `<percentage>`s between `0` (black) & `100%` (white)

May add a 4th place for alpha channel

```
rgb(red green blue / alpha)
```

The alpha channel is a number between `0` & `1`, where:

» `0` is fully transparent
» `1` is fully opaque

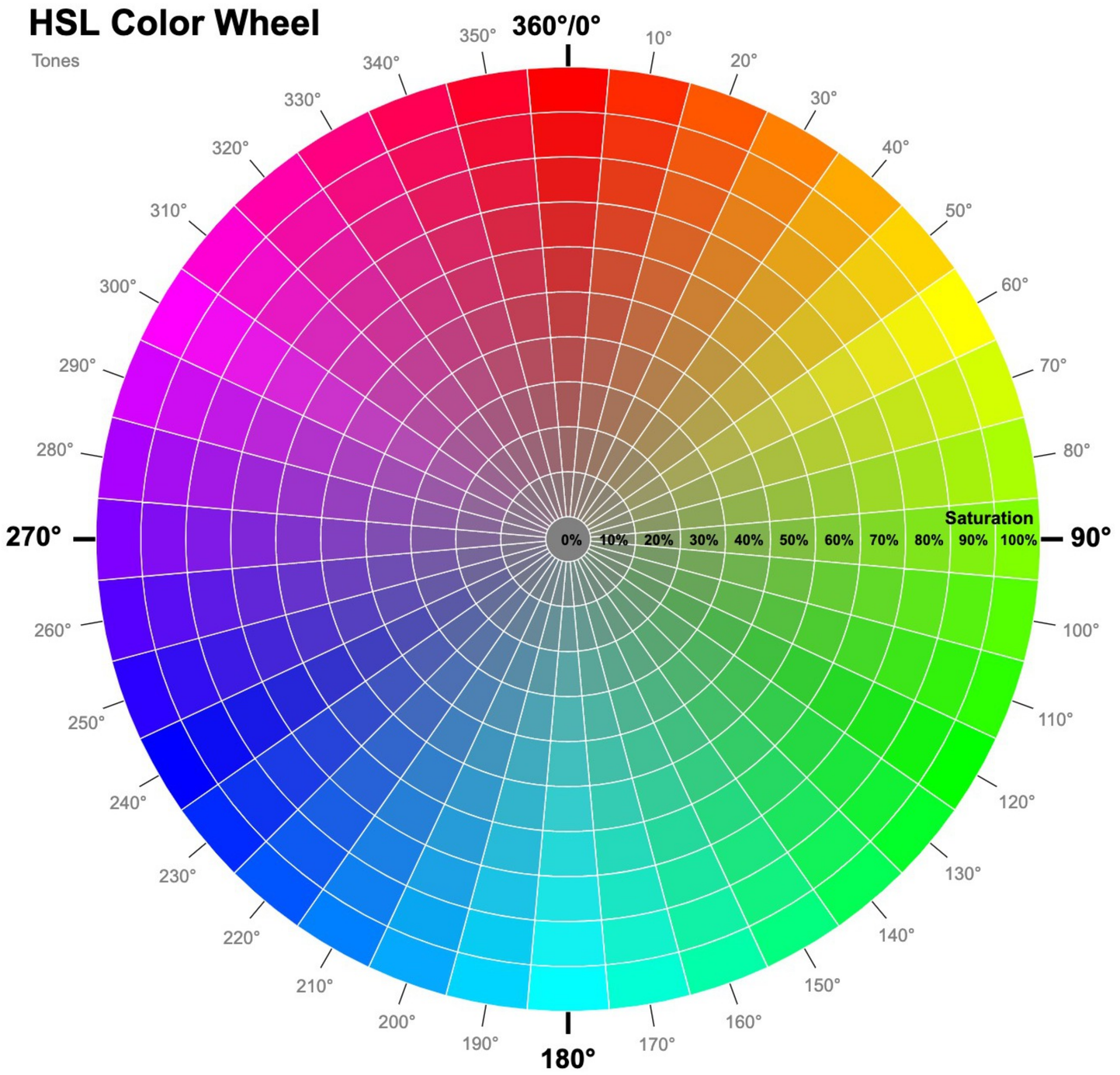
```
rgb(255 0 0)
rgb(255 0 0 / .5)
rgb(0 255 201)
rgb(0 255 201 / .66)
```

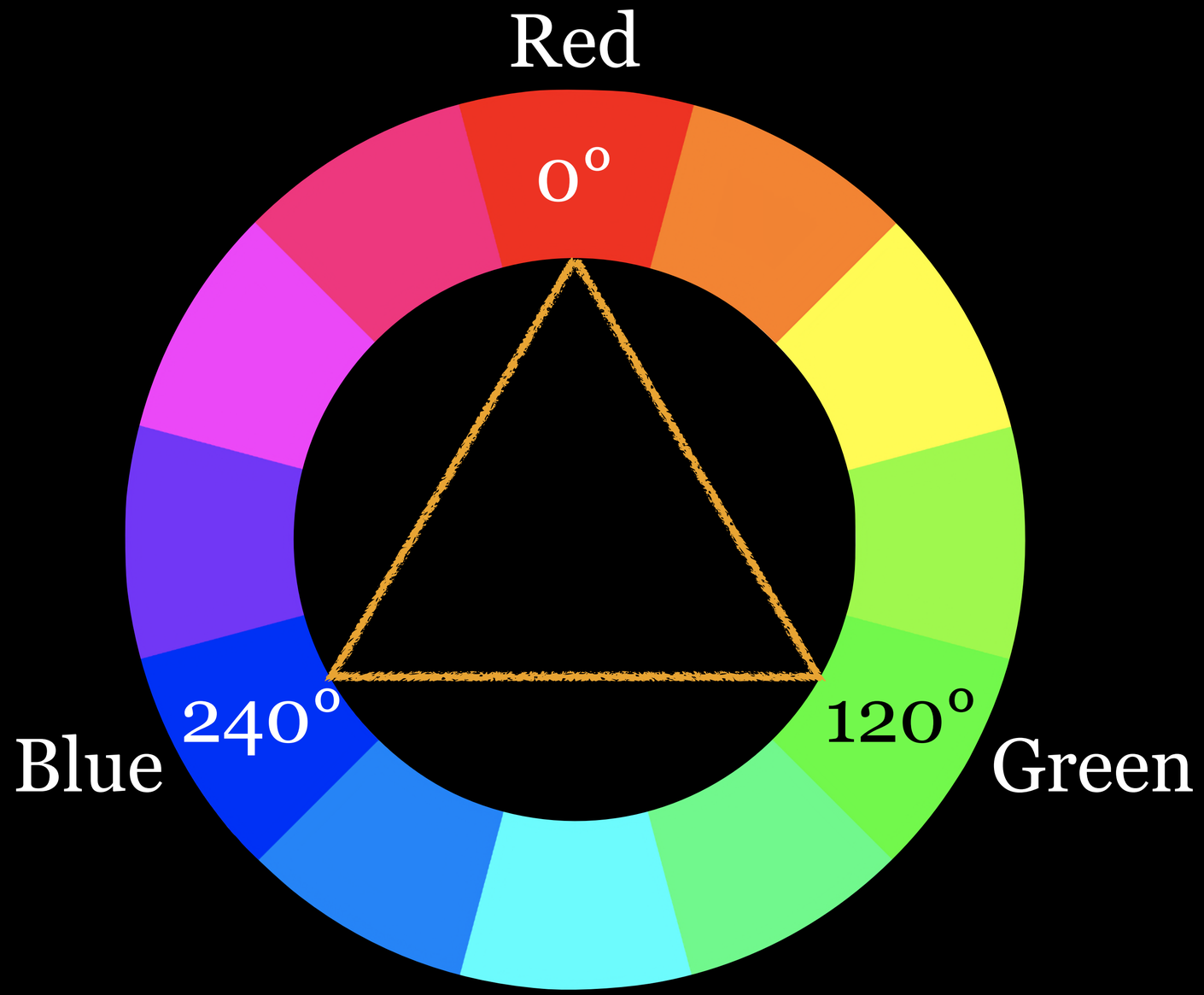
HSL functional notation

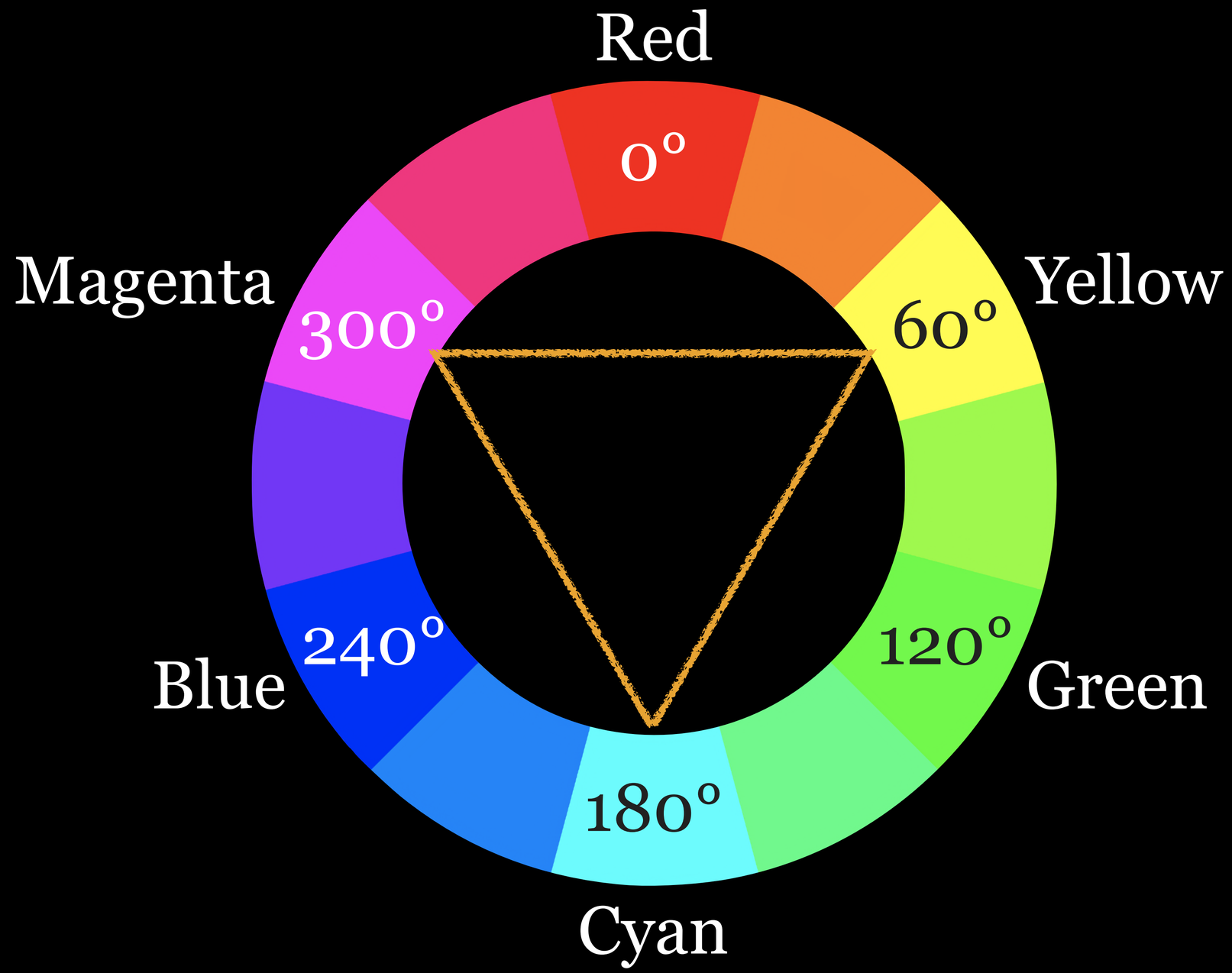
```
hsl(hue saturation lightness)

hsl(0–360 0–100% 0–100%)
```

Hue is another word for color

In HSL notation, it's an `<integer>` between `0` & `360`


HSL Color Wheel
Tones
360°/0°
10°
20°
30°
40°
50°
60°
70°
80°
90°
100°
110°
120°
130°
140°
150°
160°
170°
180°
190°
200°
210°
220°
230°
240°
250°
260°
270°
280°
290°
300°
310°
320°
330°
340°
350°
Saturation
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%


Red
0°
120°
Green
240°
Blue

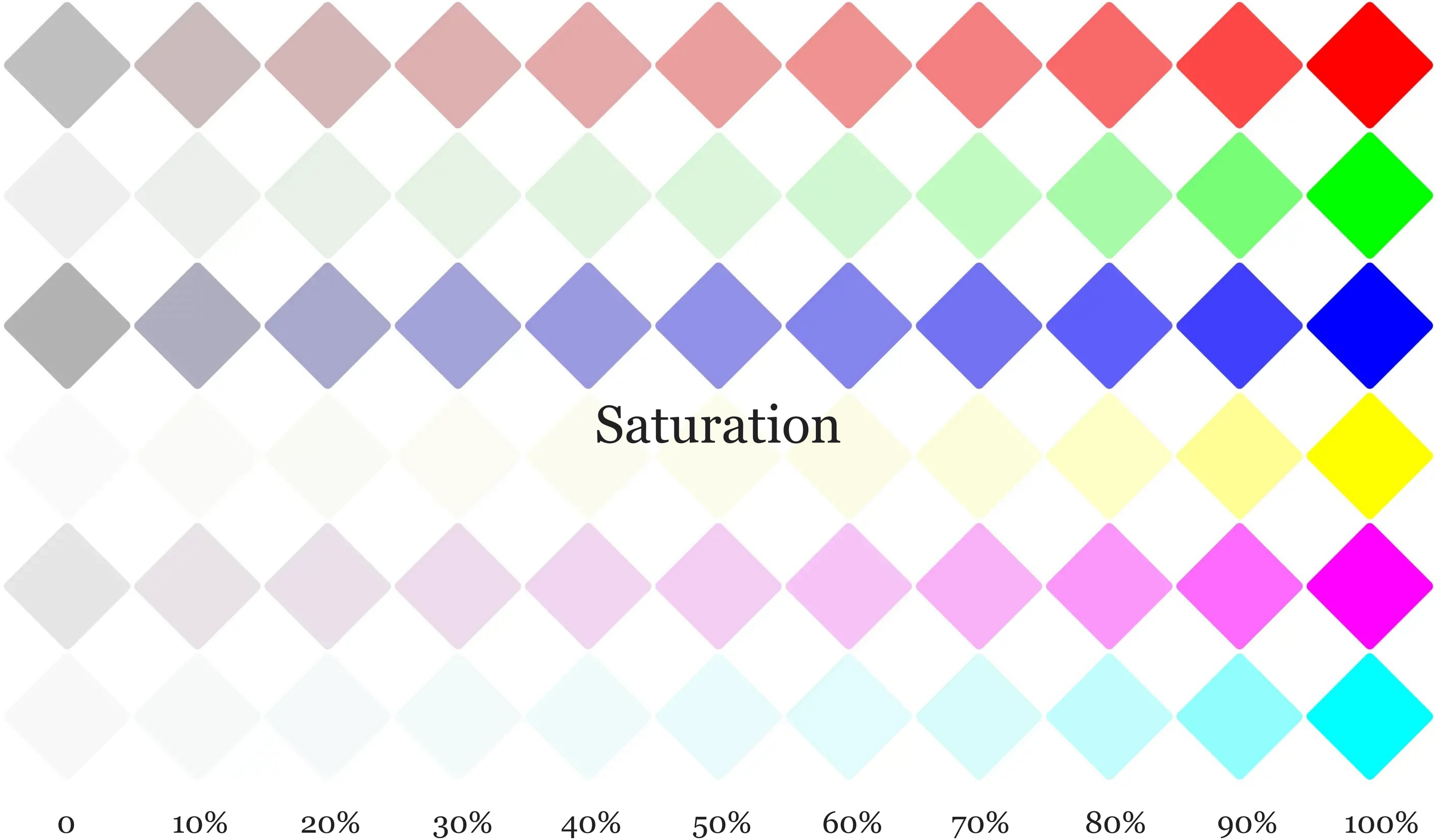

Red
0°
Yellow
60°
Green
120°
Cyan
180°
Blue
240°
Magenta
300°

Saturation (AKA *chroma*) describes how vivid, intense, or pure a hue is

In HSL notation, it's a `<percentage>` between `0` & `100%`

`0%`: Completely gray & washed out with no intensity

`100%`: Full, pure color without any grayness


Saturation
0
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

Lightness defines how much black or white the hue has in it

In HSL notation, it's a `<percentage>` between `0` & `100%`

`0%`: Completely black

`50%`: Fully vibrant hue without any black or white

`100%`: Completely white


Lightness
0
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

May add a 4th place for alpha channel

```
hsl(hue saturation lightness / alpha)
```

The alpha channel is a number between `0` & `1`, where:

» `0` is fully transparent
» `1` is fully opaque

```
hsl(360 100% 100%)
hsl(360 100% 100% / .5)
hsl(165 87% 100%)
hsl(165 87% 100% / .66)
```

✏️ SIDE NOTE

In the olden days you had to use commas with `rgb()` & `hsl()`:

`rgb(255, 159, 0)` • `rgb(255, 159, 0, .5)`
`hsl(39, 100%, 43%)` • `hsla(39, 100%, 43% / .5)`

Since 2018 you don't have to, which is good (& you can use percentages with `rgb()` too)!

`rgb(255 159 0)` • `rgba(255 159 0 / .5)`
`hsl(39 100% 43%)` • `hsla(39 100% 43% / .5)`

					iOS		
`rgb()`	4	12	1	1	1	1	1
`rgba()`	9	12	3	3.1	3.2	4	65
`hsl()`	9	12	1	3.1	5.1	1	2.3
`hsla()`	9	12	3	3.1	3.2	4	65
Space-separated values	–	79	52	12.1	12.2	65	65

Keywords

2 keywords for `<color>`

`transparent`: *Fully transparent*

`currentcolor`: Value of the *selected element's font color*, allowing you to use the color value on properties that do not receive it by default

HTML

```
<p>Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!</p>
```

CSS

```
p {
  color: green;
  border-color: currentcolor;
}
```

JS

Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!

					iOS		
transparent	9	12	3	3.1	3.2	1	Y
currentcolor	9	12	1.5	4	3.2	1	Y

Thank you!

scott@granneman.com
www.granneman.com
@scottgranneman@mastodon.social

jans@websanity.com
websanity.com

CSS Data Types

Classifying Data Used as CSS Values

R. Scott Granneman & Jans Carton

Changelog

2025-02-24 2.9: Added more detail on `rebeccapurple`; small typographical & other fixes

2025-02-23 2.8: Added screenshots & details to *Font-Relative* `<length>`; completely updated `<color>`; added headings for each data type; moved screenshot of W3C spec to front; updated theme to Granneman 1.14; minor formatting & fixes

Changelog

2023-12-10 2.7: Changed Twitter link to Mastodon

2023-04-04 2.6: Added full list of data types; added slides re cryptic nature of hexadecimal colors; added much better examples of HSL notation; clarified RGB & HSL notation

Licensing of this work