Test your CSS/Sass with the Chrome Developer Tools

Ben Frain

Who is this guy?

Author of:

- ‘Responsive web design with HTML5 and CSS3’
- ‘Sass and Compass for Designers’

Front-end developer at **bet365.com**

Disclosure

- No affiliation with Google.
- I find Chrome Developer Tools easiest to work with.
- Chrome developer relations team document new features well.
- My list of Dev Tool Resources: https://gist.github.com/benfrain/5908652

Who this is for

- People comfortable with HTML and CSS/Sass
- To help you troubleshoot style sheet problems more easily.
- We will cover Developer Tools basics
- Then we'll cover the CSS specific troubleshooting features.
- Time permitting, I'll share my screen and use them live.

Orientation/Dev Tool Essentials

- Selecting with the magnifying glass
- Moving nodes/viewing hierarchy
- Editing nodes
- Moving the Dev Tools
- Styles panel – adding styles
- Computed style of an element
- Toggling the pseudo state (hover/active) of elements

CSS specific tasks with Chrome Developer Tools

- ‘Rule Rot’; how to find out which rules are being used.
- Invalid CSS properties or values.
- How ‘performant’ is your CSS?
- Costly property combinations?
- Cutting out round-trips to the text editor.
- [Non Chrome] – configuring and using CSS Lint

Every browser is different

- Every browser interprets and paints CSS differently.
- Pages that perform well in Chrome, may perform differently in other browsers.
- However, doing some (any) testing of your CSS is better than none.

Caveats over. Let’s begin.

Invoking the Developer Tools

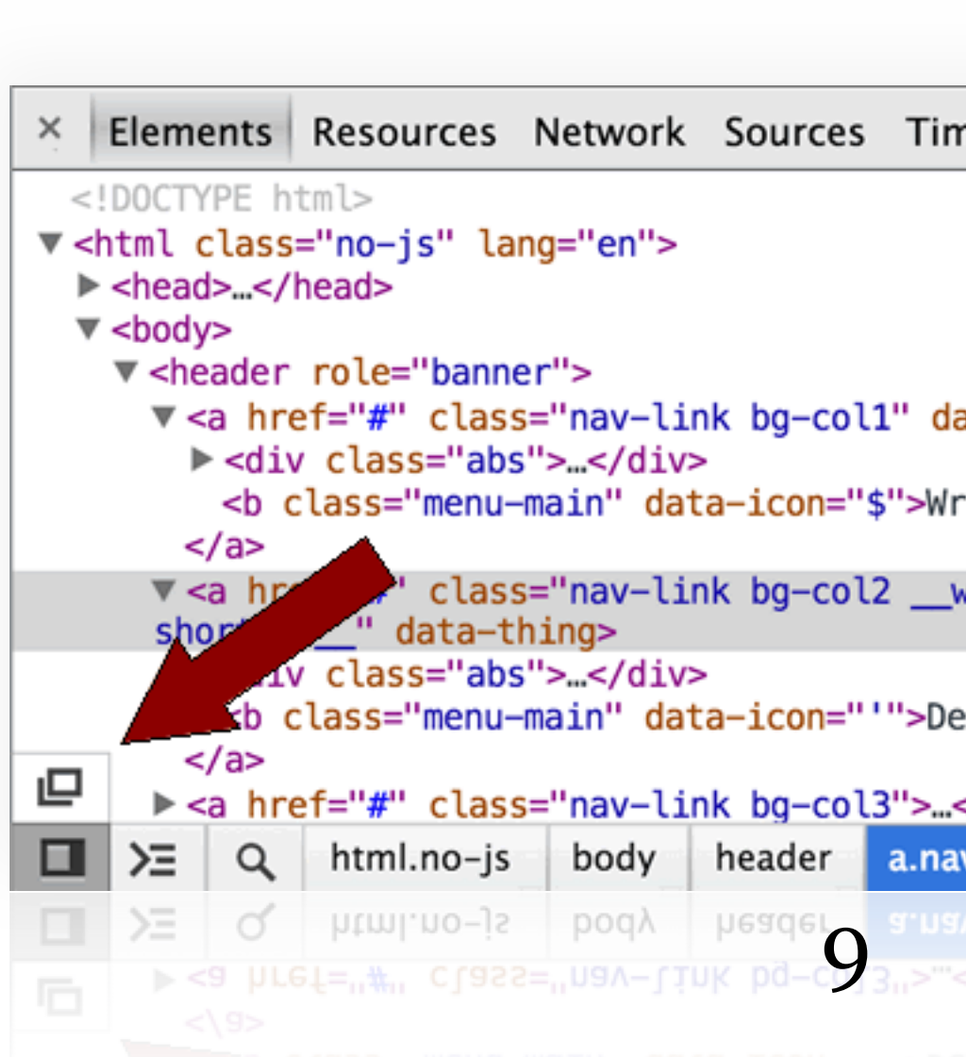
- Right/cmd-click an element and click ‘Inspect Element’

- Or choose View, Developer, Developer Tools from the main menu

- Use the shortcut key assigned

Positioning the Chrome Dev Tools: bottom, right or floating

- Drag the Dev Tools header to the right for trouble shooting slimmer viewports.
- You can always drag them back.
- Switch positions by simply clicking and holding the view button:



Selecting elements on the page

- Inspect by clicking a node in the Elements panel
- Or use the magnifying glass (bottom left) and hover your mouse over the relevant area.

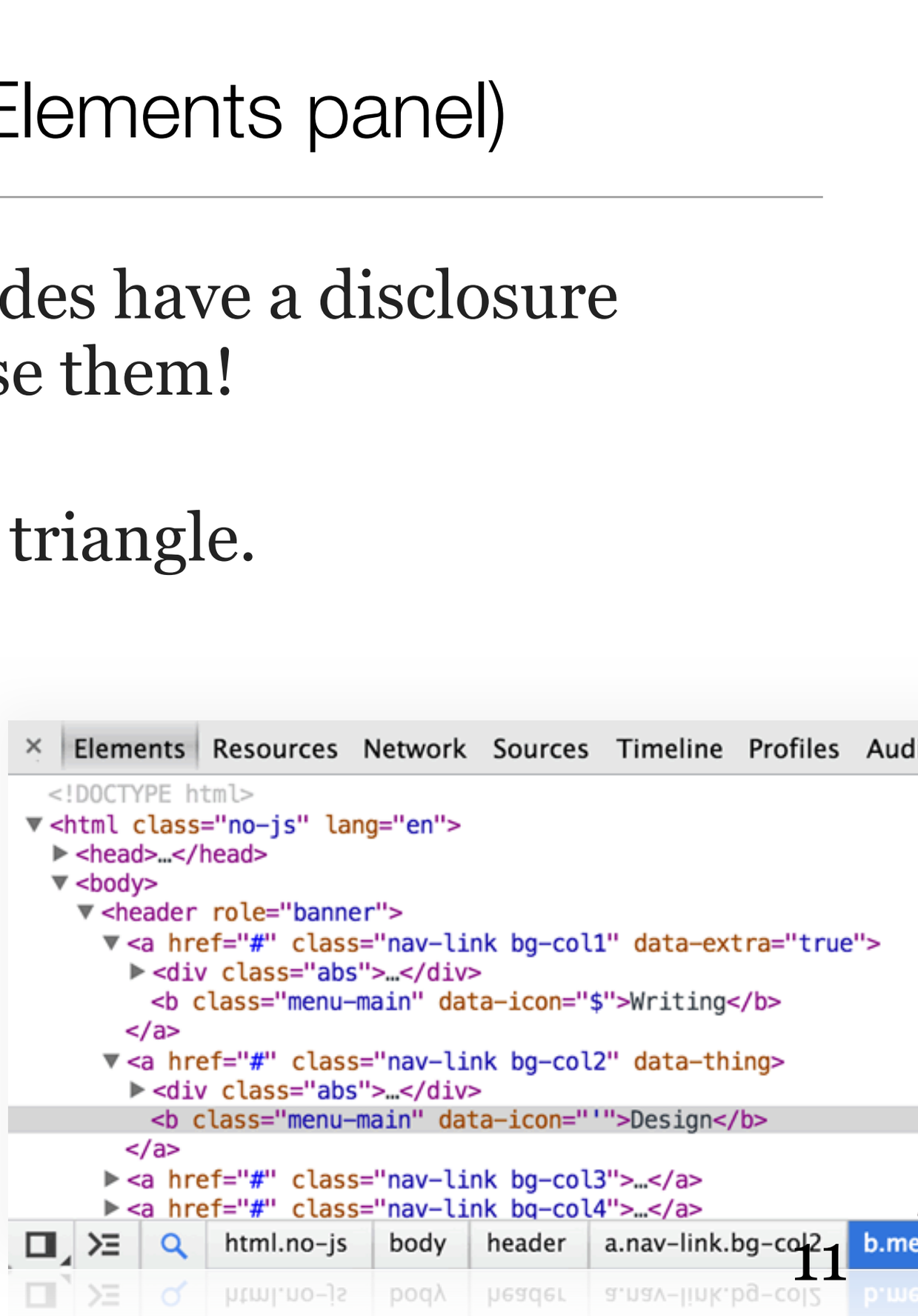
‘Nodes’ is just the term for elements within the DOM (Document Object Model) – a.k.a. the hierarchy of elements in the web page.

Expanding the DOM tree (Elements panel)

- Nodes that contains other nodes have a disclosure triangle. Click to, errr disclose them!

- ‘Leaf’ nodes lack a disclosure triangle.

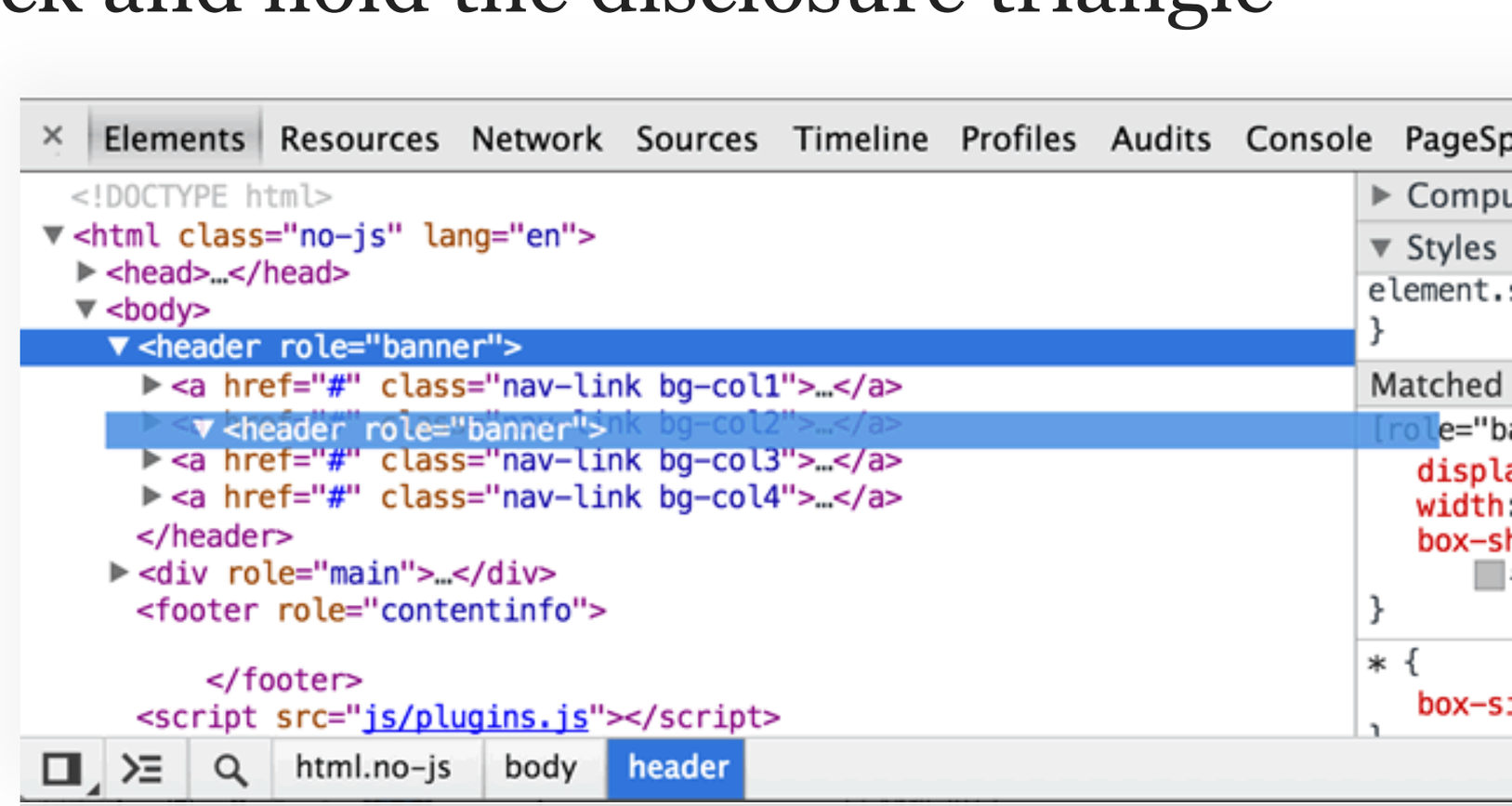
- A hierarchical path of the current node is displayed at the bottom.



How to move/edit/delete nodes

- Nodes can be moved, edited and deleted. Chrome will show the results instantly.

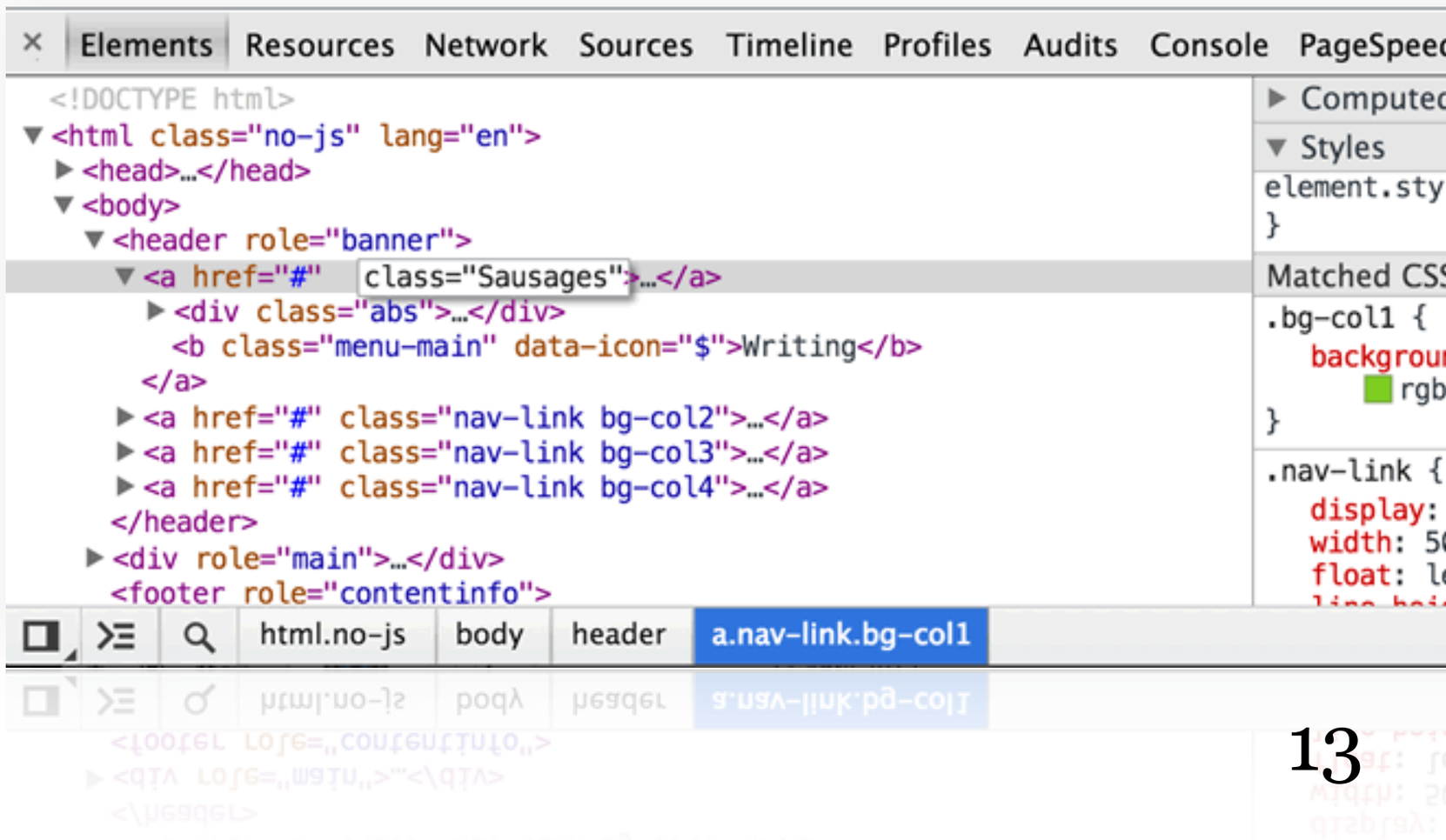
- To move a node, click and hold the disclosure triangle and drag.



Editing a node

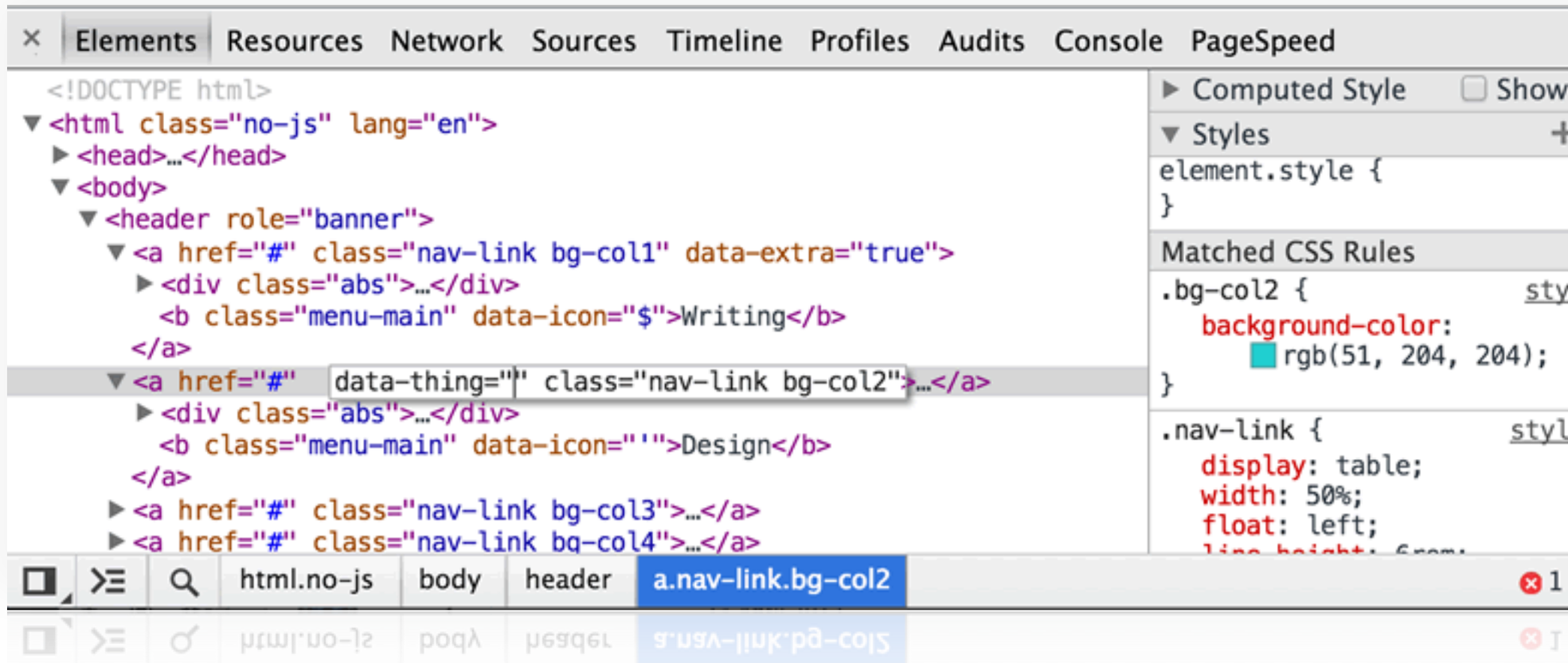
- Edit a node by double-clicking the attribute you want to edit. Amend it and then press Tab.

- Changes update instantly.



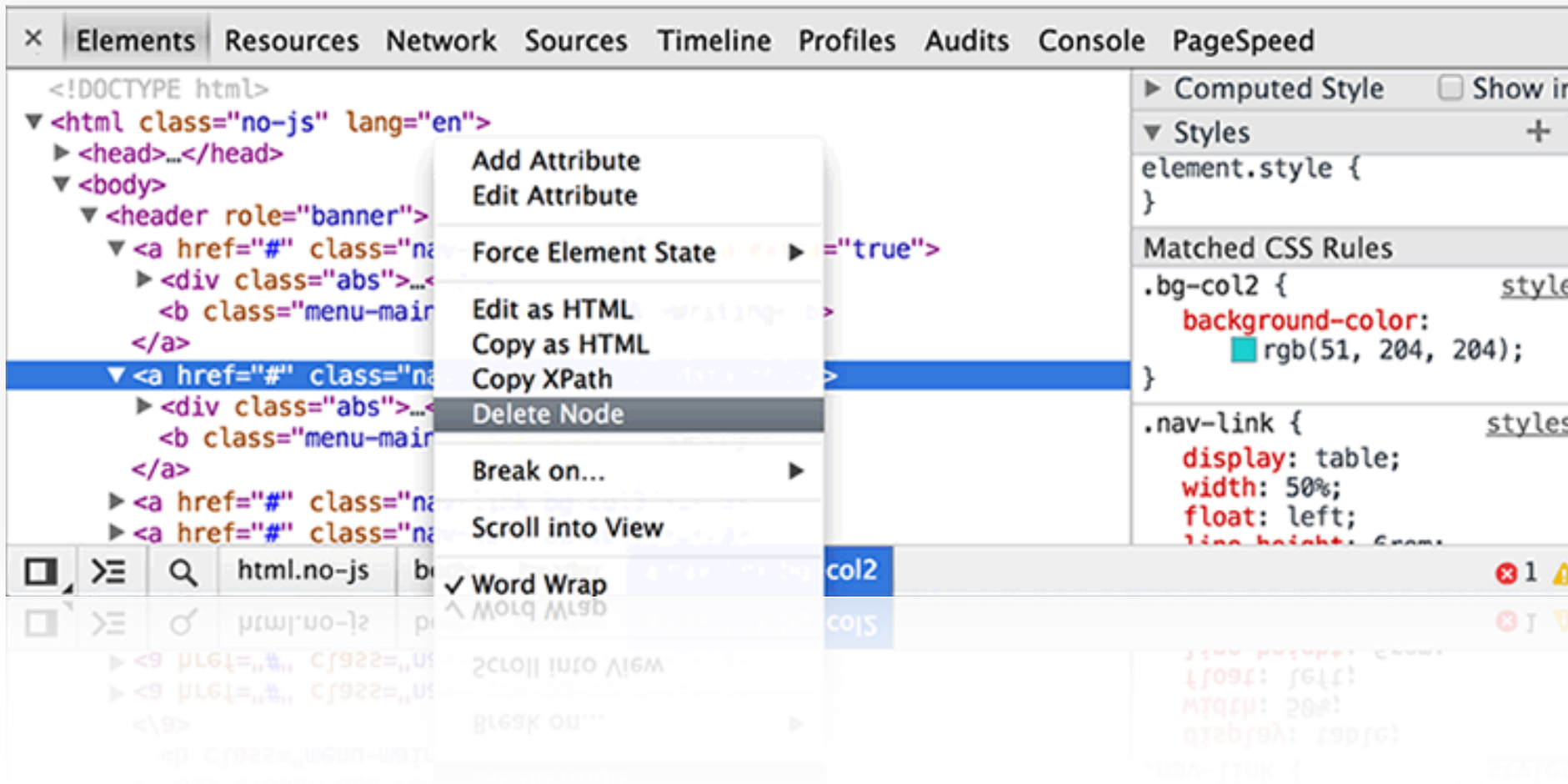
Add attributes to a node

- Add attributes by moving to the start/end of existing attribute:



Delete a node

- Delete a node by right-clicking on an element and choosing ‘Delete Node’ from the contextual menu:



Toggling visibility of a node

- With a node selected simply press ‘H’.
- This adds the class ‘web-inspector-hide-shortcut’ to the element.

Toggling visibility is not equivalent to display: none; – it retains it’s physical space and toggles visible styles.

Using the styles panel

- Styles panel provides direct access to the CSS rules being applied.

- Edit existing properties, add new ones and even save the values you amend directly back to the source Sass/CSS files.

Editing existing styles and adding new ones

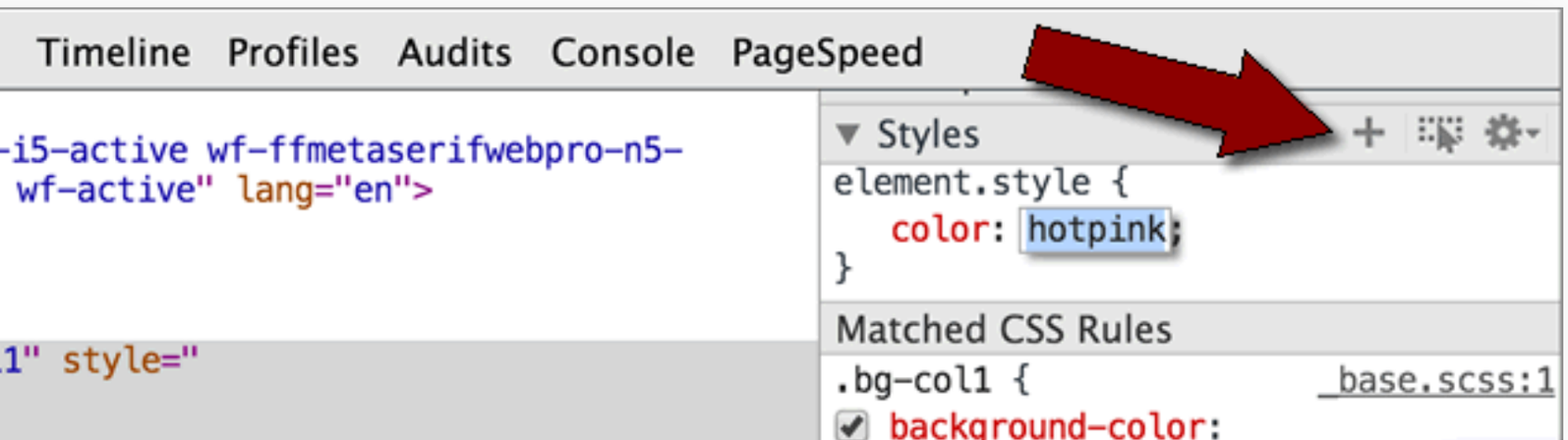
- Styles applied to an element show in the styles panel on the right

- ‘Computed styles’ are what the browser has interpreted the CSS to mean.

- Useful for comparing what you think should be applied and what the browser is actually applying!

The Styles section

- ‘Styles’ section adds inline rules. Adding styles there is similar to injecting styles with JavaScript.
- Styles header has extra buttons:
  - ‘New Style Rule’ button adds a new rule into a Web Inspector stylesheet (as opposed to inline on the element).
  - ‘Toggle element states’ button (hover, active, visited and focus).
  - ‘Settings’ button toggles how colour values are displayed (RGB/HEX/HSL).



‘Matched CSS Rules’ section

- ‘Matched CSS Rules’ includes rules from any stylesheets.
- Rules overwritten via specificity have a line through.
- Toggle individual properties with the tick box to the side.

Commented out CSS rules in the source Sass/CSS show with a line through and are unselected.

Manipulating values

- Double-click a value to edit it.
- Increment numeric values with up and down keys (hold shift to increment in 10s, hold alt to increment in .1s).
- To add a new property, click to the right of the closing curly brace.
- Chrome auto-completes CSS properties and values – handy when you can't remember the correct value.

Source location

- The ‘Matched CSS Rules’ section indicates where a rule originates. Click to open the source style sheet in the Sources tab.

- Working with Sass (*cough* buy my book ‘***Sass and Compass for Designers***’ *cough*)? With source map support enabled originating partial file is displayed.

- Clicking the Sass partial name opens it in the Source panel. From there you can **edit it directly and save it back to the source Sass files**.

Essentials Section Summary

- That was not exhaustive
- But we've covered enough to crack on with the Sass/CSS specific testing we need to do.
- Pull on your starched white lab coat.

Chrome Release Channels
(left to right: Stable, Dev, Canary)

- Different release ‘channels’: Stable, Beta, Dev and Canary.
  - Stable is for ‘muggles’.
  - Beta not much different.
  - Dev contains good mix of stability and experimental features.
  - Canary is full of dark arts (I heard a rumour it’s built purely from Horcruxes and crow spit).
- Canary will live happily alongside another Chrome.
- Get them here: http://www.chromium.org/getting-involved/dev-channel

Dealing with ‘Rule Rot’.

- What part(s) of your CSS are being used?
- Open Dev Tools. Switch to the Profiles panel. Select the ‘Collect CSS Selector Profile’ option.
- Click the Start button. The circle will go red to indicate recording has started.

About profile recordings

- As the site is browsed each selector is counted and measured.

- Click around whilst Chrome records to invoke every style.

- Activate buttons and states. When all pages are viewed, click the stop/record button.

- If you screw up. Just select the profile on the left and click ‘Clear all profiles’ button. Then start again.

- Once you have a profile recorded, select it over on the left:

Profile results table

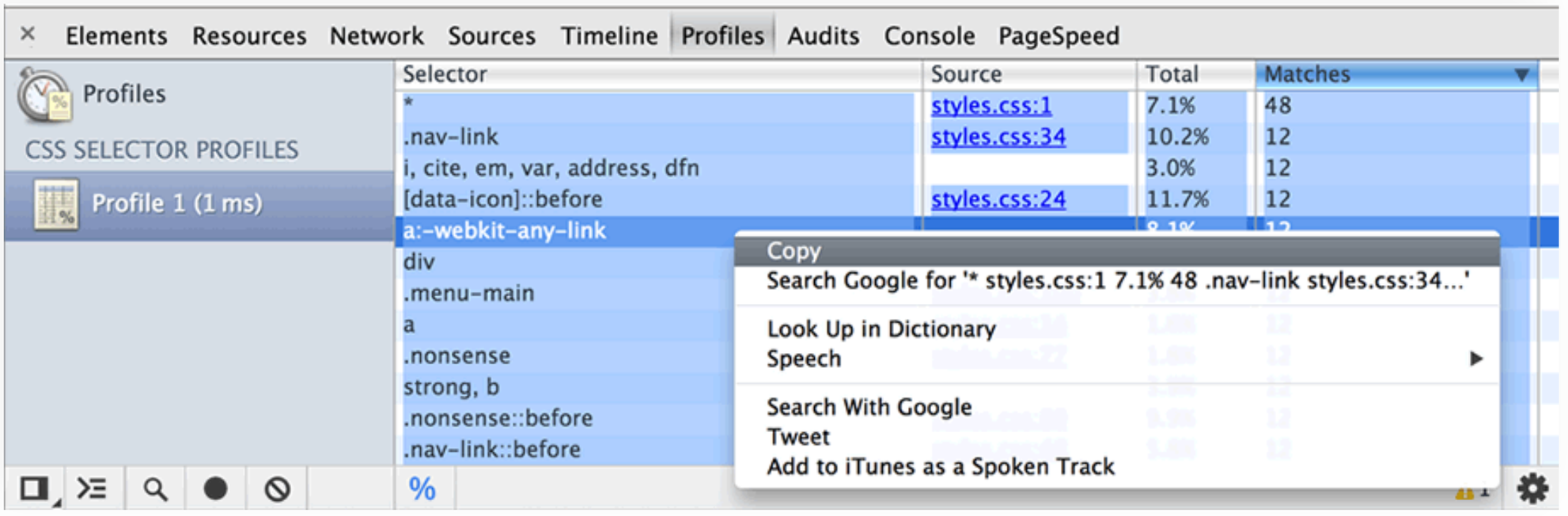
You'll be presented by a four column table. The columns:

- Selector – the CSS selector in question
- Source – where the rule originated (handy if multiple stylesheets)
- Total – how long did it take, in percentage terms to select
- Matches – how many times was the selector matched

What are we looking for?

Interpreting the Profile data

We're looking for low-hanging fruit; rules in our stylesheets that aren't used.

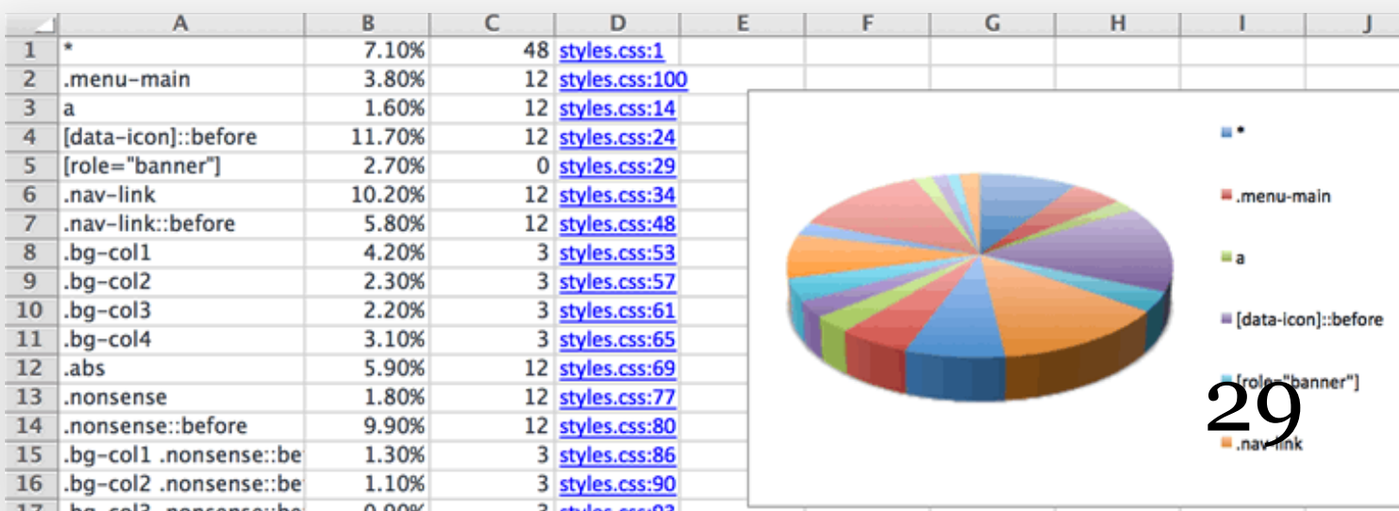


Don't forget this information is only based upon styles that were used (or not) whilst this profile was being recorded

Pro tip: Export to Excel

It’s easy to get the profile data into Excel

- Select the contents (e.g. command+A on the Mac) and copy.
- Paste contents into Excel.
- Have nothing better to do? Make a chart or one of those pivot things I don’t understand!



	A	B	C	D
1	*	7.10%	48	styles.css:1
2	.menu-main	3.80%	12	styles.css:100
3	a	1.60%	12	styles.css:14
4	[data-icon]::before	11.70%	12	styles.css:24
5	[role="banner"]	2.70%	0	styles.css:29
6	.nav-link	10.20%	12	styles.css:34
7	.nav-link::before	5.80%	12	styles.css:48
8	.bg-col1	4.20%	3	styles.css:53
9	.bg-col2	2.30%	3	styles.css:57
10	.bg-col3	2.20%	3	styles.css:61
11	.bg-col4	3.10%	3	styles.css:65
12	.abs	5.90%	12	styles.css:69
13	.nonsense	1.80%	12	styles.css:77
14	.nonsense::before	9.90%	12	styles.css:80
15	.bg-col1 .nonsense::be	1.30%	3	styles.css:86
16	.bg-col2 .nonsense::be	1.10%	3	styles.css:90

Testing the performance of your styles

- Long held beliefs regarding performance
- Whilst many hold true, they may be of negligible importance on *YOUR* site.
- Use these tools to replace conjecture with data.
- Make choices based on empirical facts, not generalisations.
- For trouble-shooting CSS performance, my favourite two features are ‘Show paint rectangles’ and ‘Enable continuous page repainting’.

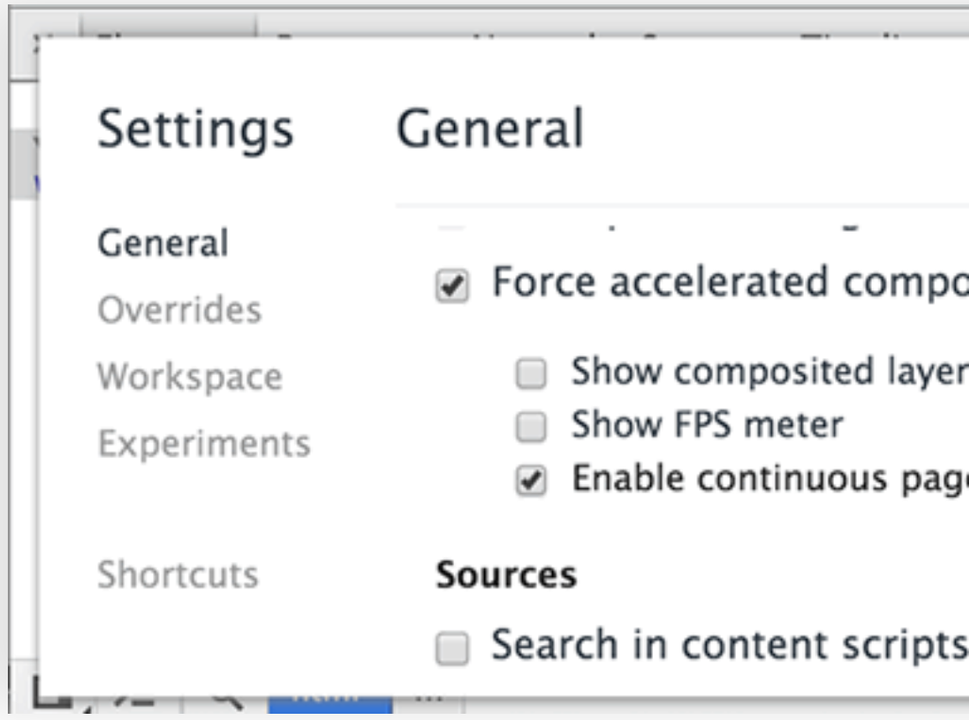
Continuous page repainting

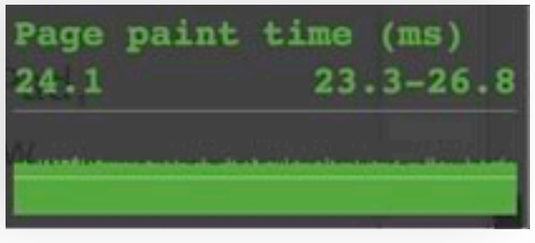
- Click the cog (bottom right). Under the General section, enable ‘Continuous page repainting’.

- The small graph (top right), shows paint time of the current viewport (in ms).

- The rolling graph shows previous paints. The line indicates a 16ms threshold. Something on desktop that has a paint time of 16ms+ could struggle on lower powered devices (e.g. mobile).

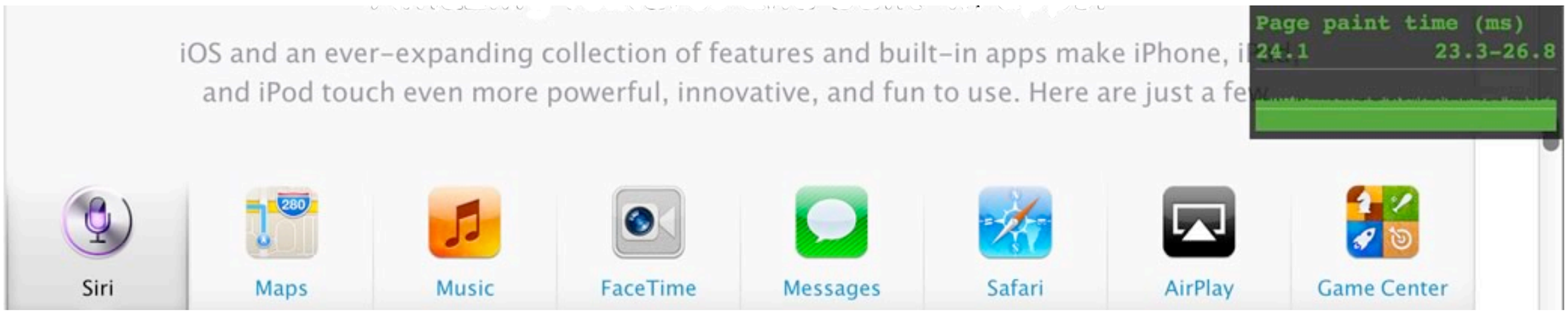
- The aim is to reduce the paint time.





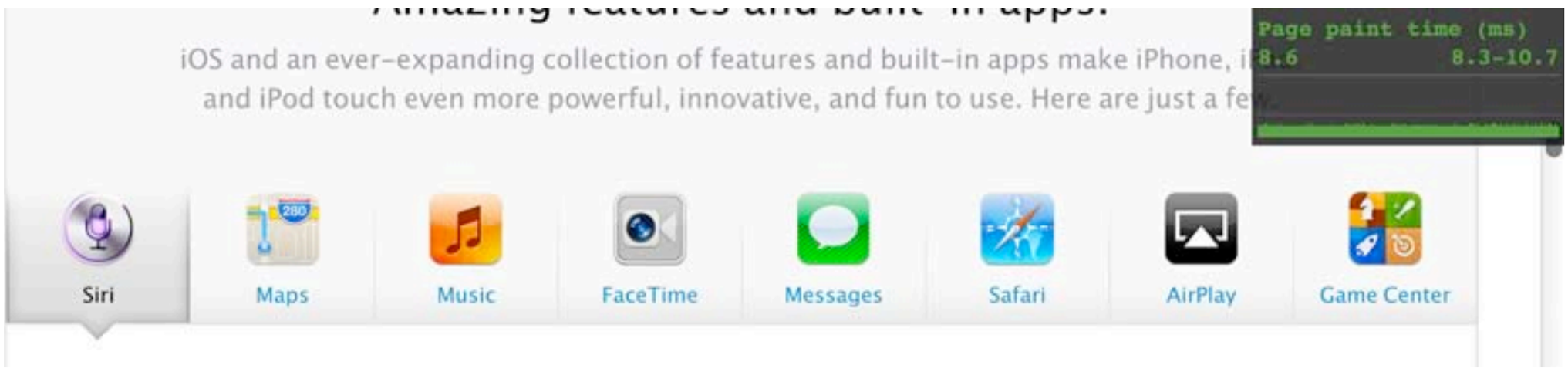
What causes expensive paint times?

- ‘Expensive’ styles can come from surprising places.
- Luckily, Dev Tools make it easy to toggle styles in the styles panel, or visually hide an element by pressing ‘H’.
- Example: http://www.apple.com/iphone/ios/ – scroll down to this section (24ms paint time):



Expensive paint times (continued)

- That element has a 24ms paint time. What's causing it?
- Images? Fancy text? Nope, a CSS inset box-shadow.
- Removing the box-shadow makes little difference to the visuals but an enormous difference to the paint time (down to 8.6ms).



Continuous page repainting conclusion

- Continuous page repainting let's you easily find problem styles. Real data on which to make performance decisions for your own site

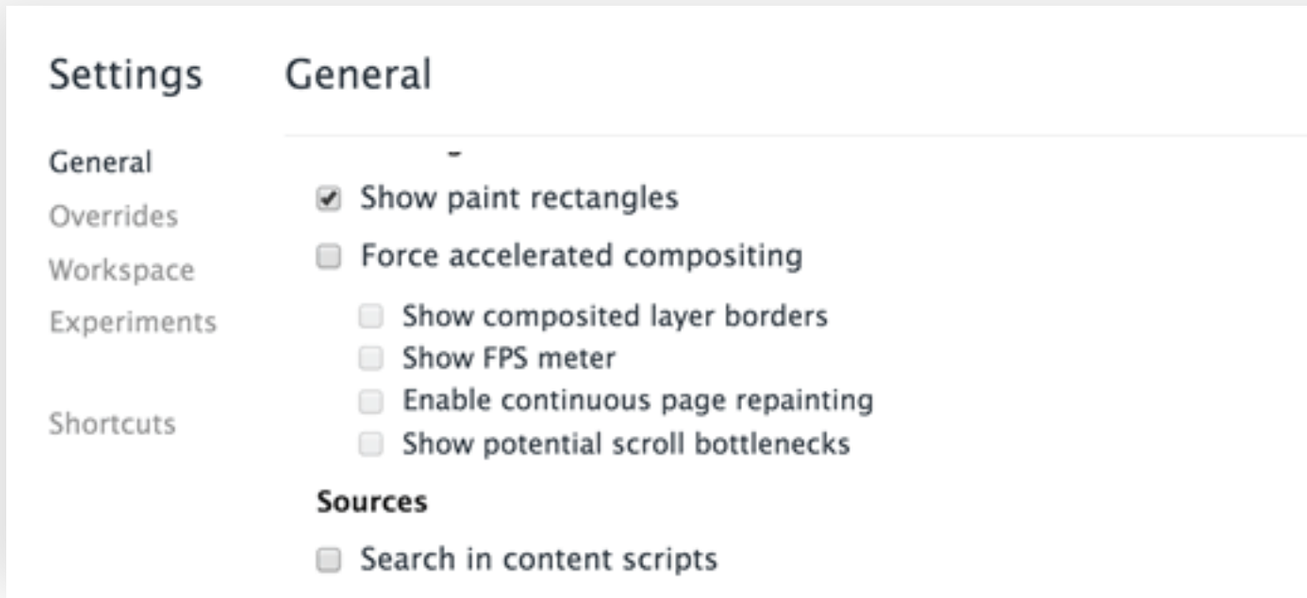
- **Note**: selectors, technically, have zero effect on paint time. Therefore when looking at paint time, consider properties and values, not selectors (plus selectors are very fast anyway).

In modern browsers, I'd argue that in many cases performance is within the braces, architecture is outside:

```
.architecture-here {
    performance: here;
}
```

Show paint rectangles

- ‘Show paint rectangles’ displays the areas of a page being ‘repainted’ as you interact with it.
- Here is how you enable it:
- ‘Painting’ means what you expect. As the browser interprets the data (e.g. HTML, CSS, JS) it composites the design to the screen as a series of ‘tiles’.



- If nothing changes, there are likely few page repaints. That means better performance.

Paint rectangles (example)

- Let's look at an example (sorry Apple): http://store.apple.com/us
- Red rectangles flash in certain areas as the page is scrolled.
- These are the areas being repainted by Chrome.

Paint rectangles (cont.)

- Switch off properties to see how it effects painting.
- In our example, the paint on scroll was caused by a hover state.
- Perhaps not a problem in itself but this also occurs as the page is scrolled (when perhaps those hover states are unneeded).
- Very common scenario and with easy to implement workarounds.
- A great post on paint times: http://www.html5rocks.com/en/tutorials/speed/unnecessary-paints/

Resource bloat and asset encoding

- Find out what assets (scripts, stylesheets, images) a page is loading.
- Details how many separate requests (number of files downloaded) and their size.
- Let's have a look at another Apple page: http://www.apple.com/ipad/
- Switch to the Network panel and refresh.
- There's a long list of assets. Scroll to the bottom and you'll see the totals in a grey bar. In this instance: *133 requests | 4.4 MB transferred | 5.06 s (load: 1.79 s, DOMContentLoaded: 1.37 s).*

Concatenate CSS and convert assets to Data URIs

- Use the column headers to sort the assets
- ‘Type’ allows us to sort by CSS files.
- Could files be combined to minimise the number of requests (for style sheets, if using Sass this is obviously trivial)?
- Could smaller image assets be converted into data URIs (use Compass alongside Sass for easy conversion) to further minimise requests?
- Thinking of Data URIs for mobile? This is worth a read: http://www.mobify.com/blog/data-uris-are-slow-on-mobile/

Compression of assets

- Are assets being served ‘gzipped’?
- Command/right-click the headers area and tick the ‘Content-Encoding’ option.
- This column indicates which assets were sent gzipped.
- If none, it’s time to look at how you can make that happen.
- Most servers use Apache. The HTML5 Boilerplate project has a great .htaccess file that typically makes gzipping happen automagically: https://raw.github.com/h5bp/html5-boilerplate/master/.htaccess

Dev Tools for CSS performance summary

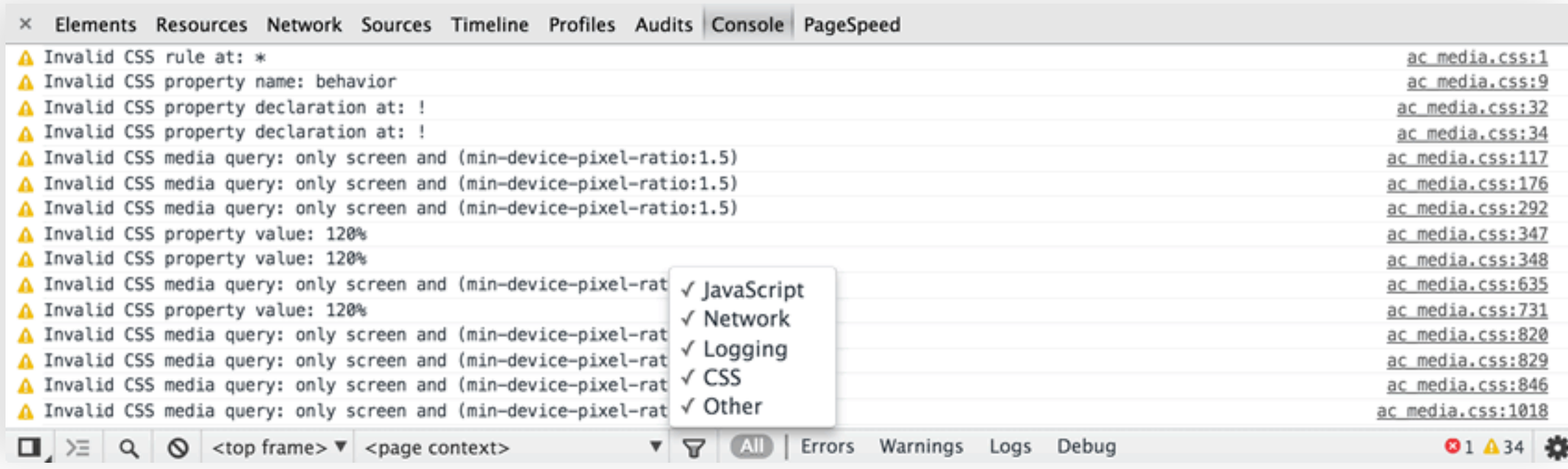
- If you've never used the Dev Tools to look at the performance of your CSS/Sass, hopefully this has given you some ideas.

- OK, so that's CSS performance, what about CSS quality?

- Chrome can help here too and we'll also take a look at CSS Lint, a (sometimes maligned) tool that can quality check (to a degree) your CSS.

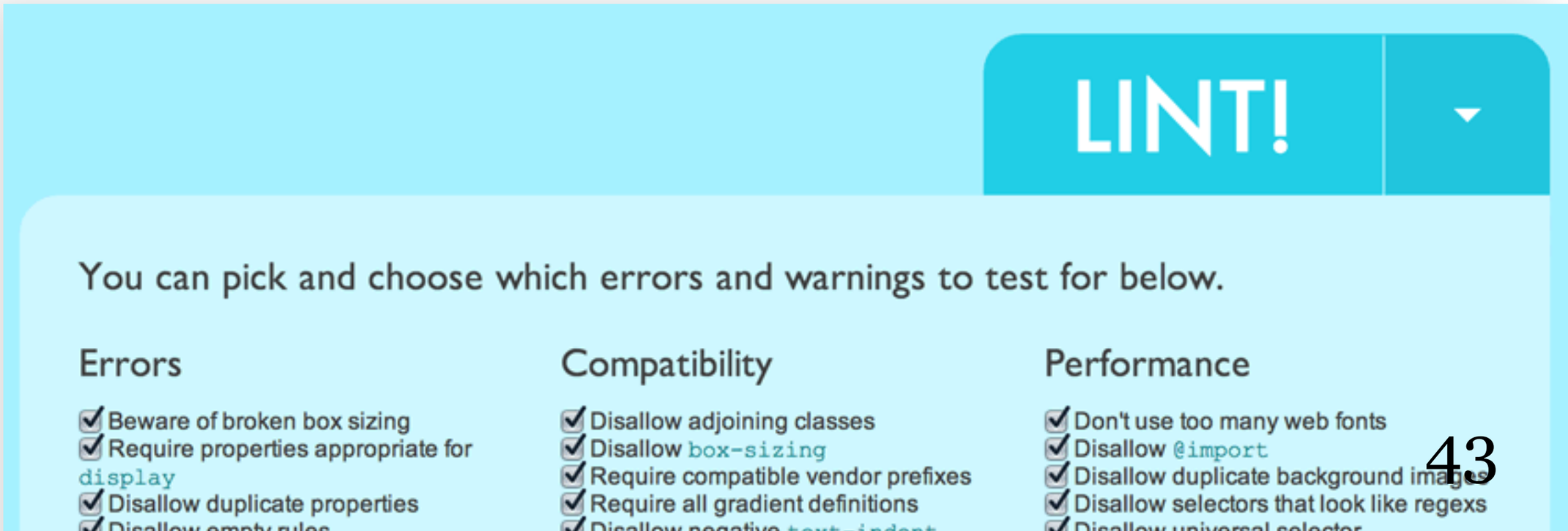
CSS code quality checks with Chrome Dev Tools

- Switch to the Console tab.
- CSS errors and warnings are shown for the page.
- If you just want to see CSS errors, use the Filter button:



CSS Lint (nothing to do with Chrome)

- CSS Lint automates CSS quality against a number of pre-defined tests.
- It's a configurable tool that can flag up common CSS authoring faux pas.
- Paste your CSS into the big empty box and then click the big down arrow to configure (or get a plugin for your editor e.g. https://github.com/SublimeLinter/SublimeLinter):



CSS Lint configuration

- Remember it's configurable. Decide what you would like checking.

- Typical examples: it can spot an un-needed float alongside `display: inline-block;` (which has no affect).

- Conversely I don't give a monkeys about 'Avoiding un-anchored hovers' (as it generally only affects IE7 badly).

- Think of it as a final sanity check before pushing CSS code live.

Chrome Dev Tools Voodoo (saving back to source)

- Chrome can save amendments made in the Developer Tools back to the source Sass/CSS files.

- This prevents round-trips to the editor and saves lots of time.

- Command-click a property or value and the relevant file opens in the Sources tab. Edit the value and command+S saves the file - changes are reloaded straight into the browser.

- Set-up is a little involved so interested parties should head here: http://benfra.in/1z1 for a walkthrough.

Conclusion

- We've covered a LOT of ground. Thanks for staying with me.
- Remember, a list of useful videos/posts on using the Chrome Dev Tools here: https://gist.github.com/benfrain/5908652

Thank you for your attention.

Any questions?