**Make your code look pretty**

If you notice and appreciate the difference between

a <- c(1:7, NA)

mean(a, na.rm = TRUE)

and

a <- c(1:7, NA)

mean(a, na.rm = TRUE)

Syntax highlighting! Just do it. Life is better when things are colourful.

Syntax highlighting means some elements of code blocks, like functions, operators, comments, etc. get styled differently: they could be colored or in italic.

Now, how do the colors of the second block appear?

First of all, it’s a code block with language information, in this case R (note the r after the backticks),

```r

a <- c(1:7, NA)

mean(a, na.rm = TRUE)

```

as opposed to

```

a <- c(1:7, NA)

mean(a, na.rm = TRUE)

```

without language information, that won’t get highlighted – although some syntax highlighting tools, not Hugo Chroma, do some guessing.

There are in general two ways in which colors are added to code blocks, **client-side syntax highlighting** and **server-side syntax highlighting**.  
The latter is what Hugo supports nowadays but let’s dive into both for the sake of completeness ([2](https://ropensci.org/technotes/2020/04/30/code-highlighting/#fn:2)).

**Client-side syntax highlighting**

In this sub-section I’ll mostly refer to [highlight.js](https://highlightjs.org/) but principles probably apply to other client-side syntax highlighting tools.  
The “client-side” part of this phrase is that the html that is served by your website host does not have styling for the code.  
In highlight.js case, styling appears after a JS script is loaded and applied.

If we look at [a post of Mara Averick’s](https://maraaverick.rbind.io/2018/07/palettes-archer-poster-edition/) at the time of writing, the html of a block is just

<pre class="r"><code>pal\_a <- extract\_colours("<https://i.imgur.com/FyEALqr.jpg>", num\_col = 8)

par(mfrow = c(1,2))

pie(rep(1, 8), col = pal\_a, main = "Palette based on Archer Poster")

hist(Nile, breaks = 8, col = pal\_a, main = "Palette based on Archer Poster")code>pre>

Now, using Firefox Developer Console,

we see colors come from CSS classes starting with “hljs”.

And in the head of that page (examined via “View source”), there’s

<script src="<https://cdnjs.cloudflare.com/ajax/libs/highlight.js/9.9.0/highlight.min.js>">script>

<script>hljs.initHighlightingOnLoad();script>

which is the part loading and applying highlight.js to the page.  
Now, how does it know what’s for instance a string in R?  
If we look at [highlight.js highlighter for the R language](https://github.com/highlightjs/highlight.js/blob/master/src/languages/r.js), authored by [Joe Cheng](https://twitter.com/jcheng) in 2012, it’s a bunch of regular expressions, for the instance.

className: 'string',

contains: [hljs.BACKSLASH\_ESCAPE],

variants: [

{begin: '"', end: '"'},

{begin: "'", end: "'"}

]

When using highlight.js on your website, you might need to specify R as a supplementary language in your config, since some languages are bundled by default whilst others are not.

A big downside of client-side syntax highlighting is loading time:  
it appears quite fast if your internet connection isn’t poor, but you might have noticed code blocks changing aspect when loading a web page (first not styled, then styled).  
Moreover, Hugo now supports, and uses by default, an alternative that we’ll describe in the following subsection and take advantage of in this post’s second section.

**Server-side syntax highlighting**

In server-side syntax highlighting, with say [Pygments](https://pygments.org/) or Chroma (Hugo default), your website html as served already has styling information.

With library(chroma), that styling information is either:

* hard-coded in html, as is since recently the case of [tidyverse.org](https://www.tidyverse.org/) and [blog.r-hub.io](https://blog.r-hub.io/);

The html source for one of the blocks of [the page screenshot above](https://www.tidyverse.org/blog/2020/04/dplyr-1-0-0-colwise/) is

div class="highlight"><pre style=";-moz-tab-size:4;-o-tab-size:4;tab-size:4"><code class="language-r" data-lang="r">df <span style="color:#666">%>%span>

<span style="color:#00f">group\_byspan>(g1, g2) <span style="color:#666">%>%span>

<span style="color:#00f">summarisespan>(a <span style="color:#666">=span> <span style="color:#00f">meanspan>(a), b <span style="color:#666">=span> <span style="color:#00f">meanspan>(b), c <span style="color:#666">=span> <span style="color:#00f">meanspan>(c), d <span style="color:#666">=span> <span style="color:#00f">meanspan>(c))

code>pre>div>

The style used is [indicated in the website config](https://gohugo.io/getting-started/configuration-markup#highlight) and picked from [Chroma style gallery](https://xyproto.github.io/splash/docs/all.html).

* via the use of CSS classes also indicated in html, as is the case of this website.

The html of the block seen above is

<div class="highlight"><pre class="chroma"><code class="language-r" data-lang="r"><span class="nf">install.packagesspan><span class="p">(span><span class="s">"parzer"span><span class="p">,span> <span class="n">reposspan> <span class="o">=span> <span class="s">"<https://dev.ropensci.org/>"span><span class="p">)span>

code>pre>div>

and it goes hand in hand with having styling for different “.chroma” classes in our website CSS.

.chroma .s { color: #a3be8c }

To have this behaviour, in our website config there’s

pygmentsUseClasses=true

which confusingly enough uses the name “Pygments”, not Chroma, for historical reasons.  
You’d use CSS like we do if none of Chroma suited you, if you wanted to make sure the style colors respect WCAG color contrast guidelines (see [last section](https://ropensci.org/technotes/2020/04/30/code-highlighting/#accessibility)), or if you want to add a button switching the CSS applied to the classes, which we did for this note [using a dev.to post by Alberto Montalesi](https://dev.to/albertomontalesi/add-dark-mode-to-your-website-with-just-a-few-lines-of-code-5baf).   
Click the button below!

It will also let you switch back to light mode.

To generate a stylesheet for a given style, [use Hugo hugo gen chromastyles --style=monokai > syntax.css command](https://gohugo.io/content-management/syntax-highlighting/#generate-syntax-highlighter-css).  
You can then use the stylesheet as is, or tweak it.

How does Chroma know what parts of code is of the string class for instance?  
Once again, regular expressions help, in this case in what is called a lexer.  
Chroma is inspired by Pygments, and in [Pygments docs](https://pygments.org/docs/quickstart/) it is explained that *“A lexer splits the source into tokens, fragments of the source that have a token type that determines what the text represents semantically (e.g., keyword, string, or comment)."*  
In R Lexer, ported from Pygments to Chroma by Chroma maintainer, for strings we e.g. see

{`\'`, LiteralString, Push("string\_squote")},

{`\"`, LiteralString, Push("string\_dquote")},

// ... code

"string\_squote": {

{`([^\'\\]|\\.)\*\'`, LiteralString, Pop(1)},

},

"string\_dquote": {

{`([^"\\]|\\.)\*"`, LiteralString, Pop(1)},

},

Chroma works on Markdown content, so if you use [blogdown to generate pages as html](https://bookdown.org/yihui/blogdown/output-format.html), you can only use client-side highlighting, like [this tidyverse.org page](https://www.tidyverse.org/blog/2017/09/erratum-tidyr-0.7.0/).  
By default nowadays Hugo does server-side syntax highlighting but you could choose to turn it off via [codeFences = false](https://gohugo.io/getting-started/configuration-markup#highlight).

We have now seen how Hugo websites have [syntax highlighting, which for Yihui Xie *“is only for cosmetic purposes”*](https://yihui.org/en/2017/07/on-syntax-highlighting/). Well, Chroma actually also offers one thing more: line numbering and line highlighting!

**Emphasize parts of your code**

With Chroma, you can [apply special options to code blocks defined with fences, i.e. starting with three backticks and language info, and ending with three backticks](https://gohugo.io/content-management/syntax-highlighting/)

**On Chroma options for line highlighting**

See how

```r {hl\_lines=[1,"4-5"]}

library("dplyr")

df %>%

mutate(date = lubridate::ymd(date\_string)) %>%

select(- date\_string)

str(df)

nrow(df)

```

is rendered below: lines 1 and 4 to 5 are highlighted.

library("dplyr")

df %>%

mutate(date = lubridate::ymd(date\_string)) %>%

select(- date\_string)

str(df)

nrow(df)

There are also options related to line numbering.

```r {hl\_lines=[1,"4-5"],linenos=table,linenostart=3}

library("dplyr")

df %>%

mutate(date = lubridate::ymd(date\_string)) %>%

select(- date\_string)

str(df)

nrow(df)

```

gives a code block with line numbered as table (easier for copy-pasting the code without line numbers), starting from number 3.

|  |  |
| --- | --- |
| 3  4  5  6  7  8 | library("dplyr")  df %>%  mutate(date = lubridate::ymd(date\_string)) %>%  select(- date\_string)  str(df)  nrow(df) |

You can also configure line numbering [for your whole website](https://gohugo.io/getting-started/configuration-markup#highlight).

The real magic to me is that if you write your code from R Markdown you can

* apply the options to the source chunk using a knitr hook like the one defined;
* use R code to programmatically produce code block between fences, e.g. choosing which lines to highlight.

**knitr hook to highlight lines of source code**

Our hook is

# knitr hook to use Hugo highlighting options

knitr::knit\_hooks$set(

source = function(x, options) {

hlopts <- options$hlopts

paste0(

"```", "r ",

if (!is.null(hlopts)) {

paste0("{",

glue::glue\_collapse(

glue::glue('{names(hlopts)}={hlopts}'),

sep = ","

), "}"

)

},

"\n", glue::glue\_collapse(x, sep = "\n"), "\n```\n"

)

}

)

The chunk[6](https://ropensci.org/technotes/2020/04/30/code-highlighting/#fn:6)

```{r name-your-chunks, hlopts=list(linenos="table")}

a <- 1+1

b <- 1+2

c <- 1+3

a + b + c

```

is rendered as

|  |  |
| --- | --- |
| 1  2  3  4 | a <- 1+1  b <- 1+2  c <- 1+3  a + b + c |

[1] 9

PSA! Note that if you’re after line highlighting, or function highlighting, for R Markdown documents *in general*,!

**Produce line-highlighted code blocks with glue/paste0**

What Chroma highlights are code blocks with code fences, which you might as well generate from R Markdown using some string manipulation and knitr results="asis" chunk option. E.g.

```{r, results="asis"}

script <- c(

"a <- 1",

"b <- 2",

"c <- 3",

"a + b + c")

cool\_lines <- sample(1:4, 2)

cool\_lines <- stringr::str\_remove(toString(cool\_lines), " ")

fences\_start <- paste0('```', 'r {hl\_lines=[', cool\_lines,']}')

glue::glue\_collapse(

c(fences\_start, script, "```"),

sep = "\n")

```

will be knit to produce

a <- 1

b <- 2

c <- 3

a + b + c

This is a rather uninteresting toy example since we used randomly drawn line numbers to be highlighted, but you might find use cases for this.  
We used such an approach in the recent [blog post about Rclean](https://ropensci.org/blog/2020/04/21/rclean/), actually!

**Accessibility**

Since highlighting syntax and lines changes the color of things, it might make it harder for some people to read your content, so the choice color is a bit more than about cosmetics.

*Disclaimer: I am not an accessibility expert. Our efforts were focused on contrast only, not differences between say green and red, since these do not endanger legibility of code.*

We referred to [the contrast criterion of the Web Content Accessibility Guidelines of the World Wide Web Consortium](https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html) that state *The intent of this Success Criterion is to provide enough contrast between text and its background so that it can be read by people with moderately low vision (who do not use contrast-enhancing assistive technology).*

For instance, comments could be lighter or darker than code, but it is crucial to pay attention to the contrast between comments and code background!  
Like [Max Chadwick](https://maxchadwick.xyz/blog/syntax-highlighting-and-color-contrast-accessibility), we darkened colors of a default Chroma style, until it passed on [an online tool](https://color.a11y.com/).  
Interestingly, this online tool can only work with a stylesheet: for a website with colors written in-line (Hugo default of pygmentsUseClasses=false), it won’t pick up color contrast problems.  
We chose friendly as a basis because its background can stand out a bit against white, without being a dark theme, which might be bad on a mobile device in direct sunglight.  
Comments are moreover in italic which helps distinguish them from other code parts.

Our approach is less good than [having an actual designer pick colors like what Codepen recently did](https://blog.codepen.io/2019/09/09/new-high-contrast-syntax-highlighting-themes/), but will do for now.  
Apart from that, we only know of [Eric Bailey’s a11y dark and light themes as highlighting themes that are advertised as accessible](https://github.com/ericwbailey/a11y-syntax-highlighting).

A further aspect of contrast when using Chroma is that when highlighting a line, its background will have a different color than normal code.

This color also needs to not endanger the contrast between code and code background, so if your code highlighting is “dark mode”, yellow highlighting is probably a bad idea: in this post, for the dark mode, we used the “fruity” Chroma style but with #301934 as background color for the highlighted lines.  
It would also be a bad idea to only rely on line highlighting, as opposed to commenting code blocks, since some readers might not be able to differentiate highlighted lines.

Commenting code blocks is probably a good practice in general anyway, explaining what it does instead of just sharing the code like you’d share a gist.