Technology
Arts Sciences
TH Köln

Introducing Constraints into Web Layouts:

Evaluating the Intuitiveness of Current Approaches for Designers

MASTER'S THESIS

by

William Clear, 11101906

submitted to obtain the degree of

MASTER OF SCIENCE (M.SC.)

at

TH KÖLN – UNIVERSITY OF APPLIED SCIENCES

INSTITUTE OF INFORMATICS

Course of Studies

WEB SCIENCE

First supervisor: Prof. Dr. Kristian Fischer
TH Köln - University of Applied Sciences

Second supervisor: Prof. Christian Noss
TH Köln - University of Applied Sciences

Cologne, July 2016

Contact Details: William Clear
william.j.clear@gmail.com

Prof. Dr. Kristian Fischer
TH Köln – University of Applied Sciences
Institute of Informatics
Steinmüllerallee 1
51643 Gummersbach
kristian.fischer@th-koeln.de

Prof. Christian Noss
TH Köln – University of Applied Sciences
Institute of Informatics
Steinmüllerallee 1
51643 Gummersbach
christian.noss@th-koeln.de

Abstract

When it comes to web applications and their dynamic content, one seemingly common trouble area is that of layouts. Frequently, web designers resort to frameworks or JavaScript-based solutions to achieve various layouts where the capabilities of Cascading Style Sheets (CSS) fall short. Although the World Wide Web Consortium (W3C) is attempting to address the demand for more robust and concise layout solutions to handle dynamic content with the recent and upcoming specifications, a generic approach to creating layouts using constraint syntax has been proposed and implementations have been created. Yet, the introduction of constraint syntax would change the CSS paradigm in a fundamental way, demanding further analysis to determine the viability of its inclusion in core web standards. This thesis focuses on one particular aspect of the introduction of constraint syntax: how intuitive constraint syntax will be for designers. To this end, an experiment is performed involving participants thinking aloud while reading code snippets. Also, cursor movements are recorded as a proxy for eye movement over the code snippets. The results indicate that, upon first-impression, constraint syntax within CSS is not intuitive for designers.

Abbreviations

CSP	Constraint Satisfaction Problem
CCSS	Constraint Cascading Style Sheets
CSS	Cascading Style Sheets
Flexbox	CSS Flexible Box Layout Module Level 1
GSS	Grid Style Sheets
HLSA	Hue, Saturation, Lightness and Alpha
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
RGB	Red-Green-Blue (Colour Model)
sRGB	Standard Red Green Blue (Colour Space)
VFL	Visual Format Language
W3C	World Wide Web Consortium

Table of Contents

1 Introduction

This thesis looks at the question of whether introducing constraint syntax into Cascading Style Sheets (CSS) results in an intuitive development experience for designers creating layouts for the web. Phrased as a research question:

> Is constraint syntax in CSS intuitive for designers, relative to current CSS layout techniques?

The paper attempts to answer this question as well as propose and implement a process for determining the relative intuitiveness of CSS language features in general. The focus on whether the technology is intuitive stems from some of the principles that underlie the development and the success of the Web so far; in particular: availability and collective empowerment (Open Stand, 2016). This paper suggests that considering the intuitiveness of the standards underpinning the Web works towards aligning the Web with these principles, as more intuitive standards would allow more people to utilise the Web to a greater extent.

This is one aspect of the practical relevance of this thesis. The other is establishing a process of determining the intuitiveness of CSS language features which might be used to challenge or support decisions in the creation of standards; a process that seems to have been largely based on opinion during the formative years of CSS (discussed in section 2.3, supported by Appendix A). In particular, this paper examines the feasibility of the integration of constraint syntax as defined by the Grid Style Sheets 2.0 (GSS) framework (GSS, 2015) into the CSS collection of specifications by comparing how intuitive it is for designers compared to current layout approaches. Clearly, there is more to consider in the adoption of a standard; however, this paper limits its scope to a consideration of the relative intuitiveness of the technologies.

There are several baseline concepts to establish for this paper: web layouts, web designers, constraints, current approaches to web layouts, intuitiveness and the web design process. Each of these concepts is defined and described later in the introduction as it is understood in the context of this paper. Further, the introduction outlines the scope of the thesis and introduces the objectives, relevance and the motivation for writing the paper. Following the introduction, related works are considered: this includes both academic work as well as state-of-the-art material in the web layout space. Next, the analysis section considers the problem space from a historical perspective. The analysis section helps relate the paper to its context as well at looking at how some fundamental ways of thinking about web layouts arose. An experiment section follows, describing and justifying the experiment design, presenting the results and discussing the findings. The paper is summarised, key findings are highlighted and a critical review is offered in the conclusion.

1.1 Core Concepts

To establishing the ground work for this thesis, web layouts, web designers, constraints, current approaches to web layouts, intuitiveness and the design process and environment are defined and described as they are understood in the context of this paper.

1.1.1 Web Layouts

A web layout, as understood in this paper, is the composition of the graphical and textual elements on a web page or in a web application. This is based on definition of layout as a general term (Macmillan Dictionary, 2016) and made specific for the web environment. In other words, the web layout could also be seen as the compositional result produced by the interpretation of a website's source code by a web browser. (For brevity, the word "layout" also refers to web layouts in this paper). For instance, the Holy Grail Layout is a web layout utilising a full-width header at the top of the page, a three-columned content section and a full-width footer at the bottom of the page (Levine, 2006). Columns, rows, grids, headers, footers and navigation bars are common terms used in the description of web layouts. Furthermore, many websites utilise a grid-based layout framework such as Twitter Bootstrap (Bootstrap)[1], ZURB Foundation (Foundation)[2] or Ink[3] (discussed in detail in section 2.3). The grid pattern of arranging content in rows and columns is evidently a de-facto standard for web layouts. While the layout is the intended result, this paper focuses on how it is achieved by a web designer using CSS.

1.1.2 Web Designer

Although the title of the paper refers to 'designers', by placing the word within the context of 'web layouts' it is intended to imply that web designers are the target subject. In this paper, a web designer is a person whose profession involves producing and updating web layouts and interactions with web technologies (primarily CSS, HTML and JavaScript). The use of web technologies differentiates the term *web designer* from the more established term, *graphic designer*. However, since graphic design generally involves conveying information using design elements such as typography and images (Merriam-Webster.com, 2016), a web designer is seen as a particular type of graphic designer. More specifically, the web designer (subsequently referred to as 'designer' for brevity) role includes creating designs (which could be sketches, computer-generated imagery or mock ups) and translating those designs into code for browsers (Grannell, 2013); in other words, it encompasses the work of a

[1] The homepage of the Twitter Bootstrap project is http://getbootstrap.com/.
[2] The homepage of the ZURB Foundation project is http://foundation.zurb.com/.
[3] The homepage of the Ink project is http://ink.sapo.pt/.

```
#feature-video[width] <= 800;
#feature-video[left] == ::window[left];
#feature-video[top] == ::window[top];
#feature-video[width] <= ::window[width];
#feature-video[width] == #feature-video[height] * 1.7778;
```

Figure 1: Constraint syntax example stating an element should retain a given width-to-height ratio and be no wider than the width of the browser window.

frontend developer. This is not an unprecedented inclusion, as designer Andrew Clarke observed: "code became my medium when designing became more than about making an artist's impression of a website" (Clarke, 2014). This is also backed up by survey done by Gridset in which 38% of respondents, the largest category, self-identified as being a hybrid designer/developer (Gridset, 2014). Both support the idea that being a designer involves writing CSS. Therefore, in the context of this paper, a designer implements web layouts using CSS.

1.1.3 Constraints

The term "constraint" is taken from the constraint satisfaction problems area of research in artificial intelligence. A constraint satisfaction problem (CSP) consists of "a set of variables, each of which has a value… [it] is solved when each variable has a value that satisfies all the constraints on the variable" (Russell & Norvig, 2010, p. 202). A constraint consists of a relation defining the values that variables participating in the constraint can take on (Russell & Norvig, 2010, p. 202). In particular, this paper looks at how layouts may be treated as CSPs using GSS. At face value, it seems quite beneficial to consider a layout as a CSP; for instance, one may constrain two elements to have equal heights and widths (perhaps desirable when attempting to utilise the Gestalt principle of similarity (Lidwell, Holden, & Butler, 2010)), or constrain the width of an element to be always be proportional to its height (useful for maintaining a ratio on video elements and image elements with preferred aspect ratios), or constrain the centres of two elements to be at the same coordinates on the page, ensuring that one element is always neatly centred, vertically and horizontally, within the other element. In each of these examples, the constraints are satisfied when the requirements are fulfilled.

To best explain how constraints can be used for web layouts, an example is offered. Take the example of ensuring that a video element always has a particular ratio and always has a width at least as small as the window, for which the code may look like that given in Figure 1. In this example, a feature video element is being identified by the GSS selector `#feature-video`, which functions virtually the same as a CSS selector for most intents and purposes. The width of the feature video element is a variable participating in three constraints. On the first line, it is constrained to always be less than or equal to 800 pixels. In the language of

```
#feature-video[width] == #feature-video[height] * 1.7778;
#feature-video[height] == #feature-video[width] * 0.5625;
#feature-video[height] * 1.7778 == #feature-video[width];
#feature-video[width] * 0.5625 == #feature-video[height];
```

Figure 2: Alternatives, equivalent ways to prescribe an element a 16:9 ratio using constraint syntax.

CSPs, the scope of the constraint consists solely of the width property of the feature video element and "`<= 800`" dictates the relation for the constraint: the value of the width property should either be less than or equal to 800 pixels. Next, on line 4, both the width of the feature video element as well as the window width are participating as variables in a constraint. In this constraint, the relation is between the variables: it states that the width of the feature video element should always be less than or equal to the width of the window. The third constraint featuring the width property of the feature video element on line 5 has a relation defining that the width of the video element should be 1.7778 times the height of the width of the video. In other words, this line is enforcing a size ratio of 1.7778:1 (width to height) on the video element, reflecting the common 16x9 video aspect ratio.

Throughout this description, the phrase "should be" has been used to reflect the particular class of CSP that GSS is solving, a CSP with constraint weights. This allows authors to prioritise constraints: weaker constraints may be violated when it is not possible to implement them in favour of satisfying stronger constraints (GSS, 2015). The strength of constraints in GSS is given with the keywords: weak, medium, strong and required; where medium is the default strength (GSS, 2015). In regards to the intuitiveness of the constraint hierarchy concept in GSS, the use of natural language keywords may help designers quickly adopt the concept of the constraint hierarchy in GSS. Further, the GSS constraint hierarchy is analogous to two concepts in CSS: the cascade and the important rule. Analogous concepts or metaphors help make a system intuitive (Blackwell, 2006) (this is discussed greater detail in the analysis). By providing more "weight" to styles specified later and by allowing CSS rules to override subsequent declarations with the `!important` rule (W3C, 2011), CSS has established an analogous precedent for the constraint hierarchy concept in GSS. Despite the similarities and potential footholds for designers attempting to use GSS, it remains to be seen whether the constraint hierarchy concept will be found intuitive or not.

Indeed, GSS presents quite a paradigm shift from CSS as it stands, where a property value is assigned to a property in a unidirectional fashion. A constraint, on the other hand, works in two directions. Here, a similar question of intuitiveness arises in the consideration of this malleable aspect of constraint syntax semantics. To highlight the bidirectional nature of constraint syntax and that the constraint syntax is not simply assigning variables, it is

instructive to consider some of the alternative ways in which the last line of the feature video example could be have written. These alternatives are shown in Figure 2, each of these lines produces an identical result: the video element maintains a size ratio of 1.7778:1. Relating this back to intuitiveness, the paradigm-shift from reading the <property>: <property value> assignment pattern in CSS to bidirectional constraint syntax in the form <property> <relation> <property> may impose a cognitive hurdle for designers new to GSS.

In summary, it can be seen that treating web layout as a CSP and declaring a layout using GSS can be useful for solving common web layout challenges. However, significant differences between GSS and CSS raise the question about whether it will be considered intuitive by designers and consequently readily adopted.

1.1.4 Current Approaches to Web Layouts

The 'current approaches' part of the title refers to techniques that designers are currently using to accomplish various page layouts. A current approach could refer to a specific layout concept or it may refer to how an entire website has its layouts templated. For example, the use of syntax from the CSS Flexible Box Layout Module Level 1 (Flexbox) specification to create a splash page, the use of floats to position sidebars, and the use of frameworks such as Bootstrap to create page templates based on grids are all current approaches to web layouts. It is treated as a broad concept; however, this simply reflects the variety of web layouts and the flexibility of its definition. The current approaches are looked at in more detail in the Related Work section (see sections 2.3 and 2.5 in particular).

1.1.5 Intuitiveness

Intuition in this paper is defined as a thought or response that is "reached with little apparent effort, and typically without conscious awareness, [involving] little or no conscious deliberation" (Hogarth, 2001). In other words, in the mind of the thinker a conclusion or understanding might be reached with minimal conscious thought process. This paper considers the intuitiveness of designers in particular. This implies familiarity with the basic design concepts and the syntax of CSS. For example, CSS allows a user to define the background colour of an element like so:

```
footer { background-color: blue; }
```

From this snippet, it is assumed that a designer could readily understand that a footer element will have a blue background with very little thought. This rapid comprehension is considered intuitive. In this case, it is perhaps intuitive because it so closely resembles an equivalent natural language representation of the same idea that a "footer's background colour is blue."

On the other hand, not all CSS expressions are intuitive. For example, consider the declaration block:

```
img + span[data-traits~="blob"]:nth-child(4) { color: #6495ed; }
```

In this snippet, the reader of the CSS must consciously resolve that `span` elements being styled by this rule are preceded by an image element and they contain the value "blob" among other values of the `data-trait` attribute. Further, a small amount of counting is involved: it is the fourth child within its parent container. Lastly, unless the reader is particularly well versed with colour theory and hexadecimal notation, it is hard to visualise the colour being used as a background colour in this CSS snippet as it is given by the sRGB colour space and the intensities of the red, blue and green values must be combined to determine the resultant colour, cornflowerblue[4] (W3C, 2011). Counter-intuitive traits of CSS need not be found in complex selectors or colour spaces: more specifically to layouts, some CSS authors may find it counter-intuitive that the `height` and `width` properties do not include the size of the borders or padding (unless the `box-sizing` property is set to `border-box`) and the complex rules dictating margin-collapsing behaviour are known to be confusing as well (Hickson, 2004). Therefore, the intuitiveness of CSS is in no-way guaranteed. However, these counter-intuitive aspects to CSS highlight the importance of making sure new additions to the specifications favour intuitive syntax to minimise the introduction of additional points of confusion in the future.

Generally, the use of familiar keywords and the straight-forward way that CSS presents property values mapped to properties within declaration blocks with relatively little syntax seems to make CSS fairly intuitive by design. This is explored in more detail throughout the paper. However, there are aspects of the language, both advanced (such as advanced selectors) and simple (such as colour values and sizing with padding and borders) that can quickly turn the activities of reading or writing CSS into puzzle solving. This paper looks at what is required to accomplish "intuitiveness" as given in the sense of the first example in this section (`background-color: blue`); that is, the characteristics of the CSS language that result in an intuitive authoring experience, especially when creating layouts.

It is recognised that, over time, a web designer develops a sense of intuition for how to use CSS, at least at a basic level. While teaching the use of Cascading Style Sheets (CSS) for layouts, Rachel Andrew, author of several CSS related books, has observed that there is a point where those learning a particular layout technique with CSS "just get it" and it "becomes simple for them" (Simmons & Andrew, Laying Out the Future with Rachel

[4] One of the CSS colour keywords, w3.org/wiki/CSS/Properties/color/keywords.

Andrew, 2016). Comparably, it was found in a study of the intuition of mathematicians that knowledge and experience seemed to be the primary contributors to the development of intuition (Burton, 1999). Although research on the intuition of web developers is hard to come by, it is perhaps reasonable to draw parallels and assume that knowledge and experience play a major role in developing an intuition for the syntax designers are working with and the effects that it causes. Therefore, an ingredient for intuitiveness, as it is considered in this paper, is prior experience with CSS and familiarity with the basic concepts and syntax. Although the search for intuitive technology is driven here by principles of accessibility and collective empowerment, it would seem unreasonable to expect that intuitiveness could be built-in without requiring some learning effort to establish a baseline understanding of how CSS works.

1.1.6 The Design Process and Environment

The environment in which the designer is working impacts how the designer creates and maintains code sources while producing their designs. For instance, considering just the step of producing CSS, it is common to debug and tweak CSS using developer tools found in popular web browsers. These tools often include features such as: syntax highlighting, automatic indentation of code, colour-preview boxes and colour selectors, an ability to view all the styles affecting an element, file management and real-time page updates as styles are changed. Therefore, in considering how intuitive layout technologies are in real world usage, such assistance needs to be accounted for. This means that although a CSS concept may be difficult to understand while reading it from specification or implementing it in a plain text editor without syntax highlighting, it may still be termed intuitive if common tools exist to make working with such a concept easier. For example, composing RGB colours may not be intuitive because it may require consciously mixing the hexadecimal values to approximate the colour being generated, as noted earlier. However, the click-and-drag colour dialogs present in popular browsers such as Chrome and Firefox allow designers to intuitively work with colours as they can point and click among hue, saturation, lightness and alpha values with little thought as to which colour will be produced. Therefore, the design process and environment are included as core concepts for their effect on whether aspects of CSS can be termed intuitive or not.

1.2 Scope

This thesis limits its scope in several ways due to the limited timeframe of several months and virtually no budget. This restricts the scope for the size of the experiment and the breadth of the topics being covered.

Necessarily, this thesis is intended only as a preliminary usability test of constraint syntax in CSS; therefore, only a handful of participants are expected to participant in the experiment. However, as noted by Jakob Nielsen, testing usability with just 5 users may yield a significant proportion of the results (Nielsen, 2000). On the other hand, the same article proposes making many small tests; whereas, the experiment conducted as part of this thesis tests just one layout due to the limited timeframe. Experimentation on a scale required to term the results indicative for all web designers of all cultures, let alone conclusive, is well beyond the resources available and remains beyond the scope for this thesis. Rather, the scope for the experiment is generate suggestive findings and a process that could be repeated if more research in this area is warranted.

Further, there are many ways to evaluate artificial languages including characteristics such as expressiveness, definiteness, implementability and so on (Khedker, 1997). The scope of this thesis is restricted to looking especially at whether CSS is intuitive to read. In other words, it is looking at the question: if a designer was to see a code snippet used to generate a layout, could the designer quickly comprehend what the approximate outcome would be? As stated in the introduction, the outcome of test proposed is intended to support or challenge decisions about language features rather than provide a holistic view of them.

In the same way that not all criteria of good programming languages are being observed, not all capabilities are being included either. This paper distinguishes between two major applications of CSS: defining layouts and styling elements. While styling elements may include defining colours, typography and animation, layout refers only to the positioning and sizing of elements. Although the two are related and may overlap, the focus in this paper is on CSS features relevant to constructing web page layouts. Even more specifically, it is looking particularly at the layout capabilities of GSS and comparable capabilities in CSS.

The GSS framework also comes with additional capabilities that are not considered within the scope of this paper. GSS also provides a Visual Format Language (VFL) and element-based conditionals (e.g. if the width of element x is greater than y, then implement z). However, this paper focuses on the core concept of constraints and does not consider the intuitiveness of these extensions.

Phrasing the scope positively: the thesis is restricted to considering only the intuitiveness constraint syntax as implemented by GSS relative to CSS layouts techniques on a small scale.

1.3 Objectives and Relevance

This thesis aims to achieve two objectives:

- Produce a preliminary conclusion as to whether designers find constraint syntax for layouts in CSS (as defined by GSS) intuitive relative to other CSS approaches.
- Secondarily, test a method for evaluating the intuitiveness of CSS languages constructs that may be repeated and improved upon to test the intuitiveness of other language constructs.

As for the relevance: as covered in the Related Work section, there is a lot of material being generated on new layout techniques, including upcoming CSS specifications. An in-depth look at the solutions being proposed and how they compare from the perspective of a designer could be relevant to many working in the field. Secondly, deciding on whether CSS language features are intuitive or not seems to be largely a matter of opinion (as observed in section 2.3), this paper presents groundwork that could potentially be used to determine whether or not language constructs really could be considered intuitive or not based on evidence instead of opinion. This may also be of relevance to those involved in the construction of programming languages.

1.4 Motivation

With a background as a web developer, working in both server-side and frontend development, common CSS, JavaScript and HTML concepts quickly became ‘normal’ and unquestioned. This thesis provides an opportunity to ask: where did some of these concepts come from? Are there alternatives? Could the process of evolving the Web itself be improved?

Furthermore, exposure to a variety of working environments has led to an appreciation of the transferability of skills related to the core standards of JavaScript, HTML and CSS, even as tools, platforms and frameworks change. Therefore, a Master’s Thesis covering a topic that would necessarily involve a greater understanding of the standards and an outlook on their future is exactly the sort of thesis I am interested in writing.

It also seems to be the right context to write such a thesis. It seems that the Web has moved from a collection of documents, through a dynamic web and a web of data, to also being an interface to intelligent systems and even an intelligent system in its own right in some senses. While scoping for topics, I began working through the textbook *Artificial Intelligence: A Modern Approach* (Russell & Norvig, 2010) to discover more potential ways to use the Web intelligently. I came across constraint satisfaction problems in chapter 6 and, at around the same time, GSS. At this time, this was a personal insight: realising that not only could the Web enable interactions with intelligent systems, the web platform itself could become more intelligent. This is very exciting, because a more intelligent web platform means the benefit

is available to all of over one billion[5] websites rather than one or a few proprietary, black-box intelligent systems. On this scale even small, incremental improvements can have a profound impact.

[5] The estimated number of websites at the time of writing is 1,040,100,236, 14 June 2016, from http://www.internetlivestats.com/total-number-of-websites/.

2 Related Work

The related works are constituted of two significant areas. The first is other academic work that looks at the origin of the constraints in CSS concept and determining how the mind works while reading, looking especially at what constructs in code might be deemed intuitive and how to measure this. The second area considers approaches to developing technologies that support web layouts and how certain layout patterns are achieved. This includes the extent to which the CSS working group considers the intuitiveness of the standards as they are developing based on evidence from the public mailing list, how frameworks handle layouts and modern layout patterns. Additionally, in the last few years the CSS Working Group has been attempting to address the layout challenges of web developers with specifications such as the CSS Flexible Box Layout Module Level 1 (shortened to Flexbox) and, the CSS Grid Layout Module Level 1 (CSS Grid), this are also addressed as related work.

2.1 Constraint Cascading Style Sheets

In 1999, Constraint Cascading Style Sheets (CCSS) was proposed in the proceedings of the 12th annual ACM symposium on User Interface Software and Technology (Badros, Borning, Marriott, & Stuckey, 1999). CCSS is foundational for this thesis as it provided a detailed description of how constraints could be used by CSS. Further, the GSS project is openly based on CCSS concepts: an author of the original CCSS paper, Badros, is involved in the GSS project as well (GSS, 2015). Given its importance to this paper, the CCSS paper is summarised here, and key concepts are highlighted.

The CCSS paper begins by introducing the state of CSS and notes several areas for improvement, including: responding to various browser window sizes and 'ad hoc' layout restrictions (Badros, Borning, Marriott, & Stuckey, 1999). Both of these issues are addressed in this thesis as they seem to have been unresolved in universally accepted way since the original paper was written over fifteen years ago, paving the way for GSS other frameworks to gain popularity by addressing some these shortcomings. Other issues noted at the time such as a complex and vague CSS specification and inconsistent browser support are not considered within the scope of this thesis. Naturally, introducing constraints into CSS was viewed as the solution to these issues. It was noted that with CCSS 'we[6] can naturally and declaratively specify complex behaviour' (Badros, Borning, Marriott, & Stuckey, 1999, p.

[6] Interestingly, from the analysis of the CSS Working Group's public mailing list (see Appendix A) the term 'we' may not have been used in such a paper if it were written again in today's context. Over time, the contributors to CSS development have developed an awareness that they are creating syntax for web authors rather than themselves; therefore, a term like 'authors' or 'designers' may have been used instead. Distinguishing this subtle difference is important because the audience of CSS readers must be considered when determining whether or not it is intuitive.

```
(1) #t[width] = #c1[width] + #c2[width] + #c3[width]      REQUIRED
(2) #c1[width] ≥ width(\Text1")                           REQUIRED
(3) #c2[width] ≥ width(\Text2")                           REQUIRED
(4) #c3[width] ≥ width(\Text3")                           REQUIRED
(5) #c3[width] ≥ #i2[width]                               REQUIRED
(6) #c1[width] + #c2[width] ≥ #i1[width]                  REQUIRED
(7) #t[width] = 0                                         WEAK
(8) #c1[width] = 0.3 * #t[width]                          DESIGNER
(9) #c3[width] = 0.2 * #t[width]                          DESIGNER
```

Figure 3: Example layout constraints. This figure is based on Figure 6 from the CCSS paper (Badros, Borning, Marriott, & Stuckey, 1999, p. 77).

73). Indeed, the ability to 'naturally and declaratively specifying complex behaviour' is indeed the subject matter of this thesis itself (if one may consider 'naturally' as a being synonymous with 'intuitive' as it is understood in the context of this paper). Interestingly, the layout-related parts of the CCSS introduction seem to be as relevant in today's context as they were when the paper was written over a decade ago.

Following this introduction, the CCSS paper introduces the functionality of CSS at the time in more detail and reiterates the key issues given above before introducing how constraints would work with CSS. Conveniently, the first aspect considered is page layouts, and how it 'can be modelled using linear arithmetic constraints' (Badros, Borning, Marriott, & Stuckey, 1999, p. 76). A simple table layout is recreated using CCSS, the syntax governing the implementation is given in Figure 3. This example was included to illustrate some key concepts beyond the CCSS syntax itself: browsers automatically creating constraints, the impact of the strengths of constraints and how the syntax suggests an implementation strategy (Badros, Borning, Marriott, & Stuckey, 1999, p. 77).

In the description of the code snippet, it was noted that constraints 1 through 7 are automatically generated by the browser to implement certain table properties: the table width should be as wide as its columns (constraint 1), each column should be as wide as the text it holds (constraints 2 through 6) and the table should try to minimise its width to 0 (constraint 7). The given strength of `REQUIRED` present for constraints 1 through 6 ensures that the table and its columns show visible content, while the `WEAK` constraint that the table width should be 0 means that the browser should attempt to meet this constraint by minimising the table width. In other words, the `WEAK` constraint can be violated in favour of meeting the `REQUIRED` constraints and browser instead gives the table a width as close to 0 as possible while satisfying the other constraints. The `DESIGNER` constraint strength present on constraints 8 and 9 is intended to represent rules created by web page authors. This strength is more strictly enforced than `WEAK` constraints but less strictly enforced than `REQUIRED` constraints (Badros,

Borning, Marriott, & Stuckey, 1999, p. 77). In the given example, the `DESIGNER`-strength constraints produce a table where column `#c1` is 0.3 times the width of the table, column `#c3` is 0.2 times the width of the table and column `#c2` is left to take up the remainder of the table, provided that more strictly enforced constraints do not break these rules. Altogether, the example offered a glimpse into a flexible layout system showing the viability of constraints syntax for web layouts.

However, the article did not maintain a thread arguing why such syntax would be found intuitive (or 'natural' to use its own terminology) by designers despite introducing it as such and positioning it as an improvement over the difficult-to-understand CSS 2.0 specification with 'seemingly ad-hoc restrictions on layout specification' (Badros, Borning, Marriott, & Stuckey, 1999, p. 73). Indirectly, the declarative nature of the syntax was emphasised as well as the way it implied an implementation strategy (Badros, Borning, Marriott, & Stuckey, 1999, p. 77). However, when it came to testing the proposed syntax, an Amaya browser extension was tested for functionality and performance and underlying constraint-solving algorithms were discussed without having designers read and write CCSS to test its usability.

In effect, the assumption seems to have been made that constraint syntax offered a more intuitive solution to layout problems and the ground work for how the constraint syntax could work alongside CSS was established. GSS completed a full-scale implementation of the syntax (with modifications) and this paper offers evidence as to the supposed intuitiveness of constraint syntax.

2.2 Online Processing During Reading and How Code is Read

Rather than simply look at what designers report as being intuitive, this paper makes an effort to reason with concepts from psychology as to why some programming language concepts may be seen as more intuitive than others. In doing so, the relationship between reading and information processing is examined. Although it is not specific to reading code, the work of Keith Rayner is particular useful in this area, therefore an overview of it is included here. Other research, more specific to code reading is included as well, although it does not connect reading to cognitive processes to the same extent.

In 1998, Rayner offered a review, *Eye Movements in Reading and Information Processing: 20 Years of Research*, stating that eye tracking and analysis technology as well as theories of language processing had sufficiently advanced that it was "possible to use eye movement records for a critical examination of cognitive processes underlying reading" (Rayner, 1998, p. 372). While introducing an overview of the results of such progress, the article first establishes the common terminology of the field.

The terms saccade and fixation are introduced here to establish the groundwork of the rest of the overview. A *saccade* is an eye movement from one position to another. The time that the eye remains stationary between the saccades is termed a *fixation*. The duration of fixations is about 200 – 300ms, while the duration of saccades depended on degree to which the eye moves: a common saccade while reading of 2° is about 30 milliseconds whereas a saccade of 5° is about 40 to 50 milliseconds (Rayner, 1998, p. 373). The timings and introduced here because, as will be seen, the timing of eye movements can reflect cognitive processes.

The article is quick to introduce the cognitive processes behind the movements. The article notes several studies that suggest 'saccade programming is done in parallel with comprehension processing in reading' and points out that decisions of when and where to move the eyes are separate decision processes (Rayner, 1998, p. 374). Importantly, evidence is found that 'cognitive processes can influence the latency' of saccades. For example, voluntarily directing saccades away from a peripheral target increased saccade latency, directions to be careful resulted in increased latency and increase latency resulting in increased accuracy of eye movements. (Rayner, 1998, p. 374). Further, a centre of gravity effect was identified whereby a saccade directed to targets of two elements landed in an intermediate location, with a pull effect toward larger or brighter elements (Rayner, 1998, p. 374). Further on, it is identified that about 10 to 15% of saccades are regressions: the eyes traverse back over text that has already been passed. The explanation given for regressions is that they occur because the 'reader did not understand the text' (Rayner, 1998, p. 375). Furthermore, in-word regressions may be due to problems processing the currently fixated word (Rayner, 1998, p. 375). Clearly, there are inferences about cognitive processes during reading that can be made by tracking the eye movements of the reader. For this paper, this means there is a way to measure what really could be termed intuitive on a relatively fundamental level, beyond merely asking designers what they find easy to understand.

The impact of these findings on reading code is applied tentatively here, in lieu of more specific work being done in this area. Firstly, the trend appears to be that increase cognitive processing slows down eye movements during reading. Therefore, comparing two code snippets, the one with longer fixations may be the harder to

understand of the two or the less intuitive code snippet in terminology of this paper. This is an interpretation that code reading studies have also adopted (Bednarik & Tukiainen, 2006). Secondly, a relatively large number of multi-word regressions may indicate difficultly understanding what the CSS achieves while a relatively large number of in-word regressions may indicate difficulty understanding the semantics of keywords in the code.

There seems to be much less research material looking at what reading patterns imply about cognitive processes during the reading of code. Instead, research looking at code reading seems to be focused on how code is read and subsequent comprehension as opposed to what can be inferred about cognitive processes while reading, especially at a novice level (Busjahn, Schulte, & Busjahn, Analysis of Code Reading to gain more Insight in Program Comprehension, 2011; Whalley, et al., 2006; Busjahn, et al., 2015; Turner, Falcone, Sharif, & Lazar, 2014). Even less can be said for research material looking at cognitive process while reading declarative languages such as CSS for which no relevant papers were found during the construction of this paper. In some ways, it could be said that code reading research appears to be at the same level of natural language reading research prior to the 1970s: identifying the patterns and comprehension on a surface level (Rayner, 1998, p. 372), yet to look deeply into the online processing taking place as code is read and extrapolate cognitive processes. Perhaps this identifies the need for a model of code comprehension that could be verified, tested and improved in the same way that models like the E-Z Reader model have been instrumental in establishing a framework for research of eye-movement during reading (Reichle, Rayner, & Pollatsek, 2000). It must be noted that there are models for program comprehension (for example, a comparison between six is given for software maintenance and evolution (von Mayrhauser & Vans, 1995)), although none seem to predict eye movements. Nevertheless, some findings from research looking at code reading are applicable here.

In particular, it is interesting to know which programming language features seem to support the reading of code. For instance, recent research utilising eye-tracking suggests that *regularity*, repetition of code fragments, provides a better indicator of the understandability of code than metrics like lines of code and McCabe's

cyclomatic complexity (Jbara & Feitelson, 2015). This could be interpreted as giving CSS an advantage over GSS, as GSS introduces additional syntax, it may dilute the regularity which is typical for CSS. Further, another eye tracking experiment involving code reading showed that more experienced programmers utilise 'surface language features that facilitate … comprehension', also known as *beacons*, to navigate and understand programs; although, exactly which language feature acts as a beacon may differ from person to person (Crosby, Scholtz, & Wiedenbeck, 2014). One might assume that selectors appearing above each declaration block could represent beacons in CSS, a pattern shared by GSS. However, as shown in the introduction, GSS also utilises syntax outside of declaration block, this may obscure beacons and, again, result in CSS being the easier of the two to comprehend. Once again, the findings from the research must be applied tentatively here, as although the research focuses on code, it focuses on programming languages with an executable structure as opposed to declarative languages.

In summary, particular traits can be looked for from code readers to infer cognitive processing: particularly long fixations and regressions may imply additional reasoning steps and misunderstanding or unfamiliarity respectively. Furthermore, it has been noted that certain programming language features seem to facilitate comprehension, although the extent to which these findings apply to declarative languages like CSS and GSS is unclear. Therefore, when determining the intuitiveness of a GSS relative to CSS, evidence from reading patterns over code snippets and the identification of assistive language features via a think-aloud experiment component and reflection on the code would be beneficial in arguing the case one way or the other.

2.3 Approaches toward Intuitiveness from the CSS Working Group

So far, research has been identified that allows us to reason (to a limited extent) about the intuitiveness of CSS and GSS based on reading patterns and language features. It may be instructive to consider how intuitiveness has been reasoned about during the creation of CSS thus far. To this end, the group developing the CSS syntax, the CSS Working Group, maintains a public mailing list. According to the CSS Working Group Charter, this mailing list is the primary tool for technical discussion and is involved in obtaining consensus for decisions (Bos & Lilley, 2016). Therefore, it can be used as a related work to gather insight into the approach towards creating an intuitive layout authoring experience. Indeed, an exact

match for the term ‘intuitive’ is found among approximately 1% of the mailing list’s communications (see Appendix A) indicating it certainly a consideration of the group.

In order to draw out the approach toward making CSS intuitive, a basic, textual analysis of the public mailing list during the formative years of CSS was conducted. The mailing list archive interface (W3C, 2016) was used to search for the keyword ‘intuitive,’ filtering for the mailing list name: ‘www-style.’ Each of the results was opened in a browser window. Using the hotkey Ctrl + F and searching for the term ‘intuitive’ highlighted its usages on the page. The context of each usage was read and understood. The context may have been a single sentence, such as this example from 2006:

> I can (kind of) see their reasoning for having floats for alpha, since integers aren't *intuitive* for opacity. (Raymond, 2006)

Or, the context may have required several sentences to understand, as in this extract from a 2008 email:

> Thirdly, I understand from [1] that clearance was originally implemented as a change in margin-top. Superficially this seems *intuitive*, so there must be some tricky edge-cases which expose problems with this implementation. (Prowse, 2008)

In any case, the context of the usage of the term ‘intuitive’ was pasted into the table given in Appendix A, along with the year it was used and a link to source. In a fourth column of the same table, a categorisation was given as the nature of the usage of the term ‘intuitive.’ The four categorisations were:

- *Opinion*: the claim of intuitiveness was led by language that suggested it was an opinion; for instance: “I think…”, “I believe…”, “It seems to me…” or so on. Or, insufficient logic or evidence was presented to support the claim of intuitiveness.
- *Logic*: the claim of intuitiveness was backed up with a logical argument.
- *Evidence*: the claim of intuitiveness was backed up by some research or test.
- *Irrelevant*: the usage of the term *intuitive* did not actually relate to some aspect of CSS being intuitive.

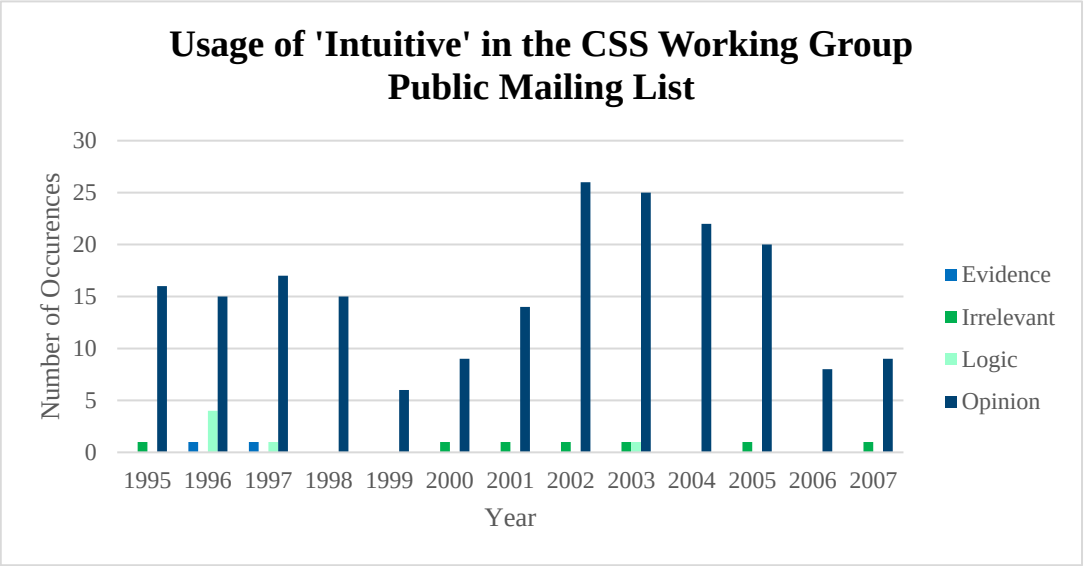


Figure 4: How the use of 'intuitive' is supported in emails belonging to the CSS Working Group's public mailing list over the years.

As shown in Figure 4, declaring some aspect of CSS as intuitive or counter-intuitive in the majority of cases seemed to be based on the opinion of the mailing list contributor and rarely on usability studies analysing CSS concepts or detailed reasoning to justify why a particular concept is intuitive or not. It is perhaps instructive to highlight the cases where a concept was declared intuitive based on some underlying reason, as this may illustrate why it is not more often the case. For instance, a loose reference was made to usability studies when discussing the intuitiveness of colour notations:

> …in particular, it has nothing to do with HLS, HSB and suchlike polar representations of RGB (which are, in usability studies, often shown to be *not* very *intuitive*) (Lilley, Re: CNS colors, 1996).

In another email discussing a potential default setting of 0 for the volume of an element read by a speech synthesiser, an analogy is drawn:

> This seems strongly counter-*intuitive*. The default is that there is no sound? Perhaps a stylesheet for visual presentation could specify that the default is black text on a black background, so the screen is entirely dark? (Lilley, Re: T.E.O.'s Draft--Cascading Speech Style Sheets (txt), 1996).

In these cases, the justification for terming something as intuitive or not had analogous parallels. When discussing colour notation, a general reference was made to usability studies that had looked at the topic and when discussion default element volume, an analogy was made to the established default background and text colours. Figure 4 indicates logic and evidence were used to a greater extent when justifying the intuitiveness of CSS concepts in

the early years of its development when it was drawing from existing concepts, for example in print media (as will be seen in 3.1 Historical Perspective). As CSS developed, perhaps a larger number of unique and novel aspects had to be considered for which there were no obvious analogies to draw from.

Summarising the results, it seems as though the intuitiveness of the CSS standards is certainly a recurring a discussion point in the mailing list. This reflects efforts made to produce syntax that is indeed intuitive for CSS authors. However, the efforts toward this end are largely informal. In the majority of cases where intuitiveness is discussed, the writer asserts some aspect is intuitive based on their own opinion rather than research or substantial reasoning. It is understandable that this is the case, as mentioned: concepts are being formulated for a new domain and styling for the Web has its idiosyncrasies, limiting the material it is able to borrow from existing fields. Perhaps this represents an opportunity to increase efforts in formally conducting usability testing on CSS itself.

The basic methodology used here to analyse the CSS Working Group's consideration of intuitiveness could itself be improved. For instance, having a second, independent reviewer verify the results would add weight to the results. Further, the methodology could be improved by looking further outside of the public mailing lists into how the CSS Working Group considers the usability of CSS as it develops. It should be noted that there exists a private, member-only mailing list which may have further information unavailable for this analysis. Lastly, due to time restrictions, to analysis covers the years 1996 through 2008 which may obfuscate any more recent efforts from the CSS Working Group in this area. Altogether, the results can only be taken as indicative rather than absolute and there may in fact be additional deliberations over how intuitive CSS languages features are before they are introduced for newer features; although the precedent of basing decisions about intuitiveness on opinion is certainly evident.

In summary, the results from this basic analysis are indicative that intuitiveness is persistently considered albeit informally. In defence of the CSS Working Group, producing standards that are intuitive is not among the success criteria given in the charter; on the other hand, one would assume it contributes towards other success criteria such as 'achieving multiple, independent, interoperable implementations,' since implementations are made by humans who will have their intuitions when interpreting the standards, and having 'user community and industry adoption' (Bos & Lilley, 2016). Perhaps the methodology testing the relative intuitiveness of GSS to CSS could be extended to test the introduction of other new CSS language features, so future decisions in this area are based on evidence. There is certainly a

significant test effort made to measure CSS's compatibility with browsers (W3C, 2016), perhaps a similar effort could be made to test CSS's compatibility with designers.

2.4 Frameworks

The history of using hacks such as table-based layouts or using floats and images for purposes other than those for which they were designed in order to achieve certain layouts has led to a demand for a better solution to web layouts. So far, this demand from designers seems to have been somewhat fulfilled by the use of frameworks such as Foundation and Bootstrap (Shepherd, 2011; Kramer, 2014). Despite the utility of such Frameworks, this paper assumes that the movement toward a generic layout component offered as part of CSS is a desirable outcome: for the performance benefit (the browser internally calculates the layout without relying on interpreting and executing external JavaScript), the ability for designers to transfer layout skills from project to project independent of whether projects use particular frameworks and to reduce network traffic as less framework code needs to be downloaded. (To this end, even though GSS is a framework, the concept considered in this paper is the inclusion of constraint syntax in CSS). Nevertheless, the popularity of using frameworks to organise the layout of a website demands that they be investigated in this paper. However, the CSS modules will be considered in more depth than the frameworks and included in experimentation while frameworks are not.

In recent years, frameworks seem to have played a major role in influencing the layout of numerous websites. As an indication, Wappalyzer, a tool that identifies website software, has detected Twitter Bootstrap on over three million websites, Ink on over seven-hundred thousand and ZURB Foundation on over two-hundred thousand via its users in the last 6 months (Wappalyzer, 2016). All three frameworks are based on grid systems. Bootstrap and Foundation offer a 12 column grid layout whereas Ink determines its columns in percentages: multiples of 5% up to 100%, 16%, 33%, and 66% (Bootstrap, 2016; ZURB Foundation, 2016; Ink, 2016). Further, all three base their grid system on the usage of floats, floating columns to the left by default. Ink and Foundation have alternative builds available that utilise the `flex` display value and Bootstrap has included it in its upcoming version 4 release as a replacement for using floats. This indicates a transitory stage from using float to using flexbox as the cornerstone for website layouts; however, it seems that using floats is still the default approach underpinning grid systems on websites. Furthermore, it is a strong indication that grid arrangements are the defacto standard for web page layouts.

In summary, the implementation of a grid system is a commonality between the frameworks; essentially adding a semantic laying for expressing layout in columns and rows although the underlying implementation is typically achieved with floats. Although it may be one of many

factors underpinning the popularity of the frameworks, this suggests that designers are looking for grid semantics to assist producing web layouts.

2.5 Modern Web Page Layouts

Here, the current state of layouts on the web is considered. Trends are identified and connected to intuitiveness. Although academic material does cover web layouts, it has proved difficult to uncover material discussing state-of-art CSS layout techniques. Perhaps this is due to the rapid, transformational progress made on the web platform and how it is used. Therefore, academic efforts have been made to capture the zeitgeist of the web such as trends in element positioning (Kumar, et al., 2013), analyse the visual complexity of web page layout (Harper, Jay, Michailidou, & Quan, 2013) or determine the impact of CSS updates across a website (Liang, et al., 2013). In other words, the general focus appears to be on the outcome of using CSS as opposed to specific techniques used to create layouts. Therefore, the current state of layout techniques has been identified by analysing the leading voices in web design, such as Jen Simmons, Rachel Andrew and Eric Meyer, who are known for interactions with W3C, browser vendors and web designers. Identifying the concepts that leading voices are promoting has been more revealing than academic literature in the process of determining the current state of techniques for implementing web layouts.

In particular, a recent talk from Jen Simmons entitled “Modern Layouts: Getting Out of Our Ruts,” summarises prevalent layout patterns and suggests how upcoming expansions to the CSS specifications, such as those analysed in this thesis, will impact the future of web page layout (Simmons, 2015). The ‘rut’ that Simmons’ refers to is shown in Figure 5. It is a common, familiar web page layout featuring a full-width heading at the top followed by navigation, content on the left and a sidebar on the right underneath the navigation and full-

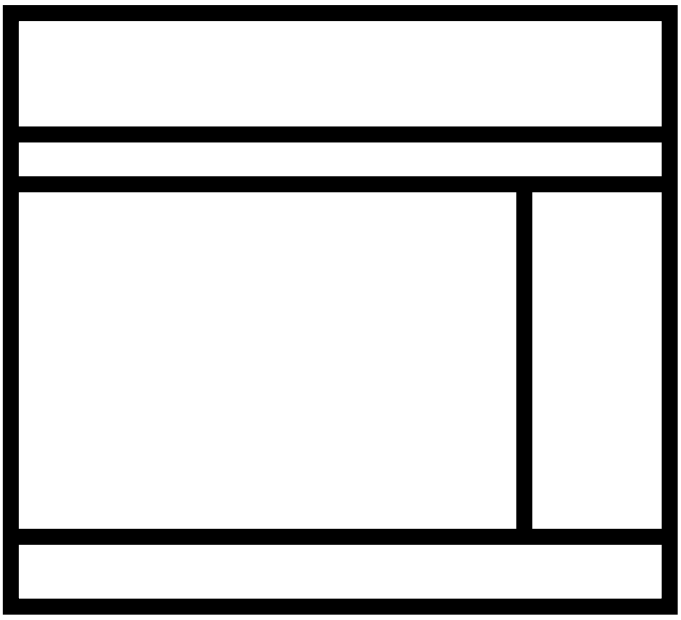


Figure 5: Simmons' asserted common layout pattern for web pages (Simmons, Modern Layouts: Getting Out of Our Ruts by Jen Simmons, 2015).

width footer at the bottom. Simmons offers twenty example websites from a range of organisations in various industries to indicate the prevalence of this pattern.

Looking at print seems to offer one way out of the 'rut'; although Simmons cautions that the ideas must "translated not transferred" (Simmons, 2015). In other words, layouts as they are being done in print cannot simply be copied for the web, the concepts behind them must be understood as well as characteristics unique to the web medium. Similarly, layouts should not be copied without sense from one project to another; instead, "layout should serve the content" notes Simmons (2015). An example of an online article from the New York Times with a full-width, full-height splash cover, like what one would see in a physical magazine, is offered; with the caveat that is suitable for a cover story, not for every blog post. Like splash covers based on Flexbox, shaping content blocks with interesting outlines using tools from CSS Shapes (W3C, 2014) is another suggested technique based on concepts from print. Many solutions offered are responses to a predominant "box, box, box way of thinking" (Simmons, 2015). However, even when using boxes, especially in form of grids, Simmons suggests ideas for using them to support the content. While three, horizontally-aligned, equally-sized boxes of content may be 'orderly and sturdy' (but perhaps overused and commonplace), three boxes growing in accordance with the golden ratio may seem 'organic and dynamic' and three boxes with chaotic sizing is 'interesting and a bit unnerving,' says Simmons quoting an article from Nathan Ford (Ford, 2014). The ability to experiment with managing columns and rows using concepts from the upcoming CSS Grid specification is

also encouraged (Simmons, 2015). The techniques offered challenge designers to break out of the prevalent, familiar layout pattern shown in Figure 5.

The impact of this commentary can also be considered in the context of the discussion taking place in this paper as well. It identifies the boilerplate effect of the prevalent frameworks and design patterns and the resultant similarity between websites. As noted in the introduction, experience is a factor when developing an intuition for layouts, and it seems that experience is increasingly constituted of the usage of template and framework code which establishes layout paradigms like the 12-column grid (see section 2.4 Frameworks). Many of the proposed solutions to the 'rut' are in the form of new W3C specifications, especially CSS Grid, Flexbox and CSS Shapes. Yet during the talk, Simmons notes that 'it's going to take us all a couple of years to wrap our heads completely around Flexbox' and proposes the Gridset as a third-party tool to create a grid layout and generate a framework (Gridset, 2016). On one hand, this may suggest that tools to assist designers in creating layouts are necessary to embed design principles and this component of the work may never be intuitively embedded into CSS language constructs. On the other hand, it may suggest that the standards enabling the creation of modern web page layouts are not yet intuitive and therefore learning effort and supportive tools are required to utilise them effectively. This would make room for something like GSS which enables layout creation in a way supposedly easier to comprehend relative to the construction of layouts based on what is currently available in CSS.

It is clear from the work covering the construction of modern web page layouts that the developments of the CSS specifications in this area must also be considered alongside any other proposal for an intuitive solution to web page layouts. Towards the end, Simmons notes that the "hardest part is changing our thinking, not our CSS" (Simmons, 2015). However, CSS is also an artefact constructed by humans and, in a way, this thesis is about determining if we can change CSS to suit our thinking.

2.6 CSS Specifications

As concluded in the previous section, considering recent developments in CSS specifications related to web layouts is necessary when proposing constraint syntax for CSS. Interestingly, these specifications have, to a degree, already exposed designers to constraints indirectly. CSS Tables (part of CSS2.1), CSS Grid and Flexbox describe implementation details of certain language features in way reminiscent of constraints.

A neat comparison to start with is that of CSS Tables, following the trend established by the CCSS paper (see section 2.1 Constraint Cascading Style Sheets) which featured an example

```
/* GSS Based on CSS Tables Specification, Snippet 1 */
td[left] == ::window[left] !weak;

/* GSS Based on CSS Tables Specification, Snippet 2 */
table {
    td {
        left: >= (^ tr:last)[left];
        right: <= (^ tr:last)[right];
        bottom: <= (^ tr:last)[bottom];
    }
}
```

Figure 6: Constraints given using GSS syntax that could be used to implement aspects of the CSS Tables specification.

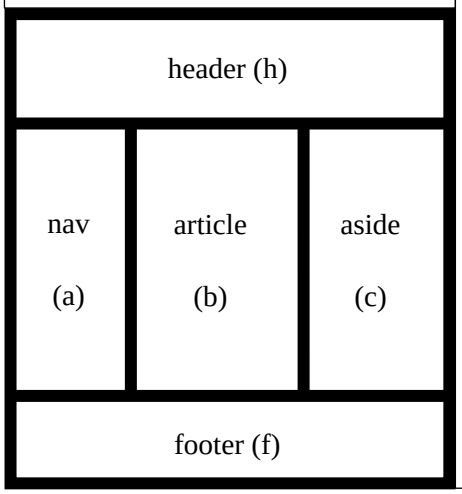
showing how constraints could be used to render a HTML table (Badros, Borning, Marriott, & Stuckey, 1999). Similarly, the CSS Table specification shows how constraints lie just under the surface of CSS syntax. In the section entitled '17.5 Visual layout of table contents' the CSS Table specification it is noted that rectangles representing table cells must be 'as far to the left as possible' (W3C, 2011), prompting one to think of a constraint such as that shown in the first code snippet in Figure 6. Here, GSS syntax is used to declare a weak constraint that the left edge of table cells should try to be at the left edge of the window. While other constraints may push table cells to the right, the above constraint would 'exert a force', so to speak, always pulling cells as far to the left as possible as required by the CSS Tables specification (swapped for right-to-left languages). Likewise, the statement that 'a cell box cannot extend beyond the last row box of a table' (W3C, 2011) may be achieved by the constraints given in the second code snippet of Figure 6. The selector `^ tr:last` selects the last table row of the table by using `^` to refer to parent selector, `table`, of the nested `td` declaration block. Subsequently, the left, right and bottom edges of table cells within a table are constrained to never be beyond the left, right or bottom edges of the table respectively. Evidently, aspects of the CSS Tables specification lend themselves to be defined in constraint syntax.

Such constraints become assumptions for designers working with these specifications. Given the CSS Table specification, designers may implement layouts using CSS Tables (not HTML Tables) assuming that cells are rendered as far to the left as possible (in left-to-right languages) and never lie beyond the last table row. Integrating constraint syntax into CSS would reverse this pattern: a designer could determine how they would like a layout to behave and write constraints to define it rather than being on dependent on the CSS Working Group implementing a collection of properties and property values that would implement the constraints as envisaged by the designer. This is a strong indicator that constraint syntax

could be considered more intuitive than current CSS features: the designer would not have to learn what behind-the-scenes constraints accompany various CSS features, they would be implementing design patterns based directly on how they reason about them. In other words, CSS keywords currently act like a middleman between designers and constraints, giving designers the ability to write constraints would cut out the middleman. Writing constraints themselves, designers may be able to implement layout behaviours in ways they find more intuitive than the keywords provided by CSS.

A situation similar to that of CSS Tables is present in the Flexbox specification: the CSS Tables specification contains the term ‘constrain’ seven times and the Flexbox specification nineteen times. Again, constraints lie beneath the surface. The Flexbox specification is made to include ‘simple and powerful tools for distributing space and aligning content in ways that web apps and complex web pages often need’ (W3C, 2016). However, unlike tables, some Flexbox behaviours can be quite difficult to translate to constraint syntax. For instance, it is unclear what GSS statements would results in flex items filling the available space in proportion to each flex item’s internal content; yet, designers achieve this effect by setting a `flex` property value of `auto` on flex items (W3C, 2016). Further, it is non-obvious how to handle wrapping with constraints in the way that Flexbox achieves. Both these effects may be particularly useful when the designer is uncertain of the length of the content. Despite some Flexbox layouts being difficult to achieve with constraints; other Flexbox layouts are much more straightforward. For instance, a `justify-content` property value of `center` (Flexbox) is reminiscent of constraints aligning elements using the `center-x` property (GSS) while an `align-items` property value of `center` (Flexbox) is reminiscent of constraints aligning elements using the `center-y` property (GSS, 2015; W3C, 2016). This suggests that Flexbox and GSS can be treated like tools: it is up to the designer to pick the tool that suits the intended layout effect.

```
<header>...</header>
<article>...</article>
<nav>...</nav>
<aside>...</aside>
<footer>...</footer>
```




```
main {
    display: grid;
    grid: "h h h"
          "a b c"
          "f f f";
    grid-template-columns: auto 1fr 20%;
}

article {
    grid-position: b;
    min-width: 12em;
}

nav {
    grid-position: a;
}

aside {
    grid-position: c;
    min-width: 12em;
}
```

Figure 7: Example layout based on example from the CSS Grid specification (W3C, 2016).

CSS Grid may also be counted among such tools. CSS Grid is, like Flexbox, offered as a collection of tools to 'control the sizing and positioning of boxes and their contents' (W3C, 2016). Where Flexbox is oriented on a single axis, CSS Grid uses column and row concepts to enable two-dimensional control over layout (W3C, 2016). Using an example borrowed from the CSS Grid specification, Figure 7 shows how the syntax allows for a layout to be templated, then various elements may be assigned to areas within the resultant grid. One may infer how constraints could produce a similar layout. For instance, that `nav`, `article` and `aside` must be the same height or that `aside` is equal to twenty percent of the width of the `main` element (which presumably wraps the other HTML elements). Despite accomplishing the same result, semantic keywords may offer an advantage of CSS Grid over GSS. The `grid`, `grid-template-columns` and `grid-template-rows` properties from CSS Grid concisely template positions and sizes the parts of the grid, while similar semantics may be lost among constraint syntax when accomplishing the same result with GSS.

It is interesting however, that although Flexbox and CSS Grid have been shown to have some advantages over constraint syntax, constraint syntax has been shown to be applicable for layouts produced by all three CSS specifications considered here: CSS Tables, Flexbox and CSS Grid. This would suggest that constraint syntax could be a versatile tool if it were readily available to designers. On the other hand, such versatility may come at the cost of semantics,

as shown in the case of CSS Grid, and this may result in constraint syntax being less intuitive when used for layouts that have built-for-purpose alternatives.

In summary, it seems that designers are already working with and perhaps intuitively reasoning with constraints as they apply to layouts, only they exist under the surface of CSS properties and values. As more specifications like Flexbox and CSS Grid are released, designers have access to a greater range of implied constraints at their disposal. Yet, perhaps enabling designers to write constraints directly would offer a versatile tool which could be used more broadly than any one particular CSS specification. However, this thesis looks at intuitiveness of constraints rather than versatility. The comparison given in this section indicates that simplicity of Flexbox to accomplish advanced layout effects and the semantics offered by the built-for-purpose CSS Grid specification may give these specifications the intuitive edge over defining similar layouts with constraint syntax.

3 Analysis

3.1 Historical Perspective

Looking at the history of the Web provides instructive context to the topic of web layouts. The story of invention of the Web as an information space is well known, but it is worth repeating here with an emphasis on the control over layouts available to web designers as the HTML and CSS standards emerged. These developments allowed a way of thinking about web page layout design to emerge, helping to shape how it is thought about today. As a warning, this section observes the specifications as they were stated at the time of publication, many of the features discussed are now obsolete and non-conforming. However, some of the design ideas have perpetuated and transcended the methods used to create them.

When compared to the print industry which started almost 600 years ago, the 27-year-old Web[7] is still a very recent development. Despite this, it has already had several revolutions in the way that content is presented and laid out. Indeed, the Web borrows many design concepts from the print industry, building on the work done in print media over the centuries, and has subsequently introduced new concepts exploring what is unique to media of the screen. For example, much of the typographical concepts in CSS come from a history in print media. On the other hand, the concept of responsive design, for instance, has arisen for the demands of the Web. However, it took many intermediary steps to go from thinking about design on the Web in terms of print, to thinking about design on the Web as a new medium, or even as designing for multiple media.

At the beginning of the Web, there were not a lot of options for the creator of a website to consider from a visual design perspective. An email from Tim Berners-Lee in 1991 and a linked "HTML Tags" document outlined 19 available HTML tags. A tag for images was not yet present. Further, it was noted that a couple of these tags, such as `<hp1>` for highlighting, were unused (Berners-Lee, 1991; Berners-Lee & Connolly, 1992). In these early stages, the focus was evidently on textual content and the ability to link documents to one another as opposed to flexibility in laying out a webpage.

The first "design revolution" on the Web could be potentially be attributed to the release of the Mosaic browser in 1993. Although it is not technical terminology from the design domain, the intent is clear when Berners-Lee states that the Mosaic browser "made webpages much sexier" due to the use of embedded images in comparison to earlier browsers where images had opened in separate windows (Berners-Lee, 1994-2006). Although it was still hard to

[7] The Web officially celebrated its 25th anniversary in 2014 with the publication of the website: www.webat25.org.

define layouts, the ability to include images directly in web pages was an initial concession of control to web designers and a lot more was to follow in a relatively short period of time.

Two years later, the informal HTML specifications, scattered across a variety of sources, were collected into a single "Hypertext Markup Language – 2.0" (HTML 2.0) document (Berners-Lee & Connolly, 1995). This serves as a coherent and comprehensive primary resource for identifying layout techniques available to web designers[8] at the time. Although overall, the HTML 2.0 specification indicates that a lot of what could be considered web design was left up to the user agent or browser. The first CSS specification was still to come in the following year (Lie & Bos, 1996) and the table element and other developments in HTML were still to come in the year after that with the HTML 3.2 specification (Raggett, 1997); although, some user agents had implementations prior to the recommendations coming from W3C. In the absence of these critical standards, the task of creating layouts for the Web was a difficult undertaking. For instance, it was up to the user agent to select an indentation for the `<pre>` tag, decide how to render various typographic elements, decide whether to append an icon to an anchor tag and decide whether to include an image or the content of its `alt` attribute instead among various other things that directly or indirectly affected the layout of the webpage (Berners-Lee & Connolly, 1995, pp. 24-25,31,33). Despite the restrictions, the HTML 2.0 specification conceded some layout options to web designers.

With the specification, web designers were beginning to see more options to handle the layout of images with hints of other forms of flexibility to come. The specification for the `<img>` tag included an `align` attribute, allowing web designers to align the image to the top, middle or bottom with respect to the text baseline (Berners-Lee & Connolly, 1995, p. 35). Furthermore, the `compact` attribute allowed designers to designate a compact rendering for the list tags `<ol>`, `<li>`, `<menu>` and `<dl>` tags (Berners-Lee & Connolly, 1995, pp. 27-29), granting a small amount of control over the layout of lists. It could also be said that the list elements themselves presented an additional way of controlling layout. Indeed, the standard noted that the ability to mix-in form elements with document structuring elements, such as lists or the preformatted text element, allowed for "considerable flexibility in designing the layout of forms" (Berners-Lee & Connolly, 1995, p. 39). Further, using the image map attribute, `ismap`, on an image element gave designers a way to respond to clicks in different areas of an image; a similar function was available by setting the `type` attribute of an `input` element to "image" (Berners-Lee & Connolly, 1995, pp. 38,42). Since far greater control was

[8] Interestingly, usage of the term "web designer" was also beginning to grow at this time, despite the discussed limitations in design. Prior to 1992, the term "web designer" had been virtually non-existent according to Google's Ngram Viewer: http://bit.ly/28ItCeX.

available over the look of an image, web designers were able to create a layout within an image and handle clicks in various regions of the image accordingly[9]. There was still much to come, although the signs from the initial HTML 2.0 specification, even at this early stage of the web, suggested a move toward granting additional control over layout to creators of webpages; however, a critical issue remained: HTML was not the right place for control over the presentation of content.

Shortly (on a historical timescale) after the inclusion of images embedded directly into webpages by web browsers, came what could be considered the second revolutionary period for web design, especially from a layout perspective, and that is the 1996 release of the first CSS specification (Lie & Bos, 1996) coupled, to a degree, to the release of the HTML 3.2 specification shortly thereafter (Raggett, 1997). This allowed HTML to focus on defining the semantic structure of documents and control over web page presentation was granted to web designers via CSS instead. The rule of thumb that structure and semantics belong in HTML, while presentation (including style and layout) belongs in CSS has persisted since this time. Furthermore, when taken together, it seems that these two specifications introduced a common way of thinking about web layouts: in boxes wrapped around the content of elements, in grids, in two dimensions and on an infinite canvas. By identifying the development of a common way of thinking about web layouts, it becomes possible to argue a case for concepts that could be considered intuitive and concepts that could be counter-intuitive.

The foundations of this common way of thinking are the box model defined by the CSS specification (W3C, 2011) (although it was termed the formatting model (Lie & Bos, 1996) at the time) and the table element which was utilised to implement layouts (a technique now discouraged). The introduction of the box model promoted an understanding of visualising the elements on a webpage as a rectangular box which is surrounded by padding, a border

[9] At the time of writing, an early example of the usage of such an image map is available on the Mosaic Communications Corporation (circa 1994) website: home.mcom.com/MCOM/index2.html.

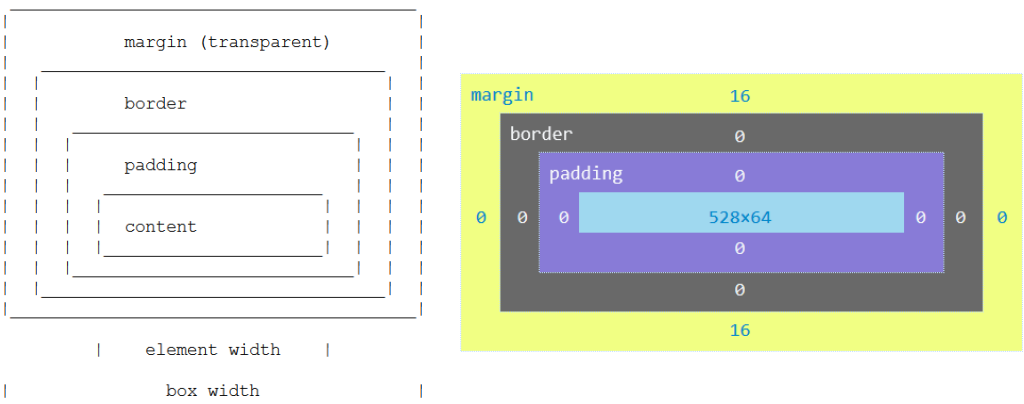


Figure 8: A comparison between the 1996 representation of the box model, taken from the CSS 1 specification (left) and a 2016 representation of the box model as shown in Firefox 48 Developer Tools for a selected element (right). It can be seen that this fundamental concept for web layouts has remained largely unchanged between the two.

and a margin respectively. Simplifying somewhat, the size of the content in each element and the interaction of the margins of the elements forms the layout of the webpage. The fundamental concept has changed little in the past two decades, as can been seen in a comparison of the graphic presenting the box model in the CSS1 standard in 1996 and a graphical representation of the box model from a modern browser in Figure 8. Although the core concept seems simple and has remained stable, the interaction between hundreds of hierarchically-arranged elements to which various display and positioning characteristics apply makes formulating a common standard a substantial undertaking, not to mention a standard that could be called intuitive.

Interestingly, the difference in the treatment of horizontal and vertical margins seems to have been implemented with the intuition of designers in mind, as written in the specification: “after collapsing the vertical margins the result is visually more pleasing and closer to what the designer expects” (Lie & Bos, 1996). Although, there was no obvious justification as to why the designer would expect margin collapsing to behave in this way; instead, it seemed to have been an intuition about intuitions. Nevertheless, not only were these early specifications defining a way of thinking about design for web layouts, they were also attempting to cater to how designers might already be thinking about designing for the Web. A second instance of this occurring, although not for layouts in particular, is that the shorthand syntax for describing the font property is “based on a traditional typographical shorthand notation to set multiple properties related to fonts” (Lie & Bos, 1996). Here, the standard reflects something analogous to what some designers may have already been familiar with. Analogies, along with the use of similes and metaphors seem to be one technique used to create intuitive technologies for layout out web pages.

An analogous concept that would have likely been familiar to web designers encountering the HTML 3.2 specification for the first time is that of the `table` element. Although tables themselves are "a systematic arrangement of data usually in rows and columns for ready reference" (Merriam-Webster.com, 2016), they are conceptually not so far from a grid, "a pattern of straight lines that cross each other to form squares" (Macmillan Dictionary, 2016). Indeed, when tables became standardised with the 3.2 specification, they were used as grids for layout out web pages: a method of dividing the page into columns and rows and placing content in the rectangles produced by this division. Despite, the now-obvious semantic disconnect of using tables as layout grids, it was even a popular method at the time. The bestselling web design book "Creating Killer Web Sites" even promoted it; giving an extended code example whereby a left-hand sidebar effect is created during a redesign of a website[10] (Siegel, 1996). Along with tutorials on the web and prominent websites[11] utilising tables for laying out websites, it quickly became a widespread layout method that persisted, in varying degrees for the next decade.

The point of singling out the table-based layout phenomena of the web in this paper is not to show how much has been learned about the importance of semantic HTML since this period; instead, it is to observe what web designers found intuitive to use, given the technology available. The popularity of the technique and its widespread adoption suggests that web designers were comfortable breaking a layout down into columns and rows, and treating the cells of such a grid as sections thematically separating content.

The move toward structure and semantics and away from presentation is most evident in the recent HTML5 specification. It is highlighted, for instance, in the difference descriptions of the `b` element. Originally, in the HTML 2.0 specification it was described simply as indicating "bold text" (Berners-Lee & Connolly, HTML 2.0, 1995). In the HTML5 specification, no specific styling is mentioned. Instead, the focus has shifted to semantic meaning of text found within a `b` element, as can be seen in this excerpt:

> The `b` element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product

[10] At the time of writing, an example of the table-based layout code is still available at: killersites.com/killerSites/2-sites/stargazer.

[11] For instance, the Apple home page predominantly utilised tables for layouts in 1996: (web.archive.org/web/19961022105458/http://www.apple.com) and it seems the utilisation of the table for layout purposes on the homepage persisted until June 2008, although, by then it was just used for a minor widget (see the news headline ticker on: web.archive.org/web/20070628220543/http://www.apple.com). Similarly, the table element provided much of the layout for the Yahoo homepage in 1997 (web.archive.org/web/19971007020952/http://www9.yahoo.com).

> names in a review, actionable words in interactive text-driven software, or an article lede (W3C, 2014).

Whilst the purpose of using the `b` element is described and accompanying examples given, there is no discussion of how it should be rendered, unlike the early HTML specifications. A similar transition has occurred for the `i` element. Furthermore, other elements which had been more presentation-based, as opposed to semantic-based in function have been dropped completely with no transition. Specifically, in respect to layouts, the `multicol` element from Netscape 3.0 used to achieve a multicolumn layout (Wilson, 2005), the `spacer` introduced also by Netscape to place spacing between elements (Mozilla Developer Network, 2013) and `center` (described in the HTML 3.2 specification as providing a centred horizontal alignment (Raggett, 1997)) elements are now obsolete and non-conforming.

Conclusively, HTML is no longer the place to look for layout options. Its historical impact with the early introduction of tables may have contributed to the common grid-based understanding of webpage layouts; however, the remainder of this paper will focus on CSS and GSS as layout technologies. Yet, HTML remains connected to layout by setting an important precedent: the visual order that a layout technology produces must respect the structure of the HTML document it displays, a principle highlighted in the accessibility section of the CSS Grid Layout specification (W3C, 2016).

4 Experiment

The purpose of this experiment is to identify whether or not constraint syntax within CSS is intuitive for designers attempting to solve common layout challenges. The hypothesis, rephrasing the research question given in the introduction, is that designers find constraints more intuitive to use to solve layout challenges than existing CSS. Since there is no direct measure of intuitiveness, it must be determined indirectly. In this experiment, web designers will be observed talking-aloud while interpreting constraint syntax as well as CSS syntax. Designers will be able to choose what they find the most intuitive from four shuffled code snippets presenting alternative ways of constructing a simple layout. Cursor data will be collected as a proxy for eye movements over each of the code snippets (Huang, White, & Buscher, 2012). This offers an indication of how fast the designer was able to understand the syntax and to reason with it. The experiment completes with the designers filling in a two-question survey to determine qualitative characteristics of the code snippets that they felt worked towards or worked against their intuitiveness. The experiment produces both qualitative data (from the survey given to the designers) and quantitative data (from the cursor movement and selection count of the various code snippets). The combination of which should provide a solid basis for justifying whether or not designers found constraint syntax intuitive.

4.1 Experiment Design

An overview of the process of the experiment can be given as follows:

1. Contact is established with a designer.
2. The designer is sent a survey regarding their experience and asked when an online meeting would be convenient.
3. Optional: The designer is sent a confirmation email.
4. A video call is established with the designer.
5. The designer views an example layout.
6. The designer is shown four code snippets which produce the layout given in the example. They think aloud while interpreting the code snippets. Simultaneously, cursor movements over the code snippets are recorded.
7. The designer selects the code snippet they find the most intuitive. At this point, they complete the think-aloud component of the experiment.
8. The designer completes a follow-up survey asking what they found intuitive and counter intuitive in the code snippets.
9. The experiment is concluded. Optionally, the designer is shown implementations of the code snippets and the data collected.

The remainder of this section describes and explains each step in its own subsection.

4.1.1 Establishing Contact

The target group for the experiment is designers, as defined in the introduction. Participants are sourced via the networks of the author and the supervisors. This provided some level of quality assurance as well as efficiency.

However, it would be assumed that sampling a greater number of web developers, particularly across several countries would produce more robust and reliable results. Given the time constraints for the thesis, and time required with each participant to record the think-aloud component as well as analyse cursor data having quick access to participants through existing networks was a critical factor.

4.1.2 Pre-experiment Survey

The survey included the questions:

- What is your age? <20, 21-25, 26-30, 31-35, 36-40, 41-45, 46-50, 51-55, 56-60, 61-65, 66+
- What is your gender?
- How many years have you worked with CSS?
- How much of that work has involved designing web page layouts in particular? Very little (<15% of projects), little (15%-35% of projects), somewhat (36%-65% of project), a significant amount (66%-85% of projects), virtually all of it (>85% of projects).
- How many hours per week do you estimate you have spent working with CSS?
- Do you use a CSS pre-compiler such as SASS or LESS for most of your work? Yes, or no.
- How many years have you worked with design in general?
- Please write down the languages you know and your proficiency with respect to reading and writing skills in each language using the scale: basic comprehension (able to work with very common words and short, simple sentences; reading and writing are very slow processes, relying heavily on support material such as dictionaries), moderate comprehension (able to work with more complex sentences regarding common topics; reading and writing are slow processes but achievable with support material such as dictionaries), high comprehension (able to work with complex sentences featuring advanced topics; reading and writing rely occasionally on support material such as dictionaries) or fluent (able to work with complex sentences featuring advanced topics with ease).

The first two questions regarding age and gender are basic demographic questions in order to understand to whom the results may apply. For anonymization purposes, the responses to age of the participants is grouped into fives. The question asking for the number of years spent working with CSS is recorded as a gauge of how much experience the participant has with CSS. As seen in the introduction, experience leads to intuitions about the technology. The hours per week in the last month question attempts to identify whether this knowledge is current and readily available. Since pre-compilers might put a layer of abstraction between the designer and the CSS, thereby changing how participants may understand raw CSS, a question about whether the participant uses such tools is also included. The next two questions are more to do with design experience. They are included since the authoring of CSS is usually just one step in a broader design process. Therefore, experience with conceptualising designs, such as page layouts, in general must also be taken into account. The combination of demographic and experience questions was inspired by a similar experiment measuring eye movements in code reading (Busjahn, et al., 2015). The data from these questions offers a way to reason with the results by understanding the backgrounds of the participants undertaking the study.

4.1.3 Confirmation Email

The confirmation email is an administrative aspect to the experiment: it ensures that the participant has a calendar entry readily available by including an invitation with the email. It is an optional step, since some runs of the experiment were arranged within a short enough timespan that no reminder was necessary.

Originally, this step also included sending an introduction to GSS. However, it was decided that first impressions of constraint syntax should be measured instead. Therefore, no clue as to the content of the experiment was given to the participants in advanced, aside from the fact that it involved CSS and layouts.

4.1.4 Establishing a Video Call

The use of Google Hangouts or Skype (since they both enable screen sharing, the use of one tool or the other comes down to whichever is more readily available for the participant) is used to establish a video call with the participant. At the beginning of the call, verbal consent is obtained for the recording of audio track of the video call. Once consent has been given, the record button within the Callnote[12] program is clicked, and the remainder of the call recorded.

[12] The Callnote program being referred to is the one based at the website here: https://callnote.net/.

Additionally, in this initial stage of the video call, the participant is given an overview of the up-coming experiment and examples of thinking aloud. This introduction reads as follows:

> Have you participated in or conducted a think aloud usability session before?
>
> (If no, share link to example think aloud video)
>
> Through-out the experiment, please try to think out loud and verbalise your thoughts as much as possible. For example, beginning sentences like "I am now looking for…", "I wonder why…", "I like that…", "I can see that…" and completing them with your current thoughts is a possible way to share your thoughts and reasoning. In the experiment, you will first be shown a picture of the intended simple layout, followed by some HTML which structures the elements in the layout. Please start thinking out loud as from the time you see the example layout picture. Further down the same page, there will be three CSS snippets and one Grid Style Sheets snippet. Please read each, then select the one you feel most intuitively represents the described layout. Once you have selected a code snippet, you may stop thinking aloud. Following the code snippet selection, there is a two question survey. Please answer the two questions given there and click the "Submit" button to complete the experiment. There is no time limit, please take as long as you would like. We would like to find out how understandable CSS and Grid Style Sheets can be, so please be open about what does and does not make sense. Do you have any questions?

This introduction attempts to set a baseline of what is expected of the participants as concisely as possible. The included description of thinking aloud as verbalising your thoughts is borrowed from a definition given by Jakob Nielsen (Nielsen, Thinking Aloud, 2012). Additionally, Nielsen's website is the source of the example think aloud video (Nielsen, 2014). The examples of thinking aloud are deliberately restricted to the beginning of the sentences to avoid biasing the thoughts of the participant. An earlier example sentence was "I can see that `margin: 0 auto;` will centre the element," an attempt to keep the experiment as layout-specific as possible, however it was decided against as it may have primed the participants to look for elements centred in this way.

The abbreviation of GSS for Grid Style Sheets was avoided because it is expected that it is still a relatively unknown acronym for most designers. A rough measure for this relative unfamiliarity with GSS was determined by comparing the gridstylesheets.org website's Alexa global site ranking of 1,278,389 with the w3.org website (technically the home page of CSS) ranking of 1,857 and the csszengarden.com (a popular website specific to CSS to counter the effect of w3.org featuring a range of topics) website ranking of 139,274. It is

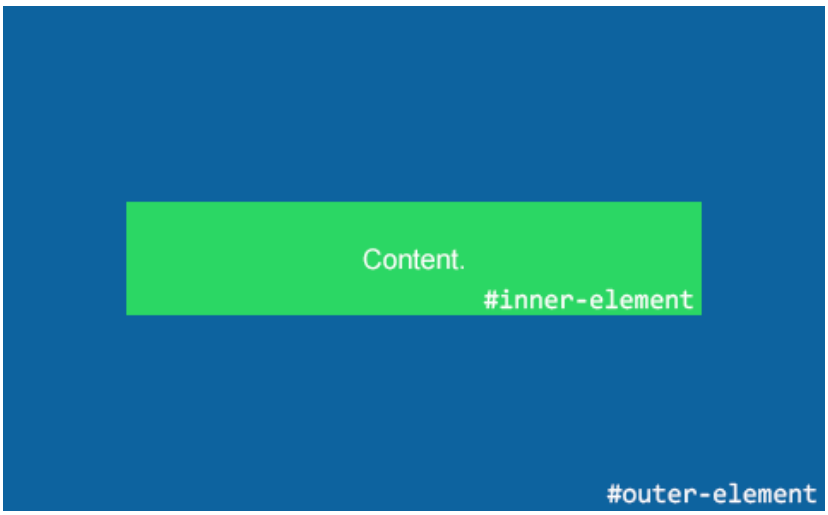


Figure 9: A simple layout involving two elements, one positioned in the centre of the other.

clear that GSS is a large margin behind having the same visibility as CSS (Alexa, 2016; Alexa, 2016; Alexa, 2016).

The last couple of sentences focusing on the absence of a time limit and the concept of evaluating CSS and GSS attempt to dissuade participants from thinking of the experiment as a test of their skill and instead encourage them to be open with their thoughts.

Overall, the sense of this script is conveyed to the participant; it may not be delivered identically to each participant depending on their prior experience.

4.1.5 Designer Views Example Layout

During the video call, the designer is given a web address pointing to a web page containing an example layout, shown in Figure 9. An explanation is provided with the image, detailing that the intention is to position the element found by the selector `#inner-element` in the centre, both horizontally and vertically, of the element found by the selector `#outer-element`. Further, it is noted in each of the upcoming code fragments the height and the width of each of the elements is virtually identical to that given in the example layout image. Additionally, the underlying HTML being styled is provided as well, it reads as follows:

```
<div id="outer-element">
  <div id="inner-element">
    <p>Content.</p>
  </div>
</div>
```

The HTML snippet has cursor tracking enabled. It is assumed that part of understanding the code that produces the layout requires reading and understanding HTML that is being styled by the CSS or GSS. Cursor tracking helps infer how much time the design spends looking at the underlying HTML compared to time spent looking at the CSS and GSS. Furthermore, it

may provide insights into aspects of the HTML that act as beacons: key locations in the code the designer refers to most often while conceptualising the design. However, by using cursor tracking in lieu of eye tracking and a small sample size, results to this effect are expected to be indicative or suggestive rather than conclusive.

Furthermore, basic syntax highlighting was implemented within the HTML snippet. This is done to aid readability in a way that replicates real world usage of HTML.

The example layout image was tested for various types of colour blindness to ensure that participants with protanopia, deuteranopia, tritanopia, protanomaly, deuteranomaly, tritanomaly, achromatopsia or blue cone monochromacy would still be able to distinguish between the two boxes given in the example layout. The online Coblis tool[13] and the downloadable Color Oracle[14] tool were used to conduct these tests and cross-check the results.

4.1.6 The Designer is Shown Four Code Snippets

By scrolling further down the page from the layout example and its description described in the previous step, the designer finds a selection of four code snippets as shown in Figure 10. The code snippets feature solutions to the simple layout described in the previous step. The solutions are based on methods utilising CSS Tables, Flexbox, CSS Grid and GSS. There is one method per code snippet. Like the HTML code snippet, each of the styling code snippets has syntax highlighting to replicate elements of the real-world web design environment. Additionally, there is a colour-preview window next to the colour declarations. Along with the selectors given as labels in the example layout diagram, this should assist designers in determining which element each of the declaration blocks is styling.

Each of the four styling snippets is automatically pulled via an `XMLHttpRequest` from a functional example of a corresponding implementation that has been tested in multiple browsers. This guarantees that the code is functional and achieves the intended effect. The code snippets are shuffled randomly to offset the impact that the order of the snippets may have on participant responses. The shuffling occurs after all snippets are loaded to ensure time to load the resources on the network does not influence the order of the code snippets.

To ensure readability, the code snippets are all the same height, have the same indentation and syntax highlighting and prevent line-breaks for as long as possible as screen width

[13] The Coblis tool is a shortening of **Co**lor **Bli**ndness **S**imulator. It was downloaded from http://www.color-blindness.com/coblis-color-blindness-simulator/.
[14] The Color Oracle creates a full screen filter for three different types of colour blindness; it was obtained from http://colororacle.org/.

Figure 10: A screenshot of code snippets presented during the experiment. Each snippet accomplishes the same result: positioning a light green element containing text content in the centre of a dark blue element. This is one possible arrangement of the snippets; they are presented in a random order each time the experiment is run. The select button moves the participant to the next step in the experiment.

reduces. This layout was achieved with a Flexbox implementation which respected the inherit width of the code snippet widths as given by the longest line (this is especially visible in the last three lines of the GSS code snippet in Figure 10) and vertically stretches all the code snippet elements to be an equal height. Further, the elements wrap as the page width decreases, ensuring that the participant does not need to horizontally scroll to see the code snippets, only vertically, avoiding a well-known usability problem (Nielsen, Scrolling and Scrollbars, 2005) and avoiding overly-preferential treatment for any particular code snippet. For future intuitive layout tests, this layout arrangement could also be the subject matter of an intuitive layout test similar to the one being presented here.

Like the layout of the code snippets, the content of each code snippet is also deliberately structured. As much as possible, the same order has been retained across the examples. For instance, all examples present declarations in the same order of the elements given in the HTML: `#outer-element`, `#inner-element`, then `#inner-element p`. The selector `#inner-element p` could have also been written as `p`; however, so the designer participating in the experiment would more readily comprehend where the paragraph element would be appearing within the structure of the document, the selector was written as `#inner-element p`. Furthermore, where possible, the order the properties are given is also same across the examples: one can see that the `#outer-element` declaration block begins with `background-color`, `height` and `width` and the `#inner-element p` declaration block begins with `text-align`, `padding` and `margin` in all four code snippets. The effort to maximise the consistency between the snippets has been undertaken with respect to the finding that small differences in code (albeit Python code) can lead to different interpretations (Hansen, Goldstone, & Lumsdaine, 2013). Furthermore, it is also based on the reasoning that when the code snippets are mostly the same, the greatest amount of time will be spent analysing the differences between them; that is, the differences most analysed by participants are specific to how layouts are generated by the given GSS and CSS. Although, it must be considered that the first code snippet viewed may take additional time to comprehend as the common components are viewed for the first time.

The remainder of this section describes the CSS present in the code snippets from which the designer can select.

CSS Common to all Snippets

In order to avoid repetition, CSS common to all the snippets is covered here before highlighting the distinguishing features of each snippet in its own section. This includes some including basic reset CSS (intended to generate a common look among the browsers) and CSS intended to assist designers identify the relationship between the snippets and the layout example.

The reset CSS sets the font-family to be sans-serif, the colour of the text to be white and the body to have a margin of 0. It is common to all code snippets. It is excluded from the code snippets because it has virtually no impact on the resulting layout, apart from the fact that it should appear at the very top, left of the browser viewport rather having an eight-pixel margin (the recommended default value for the body (W3C, 2016)). The removal of the eight-pixel margin ensures that it is clear the layouts begin at the same location when using multiple tabs to compare implementations of the code snippets.

Some CSS is common to all code snippets, yet included within the snippet rather than refactored into its own style sheet as it is intended that the designer views it. For example, in all four code snippets, the `#outer-element` declaration begins by setting the `background-color` property to `#024675` (a dark blue), the `height` property to be 294 pixels and the `width` property to be 480 pixels, before setting other properties that are more specific to the layout implementation. The use of the background colour here is to help designers identify to which element of the layout it belongs to, like an indirect colour coding. A similar effect is intended through setting the background colour of the `#inner-element` to the green shown in the layout in each of the code snippets (except in the Tables snippet, for reasons explained in the next section): it provides a colour code mapping from the CSS snippet to the example layout. Lastly, aligning the text to the centre, giving it top and bottom padding of `1.5rem` and left and right padding of `0` is included in each of the code snippets as, unlike the reset CSS, it is relevant to the layout: centring the paragraph text and ensuring that it has padding above and below, making the element a bit taller than the text it contains.

CSS Tables

This code snippet is based on CSS Tables which are included in the CSS 2.1 specification (W3C, 2011). The key features are that `#outer-element` is given a `display` value of `table` and `#inner-element` is given a `vertical-align` value of `middle` and `display` value of `table-cell`. This results in the browser treating the layout as a table with one cell and the cell has its content vertically aligned in the middle. As will be seen, this approach is slightly different to the others: since the table cell's `background-color` fills the whole table, the paragraph element was instead given a green background. Alternatives were considered such as using the `border-spacing` property or adding padding to the cell; however, unlike the other solutions, this would not have allowed `#inner-element` to expand and accommodate a larger amount of content. Thus, the behaviour of the table display forced a solution that deviated from the pattern found in the other solutions where `#inner-element` is sized and displays with a green background as opposed to the paragraph element within it. Essentially, although the paragraph element is being given the green background in this snippet, the behaviour of the layout as more content is added is visually identical to the other implementations.

Although the usage of tables for laying out web pages is discouraged (Kistner, 2004), it has been observed that CSS Tables can be 'cherry-picked' for their layout benefits, such as vertical centring, while many of the problems with using HTML Tables for layouts can be avoided (Toh, 2014). Further, being part of CSS 2.1, CSS Tables are supported by a large number of browsers, Can I Use reports almost 98% of browsers are able to use CSS Tables

with Internet Explorer 6 and 7 being the notable exceptions (Can I Use, 2016). Therefore, CSS Tables have been included as a code snippet in this experiment.

Flexbox

The key features of the Flexbox code snippet include an `#outer-element` `display` property value of `flex` and a `width` of `70%`; and `margin` of `auto` for `#inner-element`. The `auto` value vertically and horizontally centres `#inner-element`, as defined in the Flexbox specification. Specifically, "positive free space is distributed to auto margins in that dimension" (W3C, 2016) and `#outer-element` consists only of free space and `#inner-element`, it is therefore positioned centrally. The `70%` value gives `#inner-element` a width 0.7 times the size of `#outer-element`. The paragraph element within `#inner-element` has a `margin` of `auto`; although, this could have been set to `0` for consistency with the CSS Grid and Grid Style Sheets implementations. However, experimentation had already begun once this was noticed and it was decided to keep the experiment as identical as possible for all participants. Only the CSS Tables code snippet relies on the paragraph having a margin of auto, as it is required to centre the paragraph block, as opposed to the `#inner-element` block in this case.

Flexbox, although still a work in progress (W3C, 2016), has been positioned as a solution to various layout problems (Walton, 2016). Its adoption may have been hindered by significant changes after it was first introduced and varying browser implementations (Coyier, 2012; W3C, 2011). However, it now has global browser support of almost 97% (Can I Use, 2016) and its usage is encouraged by some leading voices in the web development community (HTML5 Please, 2016). Therefore, it is included among the code snippets, similarly to CSS Tables, as something that is likely to be familiar to web designers.

CSS Grid

This code snippet is based on the upcoming CSS Grid layout specification. Importantly, the line:

```
grid-template-columns: 3fr 14fr 3fr;
```

creates a grid of three columns within `#outer-element` using flexible space. The formula to calculate the size of an fr unit is given as:

`<flex> * <free space> / <sum of all flex factors>` (W3C, 2016)

In the code snippet, this results in column sizes of:

Column 1	**Column 2**	**Column 3**
$\frac{3 \times 480\ pixels}{20}$	$\frac{14 \times 480\ pixels}{20}$	$\frac{3 \times 480\ pixels}{20}$
$= 72\ pixels$	$= 336\ pixels$	$= 72\ pixels$

In other words, the second column of `#inner-element` is given a width of $336 \div 480 = 70\%$ of `#outer-element`, just like the implementations of the other code snippets. Therefore, placing `#inner-element` in column 2 using:

```
grid-column: 2;
```

has the effect of giving `#inner-element` a width 70% the size of `#outer-element`'s width.

Unlike CSS Tables and Flexbox, CSS Grid is currently only available in popular web browsers if certain configuration flags are set to enable it: `layout.css.grid.enabled` in Firefox and "Experimental web platform features" in Chrome. Therefore, along with GSS, this is expected to be unfamiliar to designers since it is unlikely to be used for client work. In this way, it provides a convenient control test: although designers may be familiar with Flexbox and CSS Tables, the novelty of GSS is compared side-by-side with CSS Grid.

GSS

The code snippet featuring a GSS implementation of the layout is the most idiosyncratic because it introduces the most novel syntax, constraint operators. Most of the constraint operators given are intended to act like assignments, to make the code as familiar as possible to the designer. For instance, setting height, width, top and left of `#outer-element` and the height of `#inner-element` all read as unidirectional value assignments. The real test of the experiment is the final three lines constraint properties between elements:

```
#inner-element[width] >= #outer-element[width] * .70;
#inner-element[center] == #outer-element[center];
#inner-element[height] == (#inner-element p)[height];
```

These three lines represent layout techniques unique to GSS (when compared to the CSS code snippets); that is, constraining one element's property values against the values of the properties of other elements; especially, aligning the centres of two elements. A "greater-than or equal to" constraint as well as an "equals" constraint is used as a hint to the designer than these are not assignments; rather constraint operators. Further, some math is included, making the width of `#inner-element` proportional to `#outer-element`. No more than three lines of constraints are included so as to not overwhelm the designer with the new syntax. Rather, the majority of the code snippet has retained syntax resembling that of CSS in an

attempt to test the intuitiveness of the concepts of GSS rather than the ability to interpret a large number of constraints given in a new syntax.

Further, this code snippet involves positioning the layout relative to the window, else the design may be under-constrained and may render `#outer-element` with negative top and left margins, pulling part of the layout off-screen (GSS, 2015). This is not required in any of the other code snippets.

4.1.7 Cursor Movements Over Code Snippets Are Recorded

This step of recording cursor movements over the code snippets by the participants occurs simultaneously with the previous step as the participant moves their mouse over the code snippets. Only cursor movements over the code snippets is recorded. There were several characteristics to consider in the when establishing the mouse tracking system:

- The page is dynamically generated. Specifically, the code snippets are retrieved via a `XMLHttpRequest` and shuffled.
- The page responds to changes in browser window size; therefore, code snippets may be different widths when viewed by the participants.
- As it is a scientific experiment with only a small set of participants, as many mouse movements as should be recorded as possible. Some mouse tracking systems only collect samples.

In consideration of the situational requirements given above, a cursor movement recording system that was aware of this context was developed to ensure they were all adequately addressed. Further, this aids the analysability of the results since the implemented system also distinguishes between interactions with each code snippet and records mouse coordinates relative to the code snippet block: beginning one interaction when the mouse enters the code snippet and ending that interaction as the mouse exits the code snippet. Additionally, it offered considerable flexibility in the presentation of the results as will be seen in later sections.

The implementation of the cursor recording software is written in JavaScript and runs directly in the browser as the participant views the code snippets. Two event listeners are added to

```
{
    ...
    "interactions": [ // A sequential record of interactions during experiment
        {
            "title": "CSS Grid element centring", // Element title
            "width": 314, // Width of element currently tracking mouse steps
            "height": 433, // Height of element currently tracking mouse steps
            "interactionSteps": [ // Sequential record of mouse steps
                {
                    "t": 1468506743538, // Timestamp from Date.now()
                    "x": 313, // Pixel distance from left content edge
                    "y": 94 // Pixel distance from top content edge
                },
                {
                    "t": 1468506743569,
                    "x": 311,
                    "y": 93
                },
                ...
            ]
        },
        {
            "title": "Flexbox element centring",
            "width": 251,
            "height": 433,
            "interactionSteps": [ ... ]
        },
        ...
    ],
    ...
}
```

Figure 11: An abbreviated sample extract of the interaction data collected during the experiments.

the example layout image, the HTML snippet and each of the code snippets. The first event listener listens for a mouse entering the element and begins recording the interaction, the second listens for the mouse exiting the element and concludes the interaction. It is possible for multiple interactions to occur on each element during the experiment. During an interaction, a third event listener is added to the element to the recording of handle mouse movements. Each mouse movement causes the script to save a mouse step in memory. A mouse step consists of a timestamp and x and y coordinates recording the position of the cursor at the timestamp. In JavaScript, the mouse step is an object with these properties, named with the single letters ‘t’, ‘x’ and ‘y’ respectively to minimise space required when the data is sent back to the server or downloaded.

Altogether, the mechanism handling the recording of interactions and their constituent mouse steps produces the structure shown in Figure 11. It can be seen that an array, `interactions`, stores a sequential record of the interactions. Each interaction features the `title` of the

element subject to the interaction, as well as its `height` and `width`. The height and the width are recorded, so the exact shape of the code snippet can be replicated when analysing the mouse movements after the experiment. By using the timestamp given by JavaScript's `Date.now()` function, the mouse step's timestamp records "number milliseconds elapsed since 1 January 1970 00:00:00 UTC," as measured by the participant's browser (Mozilla Developer Network, 2015). Generally speaking, since the `Date.now()` function uses a clock on the client side which may be unreliable, its usage is advised against for data that will be processed further on the server. Further, it has been noted that the system time as reported by the browser may be off by an average of 7.5 milliseconds, up to 15 milliseconds (Resig, 2008). Although, this test is somewhat outdated at the time of writing and accuracy to the nearest 100ms acceptable in this experiment as it is only intended to approximate where participants direct their gaze and for how long. As to the issue of doing further processing on timestamps generated by the client machine, for this experiment the timestamps are treated relative to one another, so long as the participant does not make changes to system time the results should be sufficiently reliable.

As well as clock performance, consideration of space usage for this record-all-movements approach is required, since the number of mouse steps recorded can quickly move into the thousands. As a rough indication, moving the mouse reasonably consistently and recording the corresponding mouse steps for 4659 milliseconds in the Firefox 49 browser resulted in 379 mouse steps being recorded. As a minified JSON file, this array of mouse steps takes up about 15 KB of space. The net effect may cause lag to affect the participant's experience during the experiment: around 4 MB of data could be generated during a twenty-minute session with consistent mouse movement over the elements that recording it. Despite the challenge of the large amount of data being generated, it is critical that the results are recorded the first time a participant engages with the experiment, since the participants

4.1.8 The Designer Selects a Snippet

Once the designer has viewed all the code snippets while talking aloud, they select a code snippet they believe most intuitively creates the given layout example by clicking its 'Select' button. Once a 'Select' button has been pressed, a two-question follow-up survey revealed on the page, and the browser scrolls down to its position. At this point the designer is advised they may stop thinking aloud.

4.1.9 The Designer Completes a Follow-up Survey

This step aims to capture qualitative aspects of the code snippets that relate to how intuitive the designer found them. The questions are:

- Explain why you chose {title of selected snippet}. Please try to include specific details about what you found intuitive in the code.
- Describe characteristics you found counter-intuitive or required additional thought in some of the code snippets.

The placeholder {title of selected snippet} is filled in with the title of the code snippet the designer selected in the previous step. This step gives respondents a chance to reflect on the code they had just read, rather than drawing all the information from the initial reaction to the code during the think-aloud component of the experiment.

At this step, the participant is also asked for their survey code. This helps connect the survey results with the experiment results without the need to use personal identifiable information.

The results are then sent as a JSON file to the server. The data included in the JSON file and an explanation is given in the table.

Data	**Reason for inclusion**
Survey Code	Used to connected the experiment results with a survey submission.
Browser Width	Recorded so that the layout of the experiment as seen by the participant can be recreated later on when viewing the results.
Code Snippet Order	Records the order that the code snippets were presented to the participant so it is possible to match up phrases like 'the first example…' and 'I found the third code block…' to their corresponding code snippets.
User Agent	Indicates the browser used during the experiment so that the same browser may be used when viewing the results (if it is available).
Selection Explanation	Stores a written explanation from the participant stating what they found intuitive about the code snippet they selected.
Non-selection Explanation	Stores a written explanation from the participant stating what they found counter-intuitive in the code snippets.
Interactions	A chronological, timestamped record of mouse movements over the code snippets, to approximately visualise how the participant read the code snippets.

Table 1: Data included in the results of the experiment sent from the participant's browsers to the server.

Additionally, notes are taken during each experiment and included in Appendix B and the audio during the session is recorded (if permission has been granted during the experiment). If there is a technical issue preventing the data from being to the server, the participant has

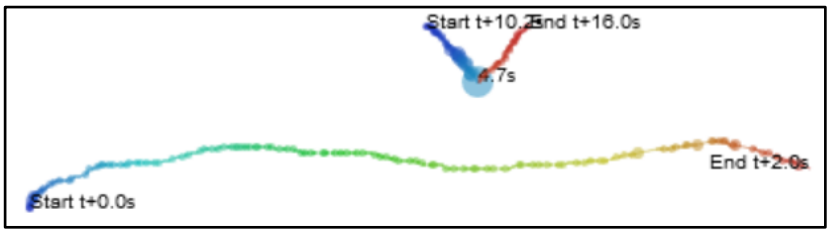


Figure 12: A sample of the visual generated from the collected mouse movement data.

an option to download the data captured during the experiment as a JSON file which can then be emailed.

Once the data constituting the experiment results for the participants is stored on the server, it can be analysed.

4.1.10 The Results Are Combined

Qualitative and quantitative data is collected during each run of experiment with a participant.

The quantitative data includes:

- The count of selections of each code snippet made by the participants
- The timestamped mouse movements over code snippets recorded during the participant's participation.

The qualitative data includes:

- The content of what the participant stated during the think-aloud component.
- The content of the textual description of what the participant found intuitive.
- The content of the textual description of what the participant found counter-intuitive.
- The content of the experiment notes.

Most simplistically, the count of the selections of each code snippet provides an indication of which code snippet designers found most intuitively represented the given layout. The mouse movements may also support this by signalling what the participant was focusing on and for how long. Figure 12 shows a sample of a visualisation generated by the experiment system interpreting the mouse movement data. Each line represents an interaction. The lines begin with a blue hue and ends with a red hue. The hue in between is generated proportionally to how far through the interaction a given timestamp lies.

$$h = 234 - (p * 234)$$

h is the hue as represented on a scale of 0 to 255 as specified for in the hue, saturation, lightness and alpha (HLSA) CSS color value (W3C, 2011); p is percentage of the way through the current interaction the mouse step lies. A limited number of hues (0 to 234) was

chosen because of the similarity of high value hues to low value ones: both appearing red. As it stands, the spectrum covered ranges from 234 (blue) to 0 (red), both clearly identifiable, primary colours.

Labels are also present to assist in interpreting the mouse movement visualisation. Each interaction has a label at its start showing where the interaction starts and how many seconds after the very first interaction it started. Likewise, a label at the end of the interaction displays when the interaction ended in seconds since the very first interaction. In Figure 12, it can be seen that the bottom line began as the very first interaction. The mouse was moved reasonably consistently from left to right, before leaving the snippet two seconds later. The mouse returned to the snippet 10.2 seconds after the very first interaction to begin the second interaction at the top of Figure 12. In the second interaction, a second label stating '4.7s' is present next to a circle slightly larger than the others; this indicates that the mouse remained stagnate at this location for 4.7 seconds. To avoid cluttering the screen with labels, only mouse stagnations longer than 1 second are labelled.

As noted, the circles indicating the mouse steps have difference sizes. The is proportional to the amount of time the cursor stagnated at each position. For instance, the large, light blue circle labelled 4.7s is the largest in Figure 12 indicating that the cursor remained at this position the longest. The size of the radius is given in the following equation:

$$r = min(\log(d), 1)$$

where r is the radius in pixels and d is the duration of the mouse step in milliseconds. A minimum radius of one pixel is enforced on the circles so each step remains visible. Since the mouse step durations can range from several milliseconds to tens of thousands of milliseconds, the logarithm of the duration. This visually distinguishes between the mouse steps that last only a few milliseconds and likely part of the mouse gliding from one position to another, from those that last several seconds and those that last 10 or more seconds.

Drawing on the mouse-step data visualisation, it is possible to determine which parts of the code snippet were read several times, and which parts were focused upon for a large amount of time. As seen from the related works, this may indicate beacons in among the code snippets (repeated views of a particular aspect) or increased cognitive load due to counter-intuitive aspects of code snippet (longer fixations) respectively.

In summary, a lot of data is generated from the mouse step tracking, and this visualisation diagram uses lines, color, circle size and assistive labels to make it possible to interpret. While interpreting it can be see how the participant moved through the code snippets identifying possible code beacons as well as aspects in the code that induce a higher cognitive load.

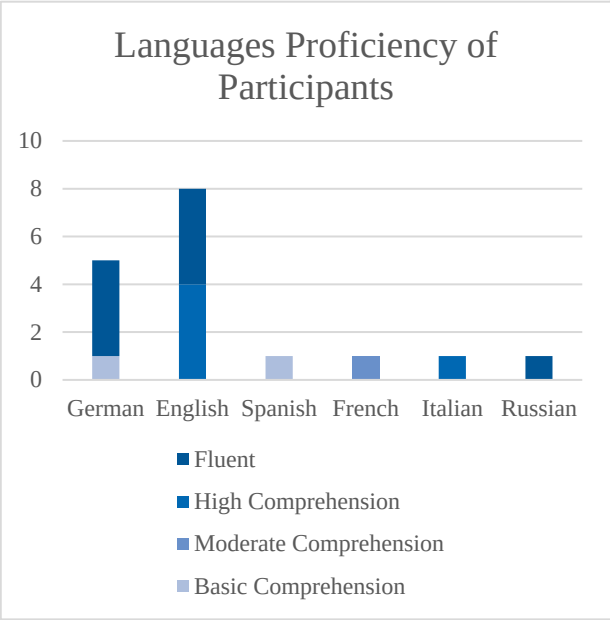


Figure 15: The languages spoken by the participants, divided by fluency.

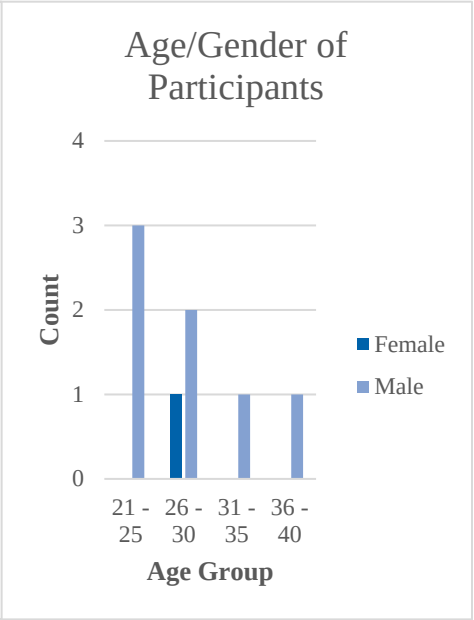


Figure 14: The age and gender of experiment participants.

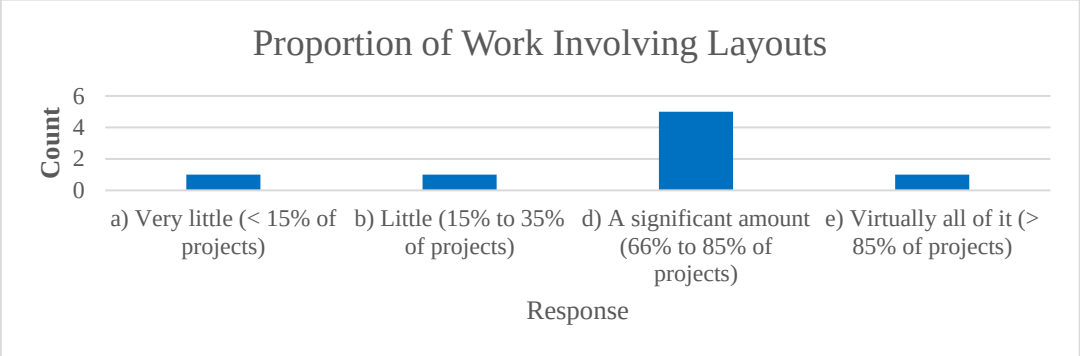


Figure 13: Graph comparing the proportion of CSS work involving layouts of the participants.

4.2 Results

The experiment intended to provide evidence as to whether or not designers found constraints in CSS intuitive or not. The results come in two parts: the results to the participant survey, which identifies to what extent the participants could be considered designers as understood in the context of this paper. The second part presents the data collected during experiment itself.

4.2.1 Participant Survey

This section looks at who is participated in the experiment. In total there were 7 participants in the experiment. As can be seen in Figure 14 and Figure 15, a range of ages and languages were presented. Figure 12 shows that there was 1 female and 6 males participating in the experiment. Additionally, the ratio of those 30 or under to 31 or older, is 5:2, weighting the results toward a slightly younger demographic, when considering the career lifespan of a designer. Figure 15 shows that English was spoken by all 7 participants; however, only 4

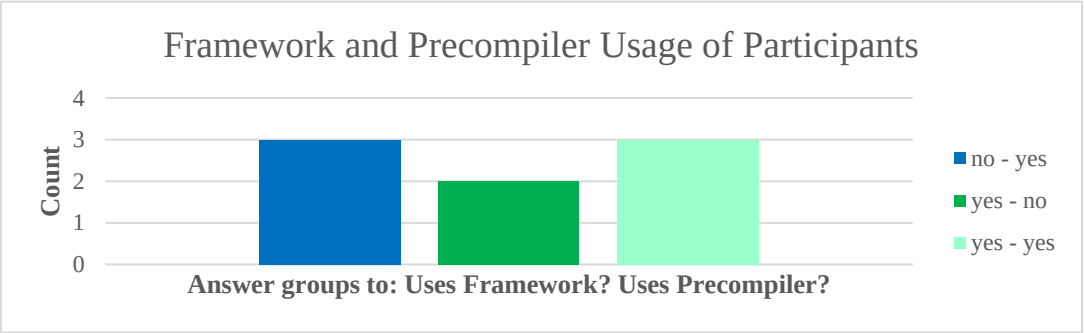


Figure 16: Answer groups to the pair of questions asking participants whether they use frameworks for the majority of their work and whether they use precompiler for the majority of their work.

reported they were fluent while 3 reported high comprehension of English. Fluent speakers of German and Russian were also present during the experiment. Although several languages are involved, they all lie within the Late Indo-European family of languages as shown by the MultiTree project (MultiTree, 2014).

All participants recorded a long-term engagement with CSS and design, as reflected in **Error! Reference source not found.** The participants noted an average of 7.7 years of e xperience with CSS (from a minimum of 4 to a maximum of 11) and 7.3 years of design (from a minimum of 4 to a maximum of 13). Although this may seem to indicate strong familiarity, the result for hours per week spent working with CSS in the last month offset this. For this metric, an average of 6.1 hours of CSS per week, with a minimum of 0 suggests that CSS is not a core part of many participant's work and it may not be fresh on their minds. Despite the semi-inactive CSS status, 5 of the 7 participants responded that 66% (termed 'a significant amount' in the survey) or more of their projects involved layouts. This goes some way to validating that the participants represent the intended group of designers.

The participants also used some form of help while using CSS. Figure 16 shows that all participants had assistance from at least a framework or a precompiler. 3 of the 8 participants use both. Precompiler were slightly more popular, used by 6 participants compared to 5 participants using frameworks.

To sum up, the participants did reflect to intended target group to a significant degree: all reported at least several years of experiences with CSS and most reported that at least significant amount of their work involved layouts. However, the present engagement with CSS of many of the participants may be considered a bit low.

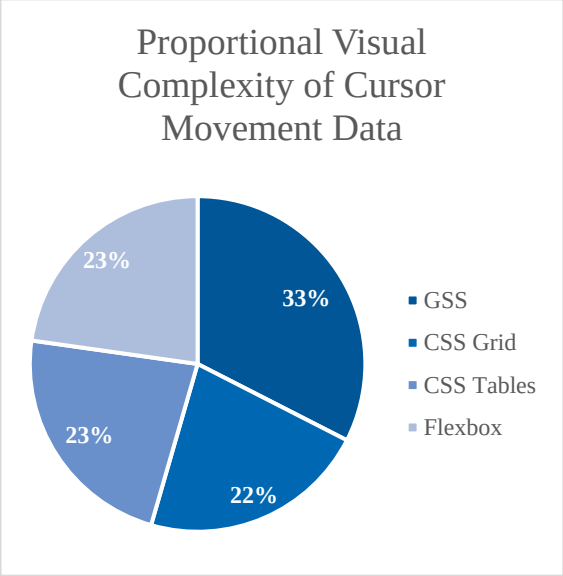


Figure 19: An approximation of how much time participants spent looking at each code snippet based on a visual assessment of the cursor movement data.

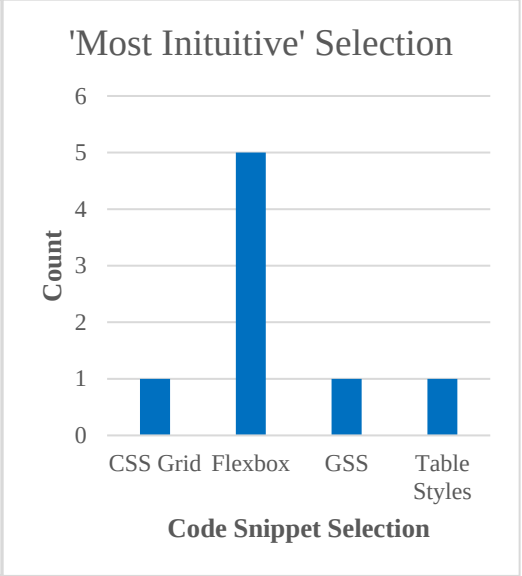


Figure 17: The sum total of the code snippets chosen as the most intuitive during the experiment.

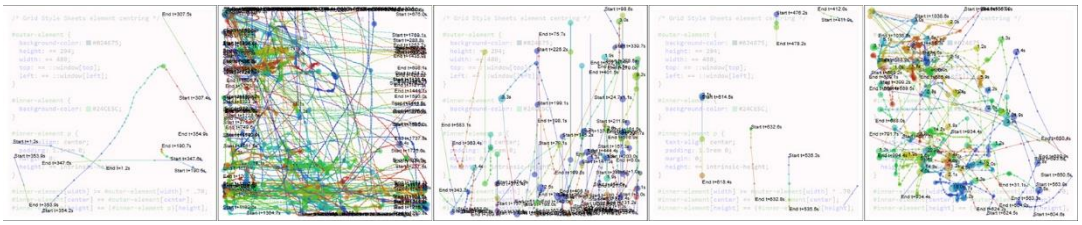


Figure 18: Cursor movements over the GSS-based layout snippets.

4.2.2 Experiment Results

The experiment has yielded quantitative results in the form of the count of selections made by participants for the most intuitive code snippet and cursor movements as well as qualitative data as participant's thought aloud during the experiment and wrote down what they found intuitive and counter-intuitive. Appendix B includes transcripts and notes taken during the experiments while Appendix C shows visualisations of cursor movements.

The most direct answer to whether GSS could be considered intuitive relative to other layout CSS techniques is given by a combination of Figure 18 and Figure 19. Only one participant favoured GSS over the alternative layouts as shown in Figure 19. Further, an approximation of the proportion of time spent engaging with code snippets based on cursor movement data suggests that participants spent most time interacting with the GSS snippet and an about an equal amount of time interacting with the other snippets, as shown in Figure 18. Using the visualisations from Appendix C, it is possible to focus-in on points of interest in the GSS snippet. The result of this is shown in Figure 17. There was significant variety in the amount of cursor movement data collected. However, concentration points toward the bottom and the top left are visible. These correspond to the positions of the constraint syntax and the GSS

`::window` keyword respectively. The second and fifth cursor-movement visualisations in Figure 17 from users highlight this especially.

The transcripts suggest a similar pattern of additional though being record for constraint syntax. The think-aloud results showed that many of the participants expressed confusion when encountering the GSS syntax for the first time. Especially, the double equals sign was noted as confusing and uncertainty as to the effect it would have was expressed by several participants. Despite this, some participants noted redeeming features about the GSS syntax. One participant found the ability to define relationships between the properties of elements compatible with thinking about layouts. At least two participants noted that Flexbox worked like 'magic'; while implying that being able to see exactly what was happening as shown by constraints instead might be beneficial. The syntax of GSS struck most of the participants as being irregular in a CSS context, despite its similarities.

The responses to the last two questions asked of participants during experiment provides further evidence consistent with the results covered given so far. The aspects of GSS considered counter-intuitive in this responses included: the use of math symbols, increased line lengths, the combination of '`:`' and '`==`' when defining values, the verbosity of GSS, the keyword `intrinsic-height` and generally confusing syntax. Further, one participant noted that they did not need to jump between selectors with the choice of Flexbox, implying that identifying all the constraints a property participated in was unwanted cognitive load. As a counter-point, a respondent familiar with GSS syntax noted that the 'code speaks for itself' observing that using constraint syntax to express relationships between element properties reflected a way of thinking about layout. A couple of other comments were supportive of GSS as well. One participant noted it was 'expressive', another participant found the idea of defining the relationships between elements interesting. However, the majority of comments about the constraint syntax were unsupportive of the idea that it was intuitive relative to other CSS layout techniques.

In sum, the results indicate that constraint syntax in CSS is not an intuitive approach for layouts relative to alternative CSS layout methods.

4.3 Discussion

On the surface, the results counter the hypothesis that constraint syntax is intuitive for designers relative to other CSS layout approaches. However, the nature of the experiment means that this conclusion is not infallible. The selection of the of the participants and the experiment method itself could be improved to make generate more concrete results.

```
#outer-element {
    /* ... */
    grid-template-columns: 3fr 14fr 3fr;
}

#inner-element {
    /* ... */
    grid-column: 2;
}
```

Figure 20: A combination of properties that was found to be intuitive.

The survey showed that the target group of participants was reached. All participants reported more than several years or more of both CSS and Design experience. However, this was a self-reported figure with no validation of its truth. Further, CSS and design may not be the core work tasks of the participants and this measurement may not be an accurate reflection of CSS ability. Indeed, this is indicated by an overall low average number of hours spent doing CSS work per week, 5.5. On the other hand, the definition of designer in this paper refers generally to those who use CSS to produce layouts professionally, it is not clear whether this must be the central task of their professional work. In today's context of increasingly cross-functional, multi-skilled teams, the survey results may actually reflect the reality of the role of a designer. In other words, designers have a long-term involvement with CSS but it is only involved a fraction of their day-to-day tasks. However, determining the extent to which this is true is out of the scope of this thesis.

Indeed, the scope of this thesis was noted as being 'small scale'. It is self-evident that a group of 8 participants cannot produce conclusive findings for designers around the world. It was noted in the results section that all the languages were Late Indo-European languages and it involved mostly those fluent in German or English, to the exclusion of thousands of others. Age groups younger than 20 and over 40 are also not represented and only one female participated in the experiment. Each of these demographic shortcomings would need to be addressed in order to draw findings that could be considered relevant for designers around the world, as CSS is implemented on a global scale. However, for the small scope of the thesis, the multiple age ranges and multiple cultures of the participants is a starting point for generating suggestive findings.

As for the results themselves, there is also room for improvement. Due to time constraints and due to the unexpectedly poor quality of the cursor data in its capacity to act as a proxy for gaze, the cursor was not analysed to the originally envisaged extent. Instead, it was only visually assessed for where participants spent the most time concentrating, increasing the possibility of errors. The poor quality of the cursor movement could be rectified in future

versions of this experiment with an eye tracking system. However, in current experiment the poor quality of the cursor movement data was made up for by the qualitative data collected.

In generally, all participants were able to effectively think out loud during the experiment. It was particularly insightful to see moments where they suddenly understood how something was working. For instance, from these observations, it could be said that the combination of lines in Figure 20 was intuitive as it is understood in the context of this paper. Although some participants expressed that they had not seen CSS Grid before, they were able to determine that the inner element would lie in a column whose width was 14fr (although what the fr unit could mean proved to be a bit more puzzling). Likewise, once they had acclimatised a little bit to the GSS constraint syntax, some participants were able read statements such as the one aligning the centres of the inner and outer elements, or the statements aligning the outer element to the top left of the window and determine their effects.

That highlights an area of further investigation. This experiment was interesting because it captured, in several cases, the first impressions of people using both CSS Grid and GSS. However, it became clear through the course of the experiment that there is a difference between considering something intuitive on first impression and considering something intuitive after having learnt as foundational set of rules. In this experiment, this gave CSS Grid an advantage over GSS because it only introduced new properties and property values, whereas GSS introduced new syntax into CSS. Several participants commented positively on CSS Grid and were able to relatively quickly determine what its properties and property values meant. On the other hand, the introduction of the double-equals sign and the double-colon prefix prepended to the window selector in GSS drew comments of confusion. Yet once the confusion passed, the majority of designers were able to determine the effect of the constraints. Therefore, the experiment could be improved by comparing the relative intuitiveness of layout implementations once a designer had learnt foundational concepts of constraint syntax to give it equal footing with CSS. Nevertheless, comparison between first-impressions of CSS Grid and GSS has been useful to produce such a finding.

In summary, the results suggest that constraint syntax is not intuitive when compared to other layout approaches in CSS. As shown by the comparison to the relatively unknown CSS Grid specification, this may be largely to do with the introduction of new operators into CSS. Further, the caveat applies that constraint syntax is not considered intuitive on a first impression. Re-testing other layout scenarios after introducing the foundational concepts would be a future direction for this investigation. Lastly, the results are suggestive, repeating and improving the experiment, with eye tracking for example, for a broader set of demographics would help solidify the results.

5 Conclusion

5.1 Summary and Key Findings

This thesis set out to look at using constraints for web layouts, evaluating how intuitive they are for designers compared to other, current approaches to web layouts. Here the flow of the document is recapped and key findings are highlighted.

The introduction defined the key concepts as they are used in this paper. This included defining the work of a designer to involve working with CSS by definition. The particular strand of CSP that is used for layouts was introduced, along with key features of GSS. The scope section established a precedent for the rest of the paper: the experiment was being treated like a small-scale usability study and there would be a heavy emphasis on layouts and constraints to the exclusion of other CSS and GSS capabilities. Next, the objective of answering the core question and the relevance is established.

The subsequent related work section draws on academic work as well as work from current material from leading voices in web design. Firstly, a paper introducing CCSS, upon which GSS is based, is summarised. It is noted that the paper positions constraints as a solution to a difficult-to-understand, restricted CSS 2.0 specification (Badros, Borning, Marriott, & Stuckey, 1999), although it does not explain why constraints might be easier for designers to understand. This is an early instance of an assumption being made that some technology is intuitive without adequate testing, a theme repeated later in this paper. Despite this, it is foundational, along with the GSS documentation for understanding how constraints could be applied to web layouts.

Switching from looking at code and technical implementations to looking at the mind, the next related work identifies key concepts to do with online processing during reading and looking at how code is read. This is a deliberate attempt to break the cycle of assuming intuitiveness, and look for bottom-up ways to reason whether something was intuitive or not. In particular, it was noted that longer eye fixations and an increased number of regressions indicated hard-to-understand material based on Rayner's work (Rayner, 1998). It was tentatively assumed that one could look for such patterns during the reading of code as well as some related studies had done. Applications of eye tracking to code readers were also covered, determining that some code characteristics could contribute to quick comprehension of code: for instance, the use of beacons and a high level of regularity.

After establishing constraints and reading patterns to reason about intuitiveness, the related work section stepped outside of academic papers and looked at the W3C Mailing List as a related work to see how intuitiveness was reasoned about during the foundational years of

CSS. After analysing emails from 1995 to 2008, it was found that the decision as to whether or not some feature of CSS is intuitive mostly comes down to opinion.

Frameworks were briefly assessed as the next non-academic related work as their popularity seems to be underpinned by the easy in which they allow designers to manager layouts compared to CSS. It was found that the most popular frameworks implemented a grid system and columns and rows may play a large role in how designers think about layouts.

Moving on from the mainstream frameworks, state-of-the-art web page layouts were considered based on a recent talk from Jen Simmons. The talk was constructed in a problem-solution way: the problem was the frequency at which a boilerplate web layout pattern was found across the web; the solution was some of the recent and upcoming CSS specifications as well as to consider alternative layouts; for example, translating ideas from magazine layouts (Simmons, 2015). Although the intuitiveness of the technologies was not considered directly, commentary on Flexbox and the note to use tools to assist with grid creation suggested intuitive use of these technologies was yet to come.

Last of the related works were the CSS specifications. It was found that specifications CSS Tables, Flexbox and CSS Grid indirectly give designers access to constraints, they are just separated by a layer of syntax. It was shown that constraint syntax could be used to implement several concepts across the specifications, making constraint syntax quite versatile. However, it was also observed that constraint syntax may quickly become verbose while trying to capture layout behaviours that are concisely expressed in purpose-built CSS specifications. Such an effect would reduce the capability of constraint syntax to be intuitive.

A brief analysis places the paper in a historical context. It looks at the development of HTML and CSS, with a focus on layouts and how the early development shaped current thinking about web layouts, especially the impact of table-based layouts.

The experiment forms a substantial part of the paper, and results in primary evidence refuting the hypothesis that designers find constraint syntax intuitive for layouts relative to other CSS approaches. An attempt was made to use cursor movement data as a proxy for reading, insufficient quality (due to trackpads) meant it was incomplete. Although, taken at face-value, it suggested the greatest amount of time was spent looking the novel syntax introduced by GSS so it nevertheless contributed to the results. Improvements for future research were identified such as switching from mouse tracking to eye tracking and the inclusion of a broader demographic to better represent the diversity of the designer population. Furthermore, it was realised that testing intuitiveness upon a first-impression, as was done in this experiment, may yield different results to testing intuitiveness once a participant has

familiarised themselves with foundational concepts. Nevertheless, the experiment accomplished what it set out to within the scope of this paper. It both suggested that constraint syntax could not be considered as intuitive as existing CSS approaches as well as implemented a process for testing for relative intuitiveness.

5.2 Critical Review

This paper set out to achieve two objectives. The first being to produce a preliminary conclusion as to whether designers find constraint syntax for layouts intuitive relative to other CSS approaches. The second was to test a method for evaluating the intuitiveness of language constructs. This critical review looks at the extent to which they are fulfilled within the given scope and identifies areas for improvements.

A task that seemed relatively straightforward at the beginning of the thesis, to determine the relative intuitiveness of various syntax (including constraint syntax) for web layouts, turned out to require a lot of work in the details. Although core definitions given in the introduction were relatively simple, the extrapolation of ideas from the related works revealed the complexities of the thesis topic. The CCSS paper (Badros, Borning, Marriott, & Stuckey, 1999) showed there are many aspects when applying constraints to CSS: how they handle cascading, weighting constraints, establishing the syntax to use, the interaction of designer and user stylesheets and so on. This paper focused on a bare minimum of constraint syntax: positioning relative to the window and a few bare-metal positioning constraints, with one mathematical operator are used in the experiment. However, any of the other concepts from the CCSS paper could have been tested for its intuitiveness. Further, GSS features constraint-based extensions, such as its VFL, representing other areas skipped by this paper. Instead, this paper focused on only included a few examples in its experiment to avoid overwhelming the participants (although many participants seem to have been overwhelmed with the small amount of constraint syntax given). As far as could be determined, assessing relative intuitiveness of CSS language features had not been done in a way similar to the method presenting in this thesis previously. Therefore, the scope limited testing to a small scale and attempting to draw out findings that were suggestive meant that focusing on just a couple of concepts available with GSS was appropriate for this paper.

Like the question of how deep into constraint syntax should the paper, it was also difficult to determine the extent to which the workings of the mind should be investigated in ordered to support the categorisation of intuitive for constraint syntax. It could have gone much further than what was presented in this paper; however, just a few basic reading patterns that allow cognitive load to be inferred were identified. Further, in order to include these concepts in the experimentation, it was decided to use cursor movements as a proxy. This obscured the

results even more, especially since a large portion of respondents used trackpads which resulted capturing very little of the participant's reading pattern. Again, the scope can be used to justify this degradation. If the scope of similar experiments were to be grown with correspondingly larger budgets for such research, eye tracking systems and perhaps brain scans could be utilised to look more deeply and what participants struggled to understand. Within the current scope and despite the poor quality, the recorded cursor movements were able to show a large amount of time spent looking at GSS relative to the other code snippets as well as indicate two points of concentration within the code snippet on the novel features of GSS. This was enough detail given the scale of this paper and that it was supplemented with the think-aloud component of the experiment as well.

Altogether, the findings from this paper may have been made more substantial by digger further into the topics that are already presented here: further into constraint syntax and further into what can be termed intuitive based on how the mind works. Nevertheless, it is believed that within the given scope of the paper, the set objectives have been achieved. A process for testing the relative intuitiveness of various CSS language features was established and tested during the course of the thesis. This lead to sufficient evidence to suggest that constraint syntax, on a first-impression, is not intuitive relative to existing CSS layout approaches.

5.3 Future Directions

From its beginning to its upcoming standards, this paper has shown CSS is changeable (or at least extendable). Assessing the usability of such changes using think-aloud tests and some way to measure the user's gaze results, especially during a comparison of language features, reveals insights about how designers think about the code. Such insights may have avoided counter-intuitive behaviours, such as some margin-collapsing rules, which are now set in stone. Testing new CSS features in demographic groups around the world could help ensure future changes to this global specification simply make sense.

6 References

Alexa. (2016, July 13). *Site Overview csszengarden.com.* Retrieved July 15, 2016, from Alexa: http://www.alexa.com/siteinfo/csszengarden.com

Alexa. (2016, July 15). *Site Overview gridstylesheets.org.* Retrieved July 15, 2016, from Alexa: http://www.alexa.com/siteinfo/gridstylesheets.org

Alexa. (2016, July 13). *Site Overview w3.org.* Retrieved July 15, 2016, from Alexa: http://www.alexa.com/siteinfo/w3.org

Badros, G. J., Borning, A., Marriott, K., & Stuckey, P. (1999). *Constraint Cascading Style Sheets for the Web.*

Bednarik, R., & Tukiainen, M. (2006). An Eye-Tracking Methodology for Characterizing Program Comprehension Processes. *ETRA* (pp. 125-132). San Diego, California, USA: ACM.

Berners-Lee, T. (1991, October 29). Re: status. Re: X11 BROWSER for WWW. W3C. Retrieved July 16, 2016, from http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html

Berners-Lee, T. (1994-2006). What were the first WWW browsers? W3C. Retrieved June 18, 2016, from https://www.w3.org/People/Berners-Lee/FAQ.html#browser

Berners-Lee, T., & Connolly, D. (1992). Tags used in HTML. Retrieved June 18, 2016, from http://info.cern.ch/hypertext/WWW/MarkUp/Tags.html

Berners-Lee, T., & Connolly, D. W. (1995, November). Hypertext Markup Language - 2.0. IETF. Retrieved June 20, 2016, from http://www.ietf.org/rfc/rfc1866.txt

Blackwell, A. F. (2006, December). The Reification of Metaphor as a Design Tool. *ACM Transactions on Computer-Human Interaction, 13*(4), pp. 490-530.

Bootstrap. (2016). Grid System. Retrieved July 21, 2016, from http://getbootstrap.com/css/#grid

Bos, B., & Lilley, C. (2016, July 12). *Cascading Style Sheets (CSS) Working Group Charter.* Retrieved July 15, 2016, from W3C Interaction Domain: https://www.w3.org/Style/2014/css-charter

Burton, L. (1999). Why is Intuition so Important to Mathematicians but Missing from Mathematics Educations? *For the Learning of Mathematics, 19(3)*, 27-32.

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., . . . Tamm, S. (2015). Eye Movements in Code Reading: Relaxing the Linear Order. *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension* (pp. 255-265). Florence: IEEE. doi:10.1109/ICPC.2015.36

Busjahn, T., Schulte, C., & Busjahn, A. (2011). Analysis of Code Reading to gain more Insight in Program Comprehension. *Koli Calling.* Koli, Finland: ACM.

Can I Use. (2016, June). CSS Table display. Retrieved July 22, 2016, from http://caniuse.com/#feat=css-table

Can I Use. (2016, June). Flexible Box Layout Module. Retrieved July 22, 2016, from http://caniuse.com/#feat=flexbox

Clarke, A. (2014, March 25). *A Modern Designer's Canvas.* Retrieved June 12, 2016, from Smashing Magazine: https://www.smashingmagazine.com/2014/03/a-modern-designers-canvas/

Coyier, C. (2012, August 7). "Old" Flexbox and "New" Flexbox. *CSS-Tricks*. Retrieved July 22, 2016, from https://css-tricks.com/old-flexbox-and-new-flexbox/

Crosby, M. E., Scholtz, J., & Wiedenbeck, S. (2014). The Roles Beacons Play in Comprehension for Novice and Expert Programmers. *14th Workshop of the Psychology of Programming Interest Group* (pp. 58-73). Brunel University.

Ford, N. (2014, March 25). Content-out Layout. *A List Apart*. Retrieved July 26, 2016, from http://alistapart.com/article/content-out-layout

Grannell, C. (2013, June 4). Beyond mockups: how leading web designers work in 2013. *Digital Arts*. IDG. Retrieved July 24, 2016, from http://www.digitalartsonline.co.uk/features/interactive-design/learn-web-designs-new-ways-work/

grid. (2016). *Macmillan Dictionary*. Retrieved June 1, 2016, from http://www.macmillandictionary.com/dictionary/british/grid

Gridset. (2014). *Gridset.* Retrieved July 27, 2016, from Responsive Report: http://2014.report.gridsetapp.com/

Gridset. (2016). *Gridset*. Retrieved July 27, 2016, from Web Layout Evolved: https://gridsetapp.com/

GSS. (2015). *Constraint CSS.* Retrieved July 13, 2016, from Grid Style Sheets 2.0: http://gridstylesheets.org/guides/ccss/

Hansen, M., Goldstone, R. L., & Lumsdaine, A. (2013, April 26). What Makes Code Hard to Understand. Retrieved July 13, 2016, from http://arxiv.org/abs/1304.5257

Harper, S., Jay, C., Michailidou, E., & Quan, H. (2013). Analysing the Visual Complexity of Web Pages Using Document Structure. *Behaviour & Information Technology, 32, 5*, 491-502.

Hickson, I. (2004, December 22). Re: Collapsing 0 width margin. *W3C Mailing Lists*. Retrieved July 24, 2016, from https://lists.w3.org/Archives/Public/www-style/2004Dec/0087.html

Hogarth, R. M. (2001). Educating Intuition. University of Chicago Press.

HTML5 Please. (2016, May). Flexbox. Retrieved July 22, 2016, from http://html5please.com/#flexbox

Huang, J., White, R. W., & Buscher, G. (2012). User See, User Point: Gaze and Cursor Alignment in Web Search. *CHI*, (pp. 1341-1350). Austin, Texas, USA.

Ink. (2016). Grid. Retrieved July 21, 2016, from http://ink.sapo.pt/ui-elements/grid/

Jbara, A., & Feitelson, D. G. (2015). How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking. *23rd International Conference on Program Comprehension* (pp. 244-254). Florence, Italy: IEEE.

Khedker, U. P. (1997). What Makes a Good Programming Language ? Department of Computer Science, University of Pune.

Kistner, G. (2004). Why Tables Are Bad (For Layout*) Compared to Semantic HTML + CSS. Retrieved July 22, 2016, from http://phrogz.net/css/WhyTablesAreBadForLayout.html

Kramer, J. (2014, February 19). *Responsive Design Frameworks: Just Because You Can, Should You?* Retrieved July 8, 2016, from Smashing Magazine: https://www.smashingmagazine.com/2014/02/responsive-design-frameworks-just-because-you-can-should-you/

Kumar, R., Satyanarayan, A., Torres, C., Lim, M., Ahmad, S., Klemmer, S. R., & Talton, J. O. (2013). Webzeitgeist: Design Mining the Web. *CHI 2013: Changing Perspectives* (pp. 3083-3091). Paris, France: ACM.

Levine, M. (2006, January 30). In Search of the Holy Grail. *A List Apart*. Retrieved July 8, 2016, from http://alistapart.com/article/holygrail

Liang, H.-S., Kuo, K.-H., Lee, P.-W., Chan, Y.-C., Lin, Y.-C., & Chen, M. Y. (2013). SeeSS: Seeing What I Broke – Visualizing Change Impact of Cascading Style Sheets (CSS). *UIST* (pp. 353-356). St. Andrews, UK: ACM.

Lidwell, W., Holden, K., & Butler, J. (2010). *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach Through Design.* Rockport Publishers.

Lie, H. W., & Bos, B. (1996, December 17). Cascading Style Sheets, level 1. W3C. Retrieved June 22, 2016, from https://www.w3.org/TR/REC-CSS1-961217

Lilley, C. (1996, February 7). *Re: CNS colors.* Retrieved July 15, 2016, from W3C Mailing Lists: https://lists.w3.org/Archives/Public/www-style/1996Feb/0019.html

Lilley, C. (1996, February 28). *Re: T.E.O.'s Draft--Cascading Speech Style Sheets (txt).* Retrieved July 15, 2016, from W3C Mailing Lists: https://lists.w3.org/Archives/Public/www-style/1996Feb/0063.html

Macmillan Dictionary. (2016). layout. Retrieved July 24, 2016, from http://www.macmillandictionary.com/dictionary/british/layout

Merriam-Webster.com. (2016). graphic design. Retrieved July 8, 2016, from http://www.merriam-webster.com/dictionary/graphic%20design

Mozilla Developer Network. (2013, November 22). <spacer>. Mozilla. Retrieved June 30, 2016, from https://developer.mozilla.org/en-US/docs/Web/HTML/Element/spacer

Mozilla Developer Network. (2015, December 26). *Date.now().* Retrieved July 14, 2016, from MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now

MultiTree. (2014). Multitree: A digital library of language relationships. Bloomington, Indiana, USA: Department of Linguistics, The LINGUIST List, Indiana University. Retrieved July 28, 2016, from http://multitree.org/

Nielsen, J. (2000, March 19). Why You Only Need to Test with 5 Users. Nielsen Norman Group. Retrieved July 24, 2016, from https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Nielsen, J. (2005, July 11). *Scrolling and Scrollbars.* Retrieved July 14, 2016, from Nielsen Norman Group: https://www.nngroup.com/articles/scrolling-and-scrollbars/

Nielsen, J. (2012, January 16). *Thinking Aloud: The #1 Usability Tool.* Retrieved July 15, 2016, from Nielsen Norman Group: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/

Nielsen, J. (2014, September 1). *Demonstrate Thinking Aloud by Showing Users a Video*. Retrieved July 29, 2016, from Nielsen Norman Group: Demonstrate Thinking Aloud by Showing Users a Video

Open Stand. (2016, July 8). *Principles*. Retrieved from Open Stand: Global advocates for open standards & technology development: https://open-stand.org/about-us/principles/

Prowse, A. (2008, November 23). [CSS21] Negative clearance. *W3C Mailing Lists*. Retrieved July 26, 2016, from https://lists.w3.org/Archives/Public/www-style/2008Nov/0482.html

Raggett, D. (1997, January 14). HTML 3.2 Reference Specification. W3C. Retrieved June 20, 2016, from https://www.w3.org/TR/REC-html32.html

Raymond, M. (2006, July 13). Re: [CSS3 Color] Percentages in Alpha Value etc. *W3C Mailing Lists*. Retrieved July 26, 2016, from https://lists.w3.org/Archives/Public/www-style/2005Jul/0351.html

Rayner, K. (1998). Eye Movements in Reading and Information Processing: 20 Years of Research. *Psychological Bulletin*, 372-422. Massachusetts, USA: American Psyschological Association.

Reichle, E. D., Rayner, K., & Pollatsek, A. (2000). Comparing the E-Z Reader Model to Other Models of Eye Movement Control in Reading.

Resig, J. (2008, November 12). *Accuracy of JavaScript Time.* Retrieved July 15, 2016, from John Resig: http://ejohn.org/blog/accuracy-of-javascript-time/

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach (Third Edition).* Upper Saddle River, New Jersey 07458: Prentice Hall.

Shepherd, R. (2011, September 19). *CSS3 Flexible Box Layout Explained.* Retrieved July 8, 2016, from Smashing Magazine: https://www.smashingmagazine.com/2011/09/css3-flexible-box-layout-explained/

Siegel, D. (1996). Web Site Design: Killer Web Sites der 3. Generation. 110-111. (N. Schwarten, Trans.) Munich, Bavaria, Germany: Markt&Technik Buch- und Software- Verlag GmbH.

Simmons, J. (2015, December 5). Modern Layouts: Getting Out of Our Ruts by Jen Simmons. An Event Apart. Retrieved July 21, 2016, from https://vimeo.com/147950924

Simmons, J., & Andrew, R. (2016, April 3). Laying Out the Future with Rachel Andrew. *The Web Ahead*. Retrieved from http://thewebahead.net/114#transcript

table. (2016). *Merriam-Webster.com*. Retrieved July 1, 2016, from http://www.merriam-webster.com/dictionary/table

Toh, C. (2014, October 27). The Anti-hero of CSS Layout - "display:table". *Blog | Colin Toh*. Retrieved July 22, 2016, from http://colintoh.com/blog/display-table-anti-hero

Turner, R., Falcone, M., Sharif, B., & Lazar, A. (2014). An Eye-tracking Study Assessing the Comprehension of C++ and Python Source Code. *ETRA* (pp. 231-234). Safety Harbor, Florida, USA: ACM.

von Mayrhauser, A., & Vans, A. (1995, August). Program Comprehension During Software Maintenance and Evolution. *Computer*, *28*(8), pp. 44-55.

W3C. (2011, June 07). Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. (B. Bos, T. Çelik, I. Hickson, & H. W. Lie, Eds.) Retrieved June 22, 2016, from https://www.w3.org/TR/CSS2/

W3C. (2011, June 7). CSS Color Module Level 3. Retrieved July 9, 2016, from https://www.w3.org/TR/css3-color/

W3C. (2011, March 22). Flexible Box Layout Module. (T. Atkins Jr., A. Mogilevsky, & L. D. Baron, Eds.) Retrieved July 22, 2016, from https://www.w3.org/TR/2011/WD-css3-flexbox-20110322/

W3C. (2014, March 20). CSS Shapes Module Level 1. (V. Hardy, R. Atanassov, & A. Stearns, Eds.) Retrieved July 27, 2016, from https://www.w3.org/TR/css-shapes-1/

W3C. (2014, October 28). HTML5: A vocabulary and associated APIs for HTML and XHTML. (I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. Navara, E. O'Connor, & S. Pfeiffer, Compilers) Retrieved June 2016, 29, from https://www.w3.org/TR/html5/Overview.html

W3C. (2016, May 26). CSS Flexible Box Layout Module Level 1. (T. Atkins Jr., E. J. Etemad, & R. Atanassov, Eds.) Retrieved July 22, 2016, from https://www.w3.org/TR/css-flexbox-1/

W3C. (2016, May 19). CSS Grid Layout Module Level 1. (T. Atkins Jr., E. J. Etemad, & R. Atanassov, Compilers) Retrieved June 29, 2016, from https://www.w3.org/TR/css-grid-1/

W3C. (2016). *CSSWG Test Server*. Retrieved July 26, 2016, from CSSWG Test Server: http://test.csswg.org/

W3C. (2016, June 21). HTML 5.1. Retrieved July 22, 2016, from https://www.w3.org/TR/html51/rendering.html#the-css-user-agent-style-sheet-and-presentational-hints

W3C. (2016, July 15). *Mailing-lists Search service.* Retrieved from W3C: https://www.w3.org/Search/Mail/Public/advanced_search

Walton, P. (2016, July 19). Solved by Flexbox. Retrieved July 22, 2016, from https://philipwalton.github.io/solved-by-flexbox/

Wappalyzer. (2016, July 21). Web Frameworks: Websites using Web Frameworks. Retrieved July 21, 2016, from https://wappalyzer.com/categories/web-frameworks

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P., & Prasad, C. (2006). An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. *Eighth Australasian Computing Education Conference.* Hobart, Tasmania: Australian Computer Society, Inc.

Wilson, B. (2005). Multi Column. *Index DOT Html*. Retrieved June 30, 2016, from http://www.blooberry.com/indexdot/html/tagpages/m/multicol.htm

ZURB Foundation. (2016). The Grid. Retrieved July 21, 2016, from http://foundation.zurb.com/sites/docs/grid.html^

Appendix A: Uses of ‘Intuitive’ in the WWW-Style Mailing List

The table below presents the uses of “intuitive” found in the www-style mailing list. The mailing list archives are available at: https://lists.w3.org/Archives/Public/www-style/. A total of 934 emails containing “intuitive” were found (which may include duplicates as it is included in responses as well), an indication that the intuitions of people using CSS are considered in its design. The term is found in about 1.127% of the 82,838 messages present in the archive. This covers the time range from May 1995 until June 2016. However, only messages up to and including 2008 are analysed. The purpose of this table is to ascertain the reasoning behind determining whether something is intuitive or not by examining the context of its usage. Each usage is categorised into “opinion”, “logic”, “evidence” or “irrelevant” based on how the reference to something being intuitive is justified. For the most part, spelling and grammatical errors have been included as they were given in the original documents; however, autocorrect may have fixed the occasional word. To the interpretation of the author, usage of mailing list content is granted given the following statement appears: Copyright © 1995-2016 World Wide Web Consortium, (MIT, ERCIM, Keio, Beihang). http://www.w3.org/Consortium/Legal/2015/doc-license.

Message	**Year**	**Link**	**Categorisation**
…Howeverm I'm sure you can make something more *intuitive* to allow authors to change properties of capitals…	1995	Link	Opinion
… I would prefer the style sheet language to be a bit more *intuitive*. "@archform" isn't…	1995	Link	Opinion
…While coding, a couple of issues that I didn't resolve *intuitively* came up. I've described one of them below. Input is welcome….	1995	Link	Opinion
… The problem with this solution is <DOC> or <SOURCE> may appear as real HTML tags one day and the style sheet language will become ambiguous. If anyone has a clear vision of how one can get out of this with an *intuitive* notation in place, please let me know...	1995	Link	Opinion
…Yes, I like your [] syntax. It's much cleaner and more *intuitive* than what I came up with…	1995	Link	Opinion

…Whether the default properties are inherited or not depends on the property. Making it depend on the element instead is rather counter-*intuitive*…	1995	Link	Opinion
…I seem to understand you, by using this notation, to mean that if "[.]font[.]" were anywhere within the classification, then the current class should have its attributes cascaded from "font". I do NOT find it *intuitive* to say that…	1995	Link	Opinion
…So the notion of HTML_ELEMENT : SPECIFICATION would be better rendered as CLASS_NAME { SPECIFICATION } where SPECIFICATION may be a multi line set of elememts. This makes a complex element like the EM.first much more *intuitive* (and easier to parse too)…	1995	Link	Opinion
…Instead of making subclassing mandatory, the CSS proposal takes advantage of the existing "subclasses" (H1, P etc) and overlays styles based on these. For me, that's *intuitive*, and Peter's extra level of indirection is not. What do other people think? (I fear my intuition is somewhat damaged after thinking about these issues for some time :-)…	1995	Link	Opinion
…the value of the <STYLE NOTATION> attribute is apparently restricted to the values listed in the DTD. This seems counter-*intuitive* to me…	1995	Link	Opinion
…Or do you want separate properties for URL's, such as `text-background' vs `text-background-url'? The problem is that this may not be very *intuitive* either….	1995	Link	Opinion
…"*" is not a synonym for "HTML", it's a synonym for the top-level element. In HTML, this happens to be "HTML", but many authors omit it and I believe "*" is more *intuitive*…	1995	Link	Opinion
…While CLASS would be "usable", it is not the least bit *intuitive*…	1995	Link	Opinion
…My guess is that the most *intuitive* reading will be that everything is relative to its enclosing environment…	1995	Link	Opinion

…Then there are some not-so-obvious factors like how "mature" a property is, how "*intuitive*" it is and how "useful" it is…	1995	Link	Irrelevant
…True, and even more so, I feel future authoring tools need to present the stylesheet functionality in an *intuitive* manner. That will, I believe, influence authoring styles more than anything else…	1995	Link	Opinion
…#x65y or "x56y"? #x65y. Definitely. Using quotes in a way that has a syntactic meaning (other than encapsulation) is pretty anti-*intuitive*…	1995	Link	Opinion
…For font-weight and font-size, I appreciate that you've moved from absolute numbers to relative ones. I'm a little concerned, though, that it may not be *intuitive* that a bare positive number means an increase…	1996	Link	Opinion
…I know which printer fonts link to which screen fonts - but a name searching system is going to be more cumbersome and less *intuitive*…	1996	Link	Opinion
…Obviously CNS (which I'd never heard of before) is a subset of the HSB (hue-saturation-brightness) color model, widely used and intuitive…	1996	Link	Opinion
…In particular, it has nothing to do with HLS, HSB and suchlike polar representations of RGB (which are, in usability studies, often shown to be *not* very intuitive) …	1996	Link	Evidence
…HSB is a spectacularly bad idea as it is non intuitive. It claims for example that yellow (RGB 00FFFF) and blue (RGB 00FFFF) have the same "brightness" which is clearly false. It is extremely non linear, the hue circle is not at all even .. generally, it is a mess.…	1996	Link	Logic
…The tags and have more intuitive meanings than <i> and <b> for people who work aurally rather than visually…	1996	Link	Logic
…This seems strongly counter-*intuitive*. The default is that there is no sound? Perhaps a stylesheet for visual presentation could specify that the default is black text on a black background, so the screen is entirely dark? ...	1996	Link	Logic

…I do not discuss physical values for these attributes, as these cannot be translated as simply to values of multimodal attributes as can the less precise (but more *intuitive*) natural language or numerical values…	1996	Link	Opinion
…Space should be tied to the visual attribute of padding or margin; I picked padding, but I think that either could be chosen. Possible mnemonics: none (a bit counter-*intuitive* at 1) | narrower | narrow | normal | wide | wider | widest…	1996	Link	Opinion
…Physical values obviously should be allowed, to give authors detailed control over document formats; however, the allowed values were selected to be as useful and *intuitive* as possible, to encourage casual authors to use them rather than physical values…	1996	Link	Opinion
…allowing numbers, e.g. [1-7], to be used to represent the values of multimodal attributes for which this procedure seems to be *intuitively* reasonable…	1996	Link	Opinion
…as authors are probably more likely to prescribe visual style than audio style. In this case, the name would be a bit less *intuitive* than before…	1996	Link	Opinion
…are also useful goals for unimodal styling language designers (as are the first and seventh, but they are so *intuitive* that they are almost universally followed) …	1996	Link	Opinion
…The "immediate predecessor" idea Hakon brings up would seem more intuitive to me using '+' …	1996	Link	Opinion
…The user is presented with a color spectrum arranged in a circular pattern, with a slider bar beside it to specify the lightness/darkness of the particular color chosen. This system is simple, *intuitive*, and generates RGB codes without forcing the user to delve into the wonders of hexadecimal…	1996	Link	Opinion
…it may not be correct, but it is simple and *intuitive*…	1996	Link	Opinion
…that is not how MSIE applies the style properties applied via a STYLE attribute (anything inside a STYLE attribute on an element is immediately applied to the element, before anything else. Otherwise, it could be overridden by a rule	1996	Link	Opinion

with a higher specificity (e.g., "#ID1 #ID2 {}"), which doesn't seem *intuitive* at all)…			
…The major problem with SPACER, outside of any discussion of compliance with standards and Netscape vs. Microsoft, is that it's not *intuitive*. If one thinks of it like a blank image, you expect to give it WIDTH and HEIGHT attributes. But this only applies if you choose TYPE=block…	1996	Link	Logic
…The main difficulty of learning the X resource system is that very few people have a good *intuitive* understanding of what a widget is, or a window. On the other hand, most people have a very good *intuitive* understanding of what a hierarchy is, and they also understand the meanings of text, background, shadow, and many other words employed by CSS…	1996	Link	Opinion
…I like to think of leading as added space after the line. I've never seen it referred to as space above and below the line. Its counter-*intuitive*…	1996	Link	Opinion
…Also, we are not changing the rules for comments – this is all within the bounds of SGML. (I, like many other people, find those rules to be less than *intuitive*, but there we are) …	1997	Link	Opinion
… My feeling is that left/right is a little ambiguous while front/back is a bit more *intuitive* and less ambiguous…	1997	Link	Opinion
… Really, they're just two different interpretations of how intrinsic HTML support is handled, but I believe IE4's is much more *intuitive*…	1997	Link	Opinion
… The only evidence at the moment is that two authors confronted with the same problem found the same solution *intuitive*…	1997	Link	Opinion
… This is a simple solution useful in simple situations, syntactically no less *intuitive* than the shorthand, yet functionally a superset…	1997	Link	Opinion
…a conceptual change that adds an *intuitive* bit of functionality at no practical cost…	1997	Link	Opinion

… Not everyone will find it *intuitive* to lump default properties and user-defined attributes…	1997	Link	Opinion
… Should the new value be clipped (if this is necessary) before or after it is passed on to children? It is easier to clip first, but I think that this is the less *intuitive* interpretation…	1997	Link	Opinion
… If I'm understanding you, all that you want is a shorthand for the left, top, width, and height properties. This might not be a bad idea, but I think as an author it's simpler and more *intuitive* to specify these properties separately…	1997	Link	Opinion
…Of course, if I actually cared to contribute to the debate, I would point out that the issue of RGB vs HSL vs YUV vs CMYK is irrelevant, since _ALL_ of those color models are counter-*intuitive* and generally bogus, from the non-techie-nerd's point of view…	1997	Link	Opinion
…"Teach Yourself Web Publishing with HTML 3.2 in 14 days" (SamsNet, 1996): "The Hue, Saturation, and Brightness model is sometimes called the subjective or perceptive color, because this model *intuitively* describes how we preceive color and changes from one color to another." I just spent 15 months at Carnegie Mellon University getting my Masters in Human-Computer Interaction and color models were discussed in several classes. The Hex numbers required by RGB were never *intuitive* and the only way to adjust them was by random trial and error unless you had a color picker tool. On the other hand you could make changes to an HSL value and see the color change in the direction desired…	1997	Link	Evidence
…RGB hexadecimal is what we designers call the RGB notation that you are referring to. Yes - it is far from *intuitive*! …	1997	Link	Opinion
…Unfortunately HSL is not *intuitive* either. It requires that one know all the hue values…	1997	Link	Logic
…In truth, I and many other designers I know can tell approximately what a color looks like or is with plane #RGB values. I can also look at those values and know if it will	1997	Link	Opinion

work on a NTSC display or "WebTV" HSL can not provide me with that intuitive knowledge… … Don't use HSL, please it is far from *intuitive* for a designer. We might as well just have the RGB hex…			
…I'm not convinced your examples are *intuitive*, but something along those lines would be better than unmatched bracketing with forward slashes and tildes with context-sensitive meanings…	1997	Link	Opinion
… Both of these notations are simple transformations of the RGB color space, but represent more *intuitive* spaces for general use by the average non-technologist. For example, consider the following activities: …	1997	Link	Opinion
…specifying RGB colors is non-*intuitive* for people who do not have significant experience with RGB or computers…	1997	Link	Opinion
…I urge you to read the archives of this mailing list [1] for previous discussions about the non-intuitive nature of HSL (and HSV) … The corresponding activity, of changing the hue but keeping the lightness the same, does not give the intuitively expected results using HSL…	1997	Link	Opinion
…In my own work, I've always included HSL because it is significantly more *intuitive* that RGB…	1997	Link	Opinion
…I think that part of the problem is that separating document structure from document display is not a process that is *intuitive* for many people…	1998	Link	Opinion
… I confess I was wrong to refer to them as positioning properties--that's just *intuitive*…	1998	Link	Opinion
… The selector syntax is already getting complex… can we come up with *intuitive* syntax?	1998	Link	Opinion
…One idea would be to examine the different ways it is implemented and find the most *intuitive* one. It has to be said that so far, the attr selector system hasn't been the most *intuitive*…	1998	Link	Opinion
…Remember, the main advantage of HTML is that it's simple enough to teach to almost anyone, and most of the tags are their own mnemonics. Style sheets are slightly more	1998	Link	Opinion

complicated, but the properties are generally named after the effects they (should) create, so they're easy to understand and remember… Regexps are just about the exact opposite… (This is where I have trouble; my *intuitive* side keeps getting in the way and I lose concentration.) …			
…My two thoughts so far were to link from the title-- not necessarily *intuitive*-- or adding a "[Spec]" link, which isn't much more obvious…	1998	Link	Opinion
…Also, each property page has a link to the appropriate part of the specification, with a "[Spec]" link in the nav bars. I'm still taking any suggestions for ways to make this more *intuitive*…	1998	Link	Opinion
…As someone who is in the middle of translating a fairly simple *intuitive* ordering algorithm over ISO Latin-1 from hand waving into computer code I know just how important (and difficult) these things are…	1998	Link	Opinion
… (Since the default value of background-color is transparent, setting { background: white } or whatever for the OBJECT where the text/html object itself doesn't set a BODY background-color will have the *intuitive* effect.)…	1998	Link	Opinion
…So now the background-color of the parent of the OBJECT would shine through. (At least in the way I've always interpretted (sic) it to work – which seems fairly *intuitive* to me.) …	1998	Link	Opinion
… say that background-attachment on inline elements is relative to the containing block (which is CSS2-speak for in most cases) the "parent element") No. This is counter *intuitive* at best…	1998	Link	Opinion
… I would guess that the first is what is closest to the original intent. It's the easiest to implement, and the most *intuitive*…	1998	Link	Opinion
… The effect was that the interpretation of 'border: medium red' changed from producing a solid red border to producing no border at all. Not very *intuitive* maybe, but consistent with its new role as a shorthand property…	1998	Link	Opinion

…This anonymous box also has the font properties of the block-level element, so all other inline boxes are vertically aligned within it. The above makes the spec much closer to the '*intuitive*' expectations…	1998	Link	Opinion
…nest your DIVs, add a class "section" to them and use the following rule: DIV.section { margin-left : 2px } which is IMHO simpler and more *intuitive*, works on existing browsers, but implies a rewriting of your document in a more structured way…	1998	Link	Opinion
…The CSS1/CSS2 approach is the more *intuitive* approach for replaced elements, and the IE approach is the more *intuitive* for non-replaced elements…	1999	Link	Opinion
…I'd suggest that it makes the most sense to put the columns on "bottom" with row groups and rows on top of them. While this proposal is largely arbitrary (it just seems most *intuitive* to me this way), a possible rationale is that COL and COLGROUP come before the rows in the table description…	1999	Link	Opinion
…Should the background image cover only half the element (the right one), or should it cover it all? IMO controlling the position of the tiling boundary while covering the whole element is more important/useful/*intuitive* then covering just a part of the element, but the CSS2 specifications aren't clear on this…	1999	Link	Opinion
… This means that paragraph's can't be 'backed up' on top of previous elements and have the background overlap previous content. Too bad, as that 'might' be useful (and maybe more *intuitive*?) …	1999	Link	Opinion
…It is also far more *intuitive*, and I cannot see any area in which the existing spec is better…	1999	Link	Opinion
… Letterspace and word space are common terms. Linespace seems most intuitive…	1999	Link	Opinion

… The zoom value would impy a zoom not only on font stuff, but also on images, vector graphics, etc. I think this is a more *intuitive* and reliable approach then deferring to the CSS cascade and hope for the use of percentages, ems, or other relative sizing…	2000	Link	Opinion
…'text-align' positions the HR horizontally within the available space. This is sort of intuitive, unless you're a CSS expert, in which case it is quite confusing…	2000	Link	Opinion
… That's counter-intuitive to me, since overlapping elements generally overwrite previous elements in the flow…	2000	Link	Opinion
…I see not reason to have an attribute like box-sizing. It seem counter-*intuitive* to change the definition of sizing…	2000	Link	Opinion
…The basis of my argument was that 'box-sizing' is not an *intuitive* solution, and this is a better solution…	2000	Link	Opinion
…No, the box-sizing thing doesn't do what I want. The problem with it is that it is not intuitive. Say you are trying to explain this to someone new…	2000	Link	Opinion
…The basis of my argument was that 'box-sizing' is not an intuitive solution, and this is a better solution. What isn't *intuitive* about it? …	2000	Link	Opinion
… And yes, I am advocating "changing the rules" of CSS because, as I have repeatedly said, they are counter-*intuitive*. There should ABSOLUTELY be a way to specify the ENTIRE WIDTH of a box WITHOUT having to resort to an ugly hack like "box-sizing: border-box"…	2000	Link	Opinion
…IE5 renders the second one in what is, IMO, a most *intuitive* way; it uses the height it /can/ calculate (from the other cell) as the basis for 100%...	2000	Link	Opinion
…Names of properties are clickable and colour-differentiated so navigation is *intuitive* and easy…	2000	Link	Irrelevant
…The number of columns an element with 'column-span: none' is split into is the number specified for column-span in the *next* element that doesn't have 'column-span: none', or	2001	Link	Opinion

all remaining columns if there isn't a spanned element before the end of the multicol. I'm not saying this is inconsistent, it just didn't seem very *intuitive* at first…			
… Is there any reason not to include a pattern language into CSS… A standards committee could no doubt come up with something cleaner, more pleasing to the eye, and more *intuitive* to use…	2001	Link	Opinion
…A good language implements a small set of principles that fit naturally to the domain and that can be combined in an *intuitive* way to express what one wants to express…	2001	Link	Opinion
… I totally agree ! cron-style notation is by far easier to read. I also think it's generally far more *intuitive* than the an+b notation…	2001	Link	Opinion
…Putting a web author in front of nth-child(2,5,8-11) and nth-child(1,2-*/3) didn't bring up a strange face. It brought up a "Cool !" and he could figure out what it meant immediately. Otoh, nth-child(-5n+6) didn't seem to be as *intuitive*. Linear sequences aren't hard to understand (at least in France everybody has been through them, I don't know about other educational systems), but for many people it's far behind…	2001	Link	Opinion
…so counting would be done from behind if the *first* number is negative. Unfortunatly the last two rules aren't very intuitive (but IMHO more logical then the current meaning of -3n+1) …	2001	Link	Opinion
…However, this is IMO more *intuitively* addressed by using the range proposition in a second selector: :nth-child(3n):nth-child(1..15)…	2001	Link	Opinion
…Measuring a percentage value for "left:" from the right edge of the screen is inconsistent with the *intuitive* behavior when using a pixel value. left:30px *intuitively* means start at 30px from the left edge of the screen…	2001	Link	Opinion

In trying to implement his tool he has discovered that the de facto browser implementations of transparency are not of the *intuitive* form he expects	2001	Link	Opinion
The obvious goal of these media rule tricks is to select rules based on the CSS version supported. Why not allow authors to do the same thing in an *intuitive* way?	2001	Link	Opinion
I believe that @media rules and other non-standard ways of selecting the CSS version are more bug-prone, less intuitive, and generally "worse" than a standard @version rule.	2001	Link	Opinion
…I did the "granny test" with this one. The result: it's not at all *intuitive* to people that scroll bars remain scroll bars after they change color...	2001	Link	Opinion
…Judging from the discussion here, people don't fully understand how the spec deals with centering elements and the size of the top-level element. "margin:auto" is non-*intuitive*, and it behaving differently horizontally and vertically is downright confusing...	2001	Link	Opinion
…This is even more off-topic than the original post, but I feel obliged to point out that, as counter-*intuitive* as it might seem, closed-source products have occasionally been successful in mass markets…	2001	Link	Irrelevant
… I am fully convinced of W3's method of using a value from 0 to 1. It seems neither easy (*intuitive* as in 0 to 100%) or exacting (as in the standard 256) …	2001	Link	Opinion
…Curious, why do you need to set the outer columns right, left to 75%? That does not seem *intuitive* to me…	2002	Link	Opinion
…It is easy to set "width: 100%;" or "height: 100%;", difficult / not *intuitive* to use margins to determine the width / width, especially to center a block. Don't waste your time trying to figure out how to vertically center one block within another, you can't do it period using margins…	2002	Link	Opinion
… I am a bit doubtful that CSS1 compatibility and *intuitive* behavior with an exposed counter can be achieved simultaneously…	2002	Link	Opinion

I can't see any way to define behavior consistent with that of the unexposed counter that won't be clumsy and *unintuitive* when combined with the new, exposed counter.	2002	Link	Opinion
"RGB is oriented to light rather than (what people find more *intuitive*) print. For instance, yellow is red+green in RGB… RGB is non-*intuitive*. People can learn how to use RGB, but actually by internalizing how to translate Hue, Saturation and Lightness, or something similar, to RGB. "	2002	Link	Opinion
…My vote is with Joe on the hanging indents. Yes, they can be done with CSS1, but the method is counter-*intuitive*…	2002	Link	Opinion
…Since 1996 there have been numerous proposals to improve the named colors to be more intuitive, or to allow one to specify the naming scheme used…	2002	Link	Opinion
…Well, the HTML colours almost pass the "high school" test, with the addition of orange. I know it sounds simplisitic, but with orange added they do make a pretty good "intuitive" base set…	2002	Link	Opinion
…Some folks don't find RGB *intuitive*, and find (at least some of) the color names more *intuitive*. The addition of HSL colors should help as well, as its use seems much more *intuitive* than RGB…	2002	Link	Opinion
…I respectfully disagree with the suggestion to do away with named colors. While one may become accustomed to using number values when creating Web content, it is neither *intuitive* nor easy to maintain…	2002	Link	Opinion
…But I have to memorize or look up tables; with a color naming system like the one you once recommended [2], I would only have to memorize around 20 keywords, declare no entities, start up nothing than my text editor, and get a nice range of colors, readable, *intuitive*, convenient…	2002	Link	Opinion
… The margin/border/padding values then continue to mean (in an *intuitive* sense) what they mean in any other context…	2002	Link	Opinion
…XSL is already in XML format, while CSS has it's own unique one. I find the syntax of CSS to be much much more	2002	Link	Opinion

intuitive for what it does, and would strongly object to it being rewritten in xml. Apasrt from anything else it would be much more verbose, and harder to learn. I also believe it'd be less *intuitive*, create much larger file sizes than necessary, and of course it wouldn't work in current browsers which already do a good job interpreting it as it is…			
…I see two schools of thought : the style sheet syntax must be terse and *intuitive* for human conception or the style sheet syntax should be in XML to be easily processable, verbosity is of no consequence since style sheets are more and more produced automatically or through an GUI interface (my case at least)…	2002	Link	Opinion
…How about "both *intuitive* syntax *and* straightforward processability are important"? …	2002	Link	Opinion
…The :hover state is a useful tool to provide visual indication to the user that an element accepts input from a pointing device such as a mouse, and the :active state is useful for indicating that a particular element is currently in the activation state. These visual indications are naturally intuitive to the user…	2002	Link	Opinion
…Imagine, if you will, someone new to web design. What would be more *intuitive* for them to do? Add an empty div, or use a defined CSS property like the proposed float-overflow which says exactly whose name implies the exact desired effect?...	2002	Link	Opinion
… While the above examples are contained in the margin, there might also be another reference to a *picture*; which might be wider than the margins, so that the text would have to flow around it. None of this is particularly unusual layout; and --- at least to me --- it seems very much like the concept of a "float"; yet, as far as I can tell, neither the float model, nor anything else in CSS, could be used in an *intuitive* way to generate this presentation. What I draw from these examples (others' and mine) is that a "float" (as one would think of it *intuitively* for layout purposes) has both a position in the flow and a container	2002	Link	Opinion

…Consider the definition I suggested elsewhere in this thread: A "non-CSS presentational hint" is information which is derived from the document and is translated into CSS properties by the user agent through some mechanism other than CSS style rules… Is this definition reasonable, and reasonably *intuitive*? …	2002	Link	Opinion
…how do I collapse the section to show only the heading? Here is a solution, but it is not a very intuitive one: section:0 > h {display: block} section:0 > * {display: none}…	2002	Link	Opinion
…Let's use strings to represent the characters that correspond to normal keys, let's keep keywords for special keys, let's use whitespace as the <key> separator, and let's use the comma as the <key-press-combination> separator. Incidentally, this makes dealing with the "space" key more *intuitive*: it's simply " "…	2002	Link	Opinion
…Nonetheless, that they would need to do this for the quirky behaviour to make sense seems to indicate that perhaps a more *intuitive* solution exists--and should be used…	2002	Link	Opinion
…padding, border-spacing: 1em 2em; would have very *unintuitive* results. Authors are allowed to express information in a form that is *intuitive* to them. It is a valid declaration, but behind those property names is a detailed description that sufficiently describes what role the values to take on.	2002	Link	Opinion
… My intuitive bet is that 90+% accurate algorithms probably already exist, even we aren't aware of them…	2002	Link	Irrelevant
… The transformation into ACTUAL values of the style of the DOM, is owned by the View, but since it is one-to-one correspondence to the CSS-OM, then exposing that properties (also) in DOM using OO techniques is a convenience and *intuitive*…	2002	Link	Opinion
…I agree that the call should be very simple and *intuitive* to do simple things, just not that it is anywhere close to 1:1 if	2002	Link	Opinion

you expect to expose information on the view and formatting. …			
…If someone wanted all methods to be in View, so that it is not possible to do node.Property, then I have no major qualms with that. I just think it is more *intuitive* to stick with the DOM hierarchy. In the general sense, this is a false *intuition*, because it involves questions that cannot be properly answered…	2002	Link	Opinion
…What CSS2 has, is what seems the most intuitive to me. Is the change in CSS3 a mistake or have they changed the behavior on :first-line from CSS2 to CSS3? …	2003	Link	Opinion
…Let me try and make it more concrete with two examples: Example 1: unintuitive(?) boxes…	2003	Link	Opinion
…The text-height property of CSS3 allows the second case to render the way i see as more *intuitive*, using the value max-size…	2003	Link	Opinion
…I am uncertain about the syntax, things like foo:lang(), foo:lang(""), foo:lang(-), foo:lang(none()), etc. aren't that intuitive…	2003	Link	Opinion
…A better workaround than using tables in many cases is using something like ... , and styling the span. I do think that there should be a more intuitive way of doing this though…	2003	Link	Opinion
…Christoph, who thinks "position: relative parent 1em 2em 1.5em 1.5em;" would be [more] intuitive…	2003	Link	Opinion
… (There's also the fact that having a property named "glyph-orientation" reorder content instead of just rotating glyphs is IMO just not *intuitive*.) …	2003	Link	Opinion
…I can't use CSS-P because I have content that sits under the columns and must be automatically positioned underneath which ever column is longest. Plus the CSS-P approach is counter-*intuitive*…	2003	Link	Opinion
…John suggested applying CSS table syntax to the three-col problem. This is a solution, but the content is not tabular data	2003	Link	Opinion

and presenting it as such is a non-intuitive, if effective, method for layout…			
The host of other acronyms aside, that's what /works/, and that's how web pages /have/to/ be written if they are addressed to a general audience. But these languages fail to address many basic web layout problems in a direct, simple and *intuitive* way.	2003	Link	Opinion
…If we want to get people to stop using depreciated HTML attributes like "align", having **intuitive** css equivalents makes sense…	2003	Link	Opinion
…This is a special case of the more general parameterization case discussed a few weeks ago. One of the problems is that it is likely to invovle non-intuitive interactions with cascading rules…	2003	Link	Opinion
…Looking at the syntax closely, I see one must define an age in order to use a generic voice (eg, voice:family: child male), so I can conclude 'announcer' is intended as a specific voice name. This doesn't seem very *intuitive*…	2003	Link	Opinion
…I fail to see how it would make that task _more_ difficult if @import rules had their *intuitive* meaning and no artifical restrictions…	2003	Link	Opinion
…Didn't someone once say that CSS brought an elegant solution that replaced kludgy HTML tables? Yes this certainly is worth addressing. Why isn't there a simple way of doing this in CSS? Neither approach to center is straightforward (the margin: auto; approach for horizontal center is far from *intuitive*) …	2003	Link	Opinion
…Some of the things that people seem to want, like liquid layouts as good as handcrafted layouts, that work whatever the display technology and user preferences and overrides are still research topics. Especially if you also want them to be *intuitive* to an 18 year old arts student…	2003	Link	Opinion
…Many people using margin: auto for centering. It is not an obscure feature, indeed it appears in several CSS FAQs and	2003	Link	Opinion

Wikis, and while I agree it is not *intuitive*, it is not complicated either…			
…It is also non-*intuitive* to have a block display:table-cell outside of another block display:table-row and so on…	2003	Link	Opinion
…But it's less obvious what's natural if the picture is on the left. My *intuitive* feeling is that the two cases would differ more visibly…	2003	Link	Opinion
…My personal opinion is however that positioning outside by border-edge is slightly more *intuitive* and yields slightly better results in some edge cases…	2003	Link	Opinion
…I don't think that the way the definition makes colors of the underline work is very *intuitive* either, probably because i see underline and friends as text-features rather than box properties of the ancestor setting the text-decoration? …	2003	Link	Opinion
…It's really much more *intuitive*, to me, to put it on the DocumentStyle interface or extension thereof…	2003	Link	Opinion
…I think the behavior specified here for 'scroll' is counter-*intuitive*. As an author, I would expect the background to scroll with the element's content just like it does for the <body>…	2003	Link	Logic
…Constraining the background to the padding area would allow CSS to define "background-attachment: scroll" more *intuitively*… By locking the proposed clarifications of "background-attachment: scroll" and the background's boundaries together, we get a background model that is both more consistent and more *intuitive* than the one drafted in CSS2.1… The behavior of "scroll" would be most intuitive if the background scrolled with the content, as it's called "scroll" and as is how the setting behaves when specified for the main canvas…	2003	Link	Opinion
…This leads to mostly *intuitive* results when writing documents and marking them up with CSS…	2003	Link	Opinion

…In the meantime, it's quite hard to match against colonized names, which are frequently used in XML. It's certainly not *intuitive*…	2003	Link	Opinion
…Authoring tools could provide an *intuitive* interface for XHTML+CSS, which has the technical potential of replacing the what-you-see-is-not-what-you-want text processors we use today…	2003	Link	Irrelevant
…In my opinion the path CSS3 takes is to keep (and expand) the model but make most things any browser currently implements defined through a property. IMVHO this might be a misstake, since the model is not quite good enough (not because it doesn't work, but because it is to difficult and thus introduce differences and has what i see as *unintuitive* parts)…	2004	Link	Opinion
…In addition to what i see as issues with the implementations of the inline rendering, i think the model is not quite *intuitive* compared to the box model used for blocks…	2004	Link	Opinion
…Also, having the height of a non-replaced box being defined by the font(or fonts, a 'should' changed into a 'may' which i dont think the major browsers implement, except IE due to using a different model) of its textual contents seem *unintuitive*…	2004	Link	Opinion
…i'm just trying to say that it seems more *intuitive* to me that the link box covers the area which you can click to activate it, not claiming that the current model stops links from working…	2004	Link	Opinion
…There's no guarantee that the document type will have a language attribute. CSS has to deal with more than just HTML. I agree it at least seems more *intuitive*, though…	2004	Link	Opinion
In that case the specification is so counter-*intuitive* as to be dangerous (it encourages naive authors to default line-height, as a relatively obscure property …) …	2004	Link	Opinion
…It's counter-*intuitive* because people expect to be able to control font-size using only font-size properties…	2004	Link	Opinion

…This effectively negates the specified inherit default on line-height, but line-height is the more technical parameter, so is where any counter-*intuitive* behaviour should go…	2004	Link	Opinion
…The reason i prefer to view the content value as replaced is probably that i dislike the concept of having a pseudo-document-fragment in the content value, but also because it seems to create more *intuitive* results in most sane cases…	2004	Link	Opinion
…If replacement is set, content computes to something appropriate (there is precedent with the way float affects display, eg). I _think_ that's more *intuitive* for authors…	2004	Link	Opinion
…I find this idea interesting, and potentially useful, but I don't think %% is a very *intuitive* unit….	2004	Link	Opinion
…And if anonymous elements do count, the relative positioning is far from "*intuitive*"…	2004	Link	Opinion
…Overall I believe something like 'previous', 'next' and 'different' would be more useful, more *intuitive* and more portable than absolute integer indices…	2004	Link	Opinion
…Frankly this makes my head spin. Intellectually I can see that each of these does a distinct useful thing, but trying to *intuitively* grasp which one to use in a document would take a lot of practice…	2004	Link	Opinion
…I'm sorry but i don't understand that having those two definitions gives more *intuitive* results, and it seems that Mozilla 1.7, Opera 7.50 and IE6 disagree so much on this behavior that i cannot make sense of what that point is…	2004	Link	Opinion
…Sounds to me more *intuitive* than the current draft, if nothing else…	2004	Link	Opinion
…body { background: url(foo), url(bar); } seems more *intuitive*: you pass a list of items to something…	2004	Link	Opinion
…IMO a more intuitive way to do the "9-area" button is with just one image…	2004	Link	Opinion
…Using multiple, indexed attributes is advantageous over the proposed comma-separated list as it permits use of the background-xxx attributes in a more *intuitive* manner…	2004	Link	Opinion

…I feel if would be preferable to declare backgrounds with indexes and make this sort of referencing more intuitive…	2004	Link	Opinion
…I've scratch my head about a rendering problem and it turned out that most browser (I've tested Mozilla and Konqueror, IE is said to do the same) collapse the margin of the last element and the margin of it's parent even if the parent has a zero width margin. I find it weird and counter *intuitive*…	2004	Link	Opinion
…Personally i find the whole concept of nested collapse (and even more so, collapse-through) unintuitive… Note: Ian Hickson replied to this noting that margin collapsing is confusing but too late to change here.	2004	Link	Opinion
…The only option here is *Intuitive* UI. Make it semantic, don't tell it…	2005	Link	Irrelevant
…There's no *intuitive* way to do that. At least !required is a timeless syntax that does not depend on the author…	2005	Link	Opinion
…in the case of 'scroll', the background does not scroll with the element's content. This seems counter-*intuitive*…	2005	Link	Opinion
…Once the existence of a universal authoring tool is postulated, one can only too easly dismiss any suggestion which is intended solely to make the work of an author simpler and more intuitive…	2005	Link	Opinion
…:alt and its related suggestions seems a bit odd and un-*intuitive* to me…	2005	Link	Opinion
…The above @require-all-properties { } would cause failure on superfluous properties (or cause the author to split their CSS styles into less intuitive blocks, I think)…	2005	Link	Opinion
…How would this work? !exclude? That seems kinda counter-*intuitive*…	2005	Link	Opinion
…I agree that you do gain some clarity from the block naming there, but I don't think it's as *intuitive* to use…	2005	Link	Opinion
…My reservation remains as before, that consciously thinking about which styles /don't/ matter (so as to exclude them from a @require-all block) is more difficult and less *intuitive* than thinking about which styles /do matter/…	2005	Link	Opinion

…However, I do still think that specifying a particular property as being !required within that block (opt-in) is more *intuitive* than doing the reverse…	2005	Link	Opinion
…I found Mikko Rantalainen's !not-required idea very *intuitive* indeed…	2005	Link	Opinion
…Computed values can not be uniquely identified by selectors, just like you described. Moreover, I also believe referencing rules is more *intuitive* to the CSS designer…	2005	Link	Opinion
…It seems to be an *intuitive* author feature as suggested by the amount of usage of legacy HTML constructs such as and <td align="..">…	2005	Link	Opinion
…It very frustrating as someone who has to work with these standards every day. They are not *intuitive* and overly complex. I came here to try and understand…	2005	Link	Opinion
…I think the discussion has fallen in to a religious debate about whether your proposal is "easier" or more "*intuitive*" than the equivalent CSS, which is an unwinnable, irrelevant (and terribly uninteresting) debate IMHO…	2005	Link	Opinion
…Thought: "opacity" should support percents and floats, while rgba() would support percents, floats and integers? Not consistent, but more *intuitive*…	2005	Link	Opinion
…I can kinda see their reasoning for having floats for alpha, since integers aren't *intuitive* for opacity…	2005	Link	Opinion
…Then again, if they allow all values for opacity, and just accept that integers, being counter*intuitive* for opacity, are going to be little used, then we'd probably be fine…	2005	Link	Opinion
…As typical J2EE developers that have used CSS for years, with most of our knowledge coming from simple online tutorials, and online references, but having never really studied CSS from front to back we used the following "*intuitive*" approach…	2005	Link	Opinion
…however, it still applies because for some authors it is more *intuitive* to say, position based on element x or element y…	2005	Link	Opinion

…This would resolve the ambiguity squarely in favor of IE/Win and Safari; however, I think this resolution makes the most *intuitive* sense…	2005	Link	Opinion
…In CSS percentage has always been relative to the whole width (or height) of the containing block. I find it less *intuitive*.	2006	Link	Opinion
… :not(foo[bar]) could be written as :not(foo):not([bar]), but that's not very intuitive for authors…	2006	Link	Opinion
…Our aim has been to provide a syntax and semantics which would be as *intuitive* to web designers as possible…	2006	Link	Opinion
…Coupled with max-width and min-width this is pretty *intuitive* to use…	2006	Link	Opinion
…When it comes to zooming images, however, the "scaling" property seems more intuitive to me…	2006	Link	Opinion
…Instead, the borders, background and contents are stacked according to the complex rules laid out in Appendix E, dispersed among descendants with different, specified stack levels. I had a feeling it would have been too much to ask to have an *intuitive* and logical stacking system…	2006	Link	Opinion
…One of the solutions I considered for something like this (because it is un-intuitive right now) was to let percentages of min-height and max-height be calculated from min-height or max-height of the containing block if height is auto…	2006	Link	Opinion
…Are the numbers right-aligned, or are they in the center of the column? Positioning them in the center of the column would already be more intuitive, because now I have the option to align my column header neatly above the values in that column…	2006	Link	Opinion
…It avoids the horizontal/vertical direction coupling problems of direction. It's also more *intuitive* than the direction property imo.	2007	Link	Opinion
…I feel that up/down/left/right is more *intuitive* for glyph orientation than specifying an angle…	2007	Link	Opinion

…But I thought the term 'contextual kerning' was not intuitive for Japanese punctuation processing and 'punctuation-trim' was better for it…	2007	Link	Opinion
…I'm wondering if we shouldn't add keywords 'top' and 'bottom'. They would technically be aliases for 'left' and 'right', but maybe more *intuitive* for users…	2007	Link	Opinion
…Hm ... that's one of those strange ideas that's counter-*intuitive* through naming…	2007	Link	Irrelevant
…as there is no mean for me to distinguish between them, it will be counter-*intuitive* and frustrating to have different copy-paste behaviour…	2007	Link	Opinion
…It would be easy enough to add support for this to CSS3 'text-emphasis' property by adding something the following values (not intuitive names and prone to typographic errors) …	2007	Link	Opinion
…If this is the case, the code could be made more *intuitive* by adding keywords. For example: @media screen and (aspect-ratio: portrait) …	2007	Link	Opinion
…We have a proposed last-line-align: size whose naming isn't as *intuitive* as we'd like but which means that the UA adjusts the font size so that the text of the block fits exactly on one line. Maybe 'text-align: justify; text-justify: size' is more *intuitive*.	2007	Link	Opinion
…They put any empty element exactly at the same position where it would be if it wasn't empty (as long as adding content doesn't prevent its top margin from collapsing with any children). That has continuity, and it is more *intuitive* (at least for me). If the spec says otherwise there must be a strong reason - what is it? The strongest reason is that this was what we agreed to. If we change these rules it could have very subtle effects that might not be understood for years, at which point we'd be back to the same position as we are in now: thinking the rules are *unintuitive* and wanting to change them in another subtle way.	2007	Link	Opinion

…We could either allow only degree measurements (which would be odd because all other angle properties accept all angle units, just like all other length properties accept all length units) or accept all angle values and simply round to the nearest 0deg/90deg/180deg/270deg angle. The WG, to avoid changing parsing behavior and because it seemed more *intuitive*, opted for the latter.	2007	Link	Opinion
…I'm thinking that "end-count" or "max-count" might be a more intuitive keyword than "total-count"…	2007	Link	Opinion
…This vector should be normalized (have a length of 1), otherwise the results are typically non-*intuitive*…	2007	Link	Opinion
…I guess if it is the SVG one there is a precedent, but it isn't the standard coordinate geometry space, where y is positive upwards, and I would have though z positive into the paper was more *intuitive*…	2007	Link	Opinion
…A line saying that references to the dimensions of the image refer to the size after background-size has been applied would suffice I think. For -o-background-size we currently implement the non-intuitive version and are planning to change this…	2007	Link	Opinion
…"None" would still be useful as a handy and *intuitive* way to kill the transition, I would think, either in JavaScript or if you wanted to prevent the transition in certain elements that were part of a class that received transitions…	2007	Link	Opinion
…I would have thought that in order to be usable, a site has to be accessible; if it's pretty as well, that's a bonus (for sighted users) but in the greater scheme of things, I'd look for "accessible", "navigable", "consistent", "coherent", "*intuitive*" and any one of a half-dozen or so similar concepts before "pretty"…	2007	Link	Irrelevant
…> How about 'background-position: 0 0 calc(100%-15px) calc(100%-15px)'? Not very *intuitive*, but it does make a background image of 15 x 15 px using your model :-)…	2007	Link	Opinion
…Borders can be big, though often not, so drawing the shadow only for the box/background-color itself might hide	2007	Link	Opinion

the shadow when I would expect the border to "cast" it as well. But, this has problems. If the background color is rgba(), the shadow might show through, which is counter-*intuitive*…			
…I like that idea. A bit selfishly, perhaps, since English is my primary language. I understand the need to support writing directions other than Roman-style, but I find terms "start" and "end" (and their perpendicular counterparts) non-*intuitive*.	2008	Link	Opinion
…For an element that is height:10%, setting center-y:5% should have the top edge lined up with that of the parent element. This seems more *intuitive* for me…	2008	Link	Opinion
…I suppose a very careful reading might reveal to authors what those values do for position:center and how they differ from position:absolute. I don't think it is as *intuitive*, though.	2008	Link	Opinion
…I think background-size or background-sizing would be the more *intuitive* for most designers and developers for the reason that background-stretch or resize implies that you must be stretching or resizing when that simply isn't the true meaning of the property…	2008	Link	Opinion
…background-fill (this one is pretty darned *intuitive*)…	2008	Link	Opinion
…I will go with the others who have replied. I like background-size because more *intuitive*…	2008	Link	Opinion
I'd like to add a few notes on "alignment" proposals: First, I have mixed feelings on the goals of the new property. It appears to have multiple reasons to exist: … Provide a more intuitive *alternative* to "margin:auto"…	2008	Link	Opinion
…If I specify a top, bottom, right and left of 0, then why on earth should the object's intrinsic width or height override? It's completely counter-*intuitive* that you can't use this pattern to stretch an iframe or image in CSS2.1.	2008	Link	Opinion
…CSS already has a way to center blocks, but many authors find it (use of 'auto' margins) confusing and/or non-*intuitive*…	2008	Link	Opinion

…The intuitive behavior of 200% or 2 as a line-height value would have been to be twice the value of 'normal.'…	2008	Link	Opinion
…I still think background-fill is more *intuitive* to the designer mind, but given the example you showed re: fit, I think that would work as an option, too…	2008	Link	Opinion
…Isn't it a bit odd that of one block is wider than its parent, and the parent is set to "overflow:scroll", that the margins of the child are shown on the top, bottom, and left, but not on the right? … Perhaps there is something in the spec that says it should do this? Does it also say why? It seems counter-*intuitive* to me.	2008	Link	Opinion
…There may be reasons that it is the way it is, but as a designer, I find it counter-*intuitive* that one is missing, especially when the other three are present.	2008	Link	Opinion
…You are entitled to your opinion. I do not share it. I would rather it be *intuitive* to the author, even if it complicates the calculations that the programmer puts into the software.	2008	Link	Opinion
…An author would find it easier (and more *intuitive*) to change text-shadow "red" and background "url(a.png)" since they both would occur first…	2008	Link	Opinion
…My intuition says that overloading ':checked' is dangerous and not very *intuitive*…	2008	Link	Opinion
…Even if you didn't call it "checked" (I think "checked" is as good as any other, and no more overloaded than it is for radio buttons), if you made it operate the same then it would be *intuitive* to set and change…	2008	Link	Opinion
…I made a different proposal entirely, mostly based on the fact that calling a property 'background-origin' and then having its intended effect have little or nothing to do with defining an origin point seems counter-*intuitive* at best. Thus my suggestion to completely redefine that property and shift what its currently-drafted values do to a new property with a (slightly) more *intuitively* descriptive name.	2008	Link	Opinion
…it's *intuitive* and simple enough so web authors can edit w/o having to hire an expert to understand the spec…	2008	Link	Irrelevant

…Bert: it does seem we want a separate property Steve: we are going to need it for vertical Steve: and not having the same thing for horizontal would be confusing Bert: are there any other values? Anne: left and right Bert: that's handled by margin auto Tantek: *unintuitive* and hard to teach people … Jason: because margin auto is not an intuitive way to do it … Jason: again it gets back to the *intuitive*ness of margin auto - that's the problem.	2008	Link	Opinion
Fantasai: Concerned about difference in white space syntax of nth-child() and calc(). … Daniel: "7n + 3" looks very *intuitive*, is used in many other places that people now.	2008	Link	Opinion
…As I said, I don't care what we define, either way will be non-*intuitive* to some people…	2008	Link	Evidence
…I would suggest using Anne syntax for consistency. A forward slash "/" would indicate either a fall back or a fall forward. like. content: "hello" / url(hello.png); The fall back to the left I think is more *intuitive* for authors. Also using commas would mess with multiple background strings. This would be better.	2008	Link	Evidence
…I think the *intuitive* way to look at the problem is; that no matter how or where you break inside the child of element you also break that element…	2008	Link	Opinion
…you should find the best place to break, i.e. or the place that break as few rules as possible… I believe this produces the visually best most *intuitive* results… If you have elements with borders stacked inside each other, and you are forced to break the topmost of them, the most	2008	Link	Opinion

intuitive thing to do is to try to break between the next level of bordered elements.			
…<body style='position:relative' onload="alert(a.offsetParent == document.body)"> a </body> The *intuitive* outcome of the display would be an alert with the value 'true'. However, CSSOM would make it so that the outcome would 'false' in the alert…	2008	Link	Opinion
…I agree the 'background-origin' name isn't very *intuitive*. Not sure what would be better, though, given that we also have 'background-clip' which can be set to a different value.	2008	Link	Opinion
…With enough imagination, one can even imagine a positive value on the blur being reduced a pixel at a time, until it passes zero and jumps into being a shadow of the negative space (also called "white space" or page background). Minus sign = negative space shadow; has a certain (only slightly convoluted) logic to it, even if the "blur" value is not the most *intuitive* place to put it…	2008	Link	Opinion
…Quote: "In graphic representation, an artist uses *intuitive*, artistic, scientific, or technical skills to represent the phenomenon of the visual perception of perspective. In simpler terms, these skills are used to add a suggestion of depth to what is ultimately a flat image or drawing."…	2008	Link	Irrelevant
…The draft makes no mention of whether variable lookups occur using every variable defined on the document or only variables that occurred earlier in a depth first parse of the stylesheets. I believe it's much easier (and more *intuitive* for authors) if lookups occur using every variable available for the current media (since dynamic re-evaluation when methods like setVariable get called would be much more problematic otherwise).…	2008	Link	Opinion
…because once this is implemented, how could we ever add any new features. I suggested highlight initially because the word *high* would seem *intuitive* for authors in what the property is actually doing.	2008	Link	Opinion

…"content: inhibit" or 'content: "" should serve just as well as your "visibility: background". "visibility: foreground" appears equivalent simply to "background: none", possible in any browser that supports a CSS background in the first place. These solutions seem more *intuitive* to me…	2008	Link	Opinion
…I agree. I think the model of just allowing variables to be defined at the document-level is simple and *intuitive*. It allows for centralized variable definition and reuse…	2008	Link	Opinion
…As for whether to use the sigil in the declarations as well, I don't think there would be any difference in terms of parsing it, in the syntaxes that have been contemplated so far. I think it's more *intuitive* to use the same name in declaration and use, not require an extra character for one but not the other.	2008	Link	Opinion
… Column rules are only drawn between columns that have content in the normal flow With my designer hat on, the WebKit implementation seems more *intuitive*.	2008	Link	Opinion
…I think displaying the marker makes the most sense from an authoring perspective. Treating the marker as the principal block box's child and clipping it is neither useful nor *intuitive*…	2008	Link	Opinion
…I don't think it is *intuitive* for the outside marker to affect the height of a line and then not scroll with the line…	2008	Link	Opinion
… The spec says that the ellipsis should be rendered before the overflow boundary, but this is not the browser behavior (rendered after the last non-removed character), and also not the *intuitive* behavior (in written text, the ellipsis is always placed right after the text that is truncated). I think the correct behavior is to visually remove grapheme clusters until enough space is available for the ellipsis, or until a block boundary is met (from an inline block, or the boundary from the overflowing block), and to then render the ellipsis at the insertion point of the last removed	2008	Link	Opinion

grapheme cluster. That's *intuitive*, and mostly consistent with the behavior as implemented…			
…This would correspond to @define after being blessed by an official w3 recommendation I guess. I quite like "define" as its name is quite *intuitive* wrt its behaviour, and it avoids any overloaded interpretations of what to expect from something named as variable…	2008	Link	Opinion
…This problem certainly exists, but I'm not sure how much attention we should give to it. I would prefer a short and *intuitive* syntax like $ and would welcome that some kind of "damage estimation" be performed for the most popular alternatives.	2008	Link	Opinion
…The *intuitive* mind is a sacred gift…	2008	Link	Irrelevant
…Peter: My concern is what happens when we start getting rich fonts with multiple weights. Peter: I want to be sure that result is intuitive for fonts with more than two weights…	2008	Link	Opinion
…Steve: The alternative is that 'last-line-align' doens't apply. Fantasai: The 'last-line-align' applies because there is a forced line break. Steve: I wouldn't call that an inline element (because it contains a line break). David: Maybe the term [inline] is not fully *intuitive*, but it *is* precisely defined.	2008	Link	Opinion
Elika: border-corner-shape: [sides | corner] || [round | sharp] Alex: border-length is a little more *intuitive*…	2008	Link	Opinion
…david: another way to solve same problem is calc() alex would find it more *intuitive* to have a separate property that defines alignment direction…	2008	Link	Opinion
…No, I think we are again in a case when 8.3.1 excludes collapsing (used height NOT equal to what it would have been if min-height were its initial value).	2008	Link	Logic

Really? That seems counter-*intuitive*. Compare these 2 test cases: http://lachy.id.au/dev/css/tests/adhoc/collapsing-margins-02.html http://lachy.id.au/dev/css/tests/adhoc/collapsing-margins-03.html			
…Variables as Daniel and I specified them can remain unresolved until you end up using those rules in a specific medium. This "global soup" approach is simple and *intuitive* for authors, since the variable names always cross stylesheet boundaries (without ever having to delay the parsing of a sheet because another sheet hasn't loaded yet), and the last rule specified in the sheet order wins.	2008	Link	Opinion
I think the current method of interpreting percentages is very *intuitive*. It seems your concern is with calc()…	2008	Link	Opinion
…But consensus is not always a synonym to "*intuitive*" nor to "best solution"…	2008	Link	Irrelevant
…This allows graceful degradation for browsers that do not support text-shadow, in the cases where an author would use the same "color" and "background-color", and rely on a shadow of a different color to make the text readable. As it is a fairy common use case (for example, about every text-shadow demo on the net uses it), I think such an *intuitive* way to have graceful degradation would be much appreciated…	2008	Link	Opinion
If we choose nesting of ::selection pseudo-elements, then we run into the problem that the rules: p::selection { background: purple; } ::selection { background: blue; } would cause the selection on any element inside the p to be blue. However, we could still represent the default selection as :root::selection rather than as ::selection, although this seems less *intuitive* to me.	2008	Link	Opinion
fantasai: That would not make sense if the min-height is big enough to contain the margin Alex: but its behavior is continuous Discussion about what is *intuitive*	2008	Link	Opinion

Steve: It really bothers me that we don't have any designers here			
Alex: Min-height is as currently specified has a side-effect on margin collapsing that is not *intuitive* to the designer	2008	Link	Opinion

… Daniel: I have a <pre>, and I want a minimum height for my code box
<dbaron> Designers aren't really using min-height in the wild because of IE support, I think. everybody has a different idea of what designers would want for min-height and margin collapsing
fantasai posts to twitter and gives up trying to minute
Discuss dbaron's option E
Alex: That's what IE8 impelements, and I'm not convinced it's more *intuitive*
	2008	Link	Opinion

…Hakon: next, border parts generated content for paged media spec Hakon explains example XXXV bert makes a very funny face Hakon: this is very very cool Hakon: needed for footnotes Hakon: very *intuitive* Hakon: way to define dash above footnotes	2008	Link	Opinion
…Peter: You can have "7px + -4px" Haakon: So these spaces here are significant? Bert: Some of them are. Peter: Can you nest calc()? Bert: no Bert: Seems kind of pointless. Peter: It's *unintuitive* to a user to require spaces around - but not around / or *.	2008	Link	Opinion
…15:55 * Bert wonders why HTML5 doesn't add <toc> ...</toc> elements... Peter: OK, z-index first thing tomorrow, then. Haakon: I have another issue about the page counter. <glazou> Bert: hey, that would be a too simple and *intuitive* solution :-)	2008	Link	Opinion

dbaron: We also need counters work for the HTML5 header algorithm, counter-set that doesn't create a new scope might solve it.			
…HTML solves this by allowing the onfocus handler to be attached to the BODY tag, which seems like a sensible, *intuitive* place to put it. I could see the argument for putting it on the HTML element instead, but they didn't.	2008	Link	Opinion
…I prefer a way of correcting it that is similar to the way it was dealt with in HTML (pretending that the BODY is the WINDOW, for certain things), as that would be familiar and *intuitive* for authors…	2008	Link	Opinion
…Thirdly, I understand from [1] that clearance was originally implemented as a change in margin-top. Superficially this seems *intuitive*, so there must be some tricky edge-cases which expose problems with this implementation.	2008	Link	Opinion
…The issue is that you're essentially duplicating the Grid Positioning Module (http://www.w3.org/TR/css3-grid/). In many ways Template Layout is just a pretty face on Grid Positioning, making the whole thing easier and more *intuitive*.	2008	Link	Opinion
…So the only remaining question is whether xy := <nowrap>x</nowrap><nowrap>y</nowrap> should wrap the same as <nowrap>xy</nowrap> I think that is more *intuitive* than the alternative…	2008	Link	Opinion
…In Gecko we follow two principles: 1) Break opportunities induced by white space are entirely governed by the value of the 'white-space' property on the enclosing element. So, spaces that are white-space:nowrap never create break opportunities. But 2) When a break opportunity exists between two non-white-space characters, e.g. between two Kanji characters, we consult the value of 'white-space' for the nearest common ancestor element of the two characters to decide if the break is allowed.	2008	Link	Opinion

I think these principles are reasonably *intuitive* and useful.			
To be continued from here…			

Appendix B: Experiment Notes

M9Q0L5

Notes

- In the GSS example, found all the equals sign confusing, did not know what they could mean.
- In the CSS Grid example, everything was reasonably intuitive, just unsure about the units being used.
- In the Flex example, the shortness of the code appealed to the participant.
- Commented on a preference for background-color to appear at the top of a style declaration.

Transcript unavailable: recording program crashed before reaching substantive part.

X0Z3C0

Notes

- Grid style sheets: usually much easier to see the units, but don't see a unit in grid style sheets
- Looking for differences between the code
- Grid-template-columns looks like Bootstrap, it looks like CSS Grid uses a framework
- All the properties are very common in the non GSS and CSS Grid examples
- I like tables, but they are not so flexible
- Flex looks basic
- GSS: Too complex using equals signs, too much new stuff. Not so simple to move from CSS to GSS with the new syntax. However, moving from CSS2 to CSS3, just need to know new features.
- CSS Grid is not so hard to understand. Think the units are different proportions.
- Noted browser support concerns from Flexbox and that different styles could be used for different browsers.

Transcript

E: Choose the one that's the most intuitive, the one that makes the most sense.

P: definitely not the second one probably (Grid Style Sheets). I see it is much clearer when you see what you (unit) measures are in.

E: What are you looking at now?

P: I am looking line by line and seeing what are the differences. Now I am looking at the first one, grid-template-columns, looks like Bootstrap. The first example looks like some kind of framework, so you have columns and lines. It depends on which code I should choose… Let's go to flex. I would say I would choose the last one, because I can clearly see the height and width and in pixels… but it is only because I can see what it's related too. I like Tables, but they are not so flexible.

E: You would choose flex?

P: Yeah, it is more interesting, you can do more stuff…

E: Does the code make sense?

P: Yep, I don't see anything special. What about the third one which is Grid Style Sheets element centring. I don't know, is it really normal CSS?

E: It's proposed an extension to CSS that uses constraints.

P: What's the reason, for example, what is the reason to use double equals? It is more like C++ or similar. You don't see the difference… It is too much new stuff. It is not simple to move from CSS to this one. If you choose CSS3, it's not such a difference from CSS2 because it's just adding new functions… it is not hard to move from one to another, you just need to know new (functions), but here you need to know much more: how to specify parameters in a new way. What about the first one, CSS Grid element centring, I think it's… it looks like… kind of… yeah, it's not so hard to understand. It's okay. I am not sure what grid-template-columns. I think it is different proportions… I would choose Flexbox…

Internet connection problem

E: (Which was the second code snippet?)

P: The second one is Table Styles. The last one is Flexbox.

P: I should probably say why it could be better to choose tables. Sometimes you can't choose Flexbox if you need to support old browsers. Probably better to use a mix… different styles for different browsers.

25.07.2016 5:30pm

K1C5C8

Transcript

P: So, in the first code snippet, it looks like it's made with CSS Grid. I have never done something, so I see grid-template-columns. I can't… I don't know what this does. So it's… let me look… I just can think… I really don't know… I see the grid-column: 2… I try to make sense how a column can fit in there because it is centred horizontally and vertically. And then inner element p is just padding… and is text align-centred.

P: The second code snippet, Flexbox. With Flexbox, I am familiar and display flex works like magic. The inner element is with width 60% and the p is centred… Yeah… okay… If I see this code snippet I would know it centers something in the middle and there are margin auto. But with flex it's just 'display this as flex and it's there and it's centred' but I really don't know what… why it does what I want. If that's…

P: The third code snippet is done with display: table and display: table-cells which for me is straight forward but I know tables aren't there to layout something. Tables are there to … make tabular text alignment. But you know what's going on. With display table-cell you can do vertical align middle and vertical align middle speaks for itself… it's vertically aligned in the middle. Inner element p is centered with margin auto and has a width… it's clear what it does but I know it's not the most perfect thing to do because it's a table.

P: And the fourth code snippet, I've seen this before, this is GSS. I've looked into it. But it's kinda 'whoa', it's a completely different attempt to CSS. And… let me look… it has the top, window[top], left… it's the outer element. Okay… outer element is on the window top and window left. The inner element has the background colour, green. Ah, okay, the p is text align center and then we have inner element is greater than or equals than outer element… okay… so the inner element depends on the outer element width, but 70%. It's centered with the outer element center and the inner element height is like the p height, so it's… I have think what it does, but when you see the code, it speaks for itself.

E: So which would you most like to see in a code base, where you see okay, this is doing what that layout is, what the layout shows?

P: I think for layout, GSS is very good because, like a said, it speaks for itself.

Answers written questions…

I can see… with the CSS Grid, I read it and didn't know what it did and I went to it with my cursor. With Flexbox, I knew it does something, it centres it, but hmmm… and if there was the table styles. I kind of went over it and analyse it and knew what it does. And with GSS, I went over it and this intrinsic height, I don't know what this does, that's because I went with the cursor. And then I tried to figure out these layout positions.

M706W1

Transcript

P: To start with, I don't know what a CSS grid is to be completely honest.

E: It' behind flags, like it's a very new, experimental feature. As you read through it, try and work through what it could mean.

P: So my guess is that grid is some kind of… um… so template-columns… There's another framework we've used before… bootcamp? Or something.

E: Bootstrap?

P: Yeah… Bootstrap had the column-layouts system thing going. And you could define things within a column of a grid. And this looks similar to that. And this would probably mean 3 some unit (not sure what fr means), 14 and a 3. Which means you have 3, left column over here, then 14 then 3 here (point cursor to areas on example layout). I am not 100% how the vertical is calculated. Maybe the height… ah the outer element… So the outer element is divided into 3. This a bit confusing to me because it defines… it's interesting… Align items center. Is that a new too?

E: Yeah, sort of, it can also be used with Flexbox

P: Yeah, which I also don't know. So, align item centre, I have no idea what that would mean. This is like implicitly a certain layout, I'm not sure what that would mean if it means this way… or vertically centred. It's not explicit. It would take me ages to find that line of code to do what I want it to do. Height and width are pretty standard across all… I assume we want a fixed height and width and that's not important. Inner element… this is interesting… you pick grid in reference to the outer element (looking at grid-column 2). I assume this means you go in this slot (middle of layout) here. That's like super interesting. I don't particularly like it because this means that (the column number) is dependent on (the grid-template-columns value) which could be somewhere else in the CSS. Where, in fact, this should be nested in here. It is hard to explain what I'm trying to say I guess. It's weird that we're moving the HTML logic into the CSS kinda. It's scaring me a little bit. You're basically doing programmatic… not programmatic… but like ordered layouts in CSS which is not common.

E: An the order is part of the semantics.

P: Yeah… so it's like 2, grid-column 2. Which I guess makes sense to read, but I don't like it in that regard.

P: That's the paragraph…

P: Let's go to Flexbox. So I don't know what flex is, right of the bat, so I'll try to guess what it means. I think I might have experienced it, I just don't remember. This is all standard stuff, it's no different. So the inner element is green, it will be green. So 70% of its parent width. So auto means just use whatever remains of the width on both sides. I think that's typically what it does even without flex.

E: Do you know what it does with flex?

P: No

E: Also vertical…

P: Oh, vertical alignment. Okay, I just don't know that. So that means top, bottom, right, left. That's super cool. Um… width… yeah, so since I haven't set the… have I set the height? Is the height implicitly 100%? No it's not. Where does it get the height from?

E: Here, height comes from the bottom up, from the paragraph.

P: Sure, fair enough. Text-align… centre…, yeah… same stuff… And here we're doing the same thing assumedly? To centre this content? Without being how that works… only because I know how CSS margins work kinda… but I would not have guessed that flex did the vertical alignment. I guess if this was a live demo I would probably work it out pretty quickly though. This one's alright, less code and that kind of stuff.

P: Ahh, tables… I never code the table display stuff. Vertical align middle. That means vertically aligned, is that right?

E: Yep, vertical

P: Yeah, but it's in both middles. Display table cell, um… (sigh)… This one annoys me… It uses the properties of the table, but it's not a table and nothing about it kind of says where to put… uh… I guess that's what… I am not sure what's going in this one… What's giving the side paddings this time? Ah, width 70, so it wraps the paragraphs content. Margin auto, same business… Text align centres… Not much to say about this one.

P: Grid Style Sheets element centring, this looks scary. I bet you this is your one isn't it.

E: It's not mine. It is a framework, JavaScript, and it has been proposed to used constraint syntax in CSS:

P: Right. Got it. Alright, so, what's going on here? I'm not going to pretend I know what's going on here. In traditional programming languages the double equals means evaluate the

equality between two parameters. I'm guess it means the value… I have no idea honestly. I'm getting it means it can be calculated. Top, this is what leads me to believe that, is that it's on this parameter here. It's, argh, something like: top should to be the top of the window, and left should be the left of the window. Intrinsic-height, that's a fancy keyword *looks up definition of intrinsic*, so that's means the height should be the natural height of the… I'm guessing it's going to use the padding and the height of the text rather than all the space available. Down here we have inner element is greater than or equal to the outer width times 70% (laughs) what? Okay, centre, outer element equals centre. That makes sense. I guess that's defining a range (pointing to first of three last constraint statements) for (inner element), saying that it can be greater than or equal to 70% of (outer element).

E: How would you phrase that statement in an English sentence?

P: OK, let me just… The inner element should be at least… the max width is 70% of the outer element. And for this one, they should share a center point. This one means infer the height from height of the p which has an intrinsic height. This is kinda that as well. That's interesting. Does this stuff get calculated by the browser or is just JavaScript… then CSS?

E: It's JavaScript, it's a framework. The suggestion is that the browser could do it.

P: I feel like this is the sort of thing that someone would type when they are trying to show off their programming skills. Having just learnt about grids and flex, I would probably use Flex. Partially because I am lazy and it seems to be the one that works the easiest… with the littlest… you know… things that confuse me. Like this I don't like because you have to consider these (the little fr things). Is that what fr means, fraction?

E: Even in the spec, it's not exactly clear what fr stands for. But it is a fractional unit… it's a fraction of the whole grid.

P: So why is it 20, not 10? Ah, right! So whatever number you imply is the total. So it's 3 twentieths. So could just do one, one, one and it would be equal.

E: Exactly.

P: Um… I have to pick one don't. Fine this one… (picks CSS Grid).

E: Now, there's a bit of a reflection…

P: I would not like to write anything…

E: Just talk a little bit more, but answer the last two questions.

P: Explain why you chose… ahhh, yeah I will talk. I chose this one (CSS Grid) because after looking at all of these, it's the one I understand the most. Despite the fact it is not ideal for this kind of think. This is more than just centring content, this is a specific layout; like I want a 14th of a 20th or whatever… (the grid column template value) is not as arbitrary as 70% essentially. I am not sure… but… I previously said I don't like this one, but I picked it anyway. Maybe I don't like centring with CSS. If I had to go with one, I would use (CSS Grid) simply because I understand it the most. And I have not used Grid before, but this makes a lot of sense. You got your fractions of a sum total, I think that's pretty cool. This I am not a big fan of, but it makes sense.

P: Okay, that kind of makes sense. You want to put it here (column 2), then you get a grid like that (grid column template). This I am not a big fan of, align item centre which I assume lines it this way (vertically) because that should be a part of the grid right, rows or something, who knows. I've actually been in a situation where I have wanted to centre content and had no solution. The actual solution was to press both halves of the image against each other. Then you make two boxes, one is pressed against the bottom of the other element, the other against the top of another element and they just happen to centre the image for you. It's really horrible.

P: Characteristics you found counter-intuitive… Despite picking this one, the align items centre comes out of nowhere and doesn't make sense. And if I was looking at this and this, I would not assume that this meant this. There should be grid template rows.

P: Tables stuck. This one has too much JavaScript… it's not JavaScript… but too much logic for something so simple. Even if I understand it, I would not want to read this. This one, I think I just don't fully understand. It seems like flex is just magic. It might as well say display magic. Whilst it would be nice if it worked… to be honest I would also pick this one, but only because it just works.

E: Interesting that you noted that, that it works just like magic. But with grid you can at least see the columns.

P: Yeah, there's a defined layout. I don't entirely like the fact that display affects its laid out. But (Flexbox) is magic. Display flex, then margin auto, wow amaze. The only value we're setting here is width. So that somehow means that by default everything should be pretty.

Z4T2D3

Notes

- dont know what columns means

- css grid element
    - template columns don't make sense
    - Similar to CSS.
    - rank 3
- flex
    - don't know what flex means
    - like flexbox: makes more sense in the inner and outer elements
    - everything simple and matches up
    - lot easier to remember fewer rules
    - rank 1
- table styles
    - width of 70%
    - rank 2
- grid style sheets
    - Immediately, there's a lot more stuff going on, which I don't want to read.
    - Not as straight forward as the first three.
    - Least intuitive.
    - Straightforward
    - 70% makes more sense the GSS
    - rank 4
    - More used to just using the column. Reminds of objective C. ::window

Transcript unavailable, program crashed.

28.07.2016 6:00pm M9T5F6

Transcript

P: *reads the task…*

P: I see a figure element which has a larger outer element in blue and has a green inner element with content in it. It is position centred to the out element. The HTML which is obviously for this cascaded and a paragraph tag… all good… Now I see a table with three code snippets.

E: There might be a fourth if you scroll down.

P: Ah, there's a fourth. There's four code snippets, the first is Flexbox, the next is Table Styles, the third is CSS Grid and the fourth is Grid Style Sheets element centring. Each of these boxes contains some CSS code, each for the outer, inner and paragraph element. By the

way, the text is also centred in the inner element. The colour is everywhere the same. And most importantly, it is hard to position centred on the out element. I already knew about positioning of elements with Flexbox which makes the positioning of elements quite flexible as the name suggests. But I see that the inner element misses some flex attributes. Therefore, I would say that the first snippet is not the right one.

E: (*Corrects understanding of experiment stating that each snippet produces the same result)*

P: Okay, so, actually for the first what captured my curiosity is that the inner element has a width setting of 70%, maybe that is reasonable for how this is scaled. I would expect some positioning of the inner element which is not stated there.

P: So for the Table style element, the outer element with a fixed height and width, in all of those, and you have display table. Actually, then in the inner element we see that there's a vertical align CSS property which says middle which seems reasonable for me. And the display of type table cell and the paragraph is also with the text align centred. That looks pretty descriptive the second one.

P: The CSS Grid centring, let me check out this. The outer element is of display grid. It uses some grid template columns with abbreviations: 3fr, 14fr and 3fr. I can only assume that these are three columns three columns; I can't quite get what fr means. But that should be some virtual layout-ing scale numbers. Um… it positions the inner element in grid column two which might make some sense. But it does not say how wide the inner element should be. Ah. Now, I think I get it. It says in the grid template columns that there are three layout elements to the left of the space and 14 layout columns wide centring and then there are 3 to the right and I think the inner element is in grid column 2, I position it into (indecipherable? middle of) the layout columns and then having it with 3 spaces to the left and 3 spaces to the right which obviously seems to centre the inner element. But still it doesn't say something about the height of element.

P: Then, for the fourth, we have some meta classes which positions the outer element on the top left of the window. The inner element has no positioning constraints in the first section. But (down the bottom) it says that there are some calculation rules which looks a bit complex for me to figure out what it does. But it seems to calculate the positioning of the width and height of the inner element based on the width and height of the outer element. And positioning the inner element in the center of the outer element with an assignment. At least this assignment looks pretty need. But the rest is a bit strange from the syntax. I would say the second one is the one that makes the most sense to me.

C6Y7Y6

CSS Grid element centring

- three ids
- outer element with colour web
- it's a new thing... grid template columns... I am not too sure what fr means. It makes sense this.
- inner element
- grid columns - hard to

gss

- This is crazy, I never saw this way to attributes. I think it ... some kind of variables.
- We have same pattern, with width
- sure where variables like top and left come from.
- text element has a height; not sure what it means

table styles

- on the first view, a lot simpler than GSS.
- it's a table layout.
- pretty straight forwad. Table metaphor makes sense.
- inner element has a background.
- colour

Flexbox

- one step easier
- outer element
- margin auto
- if I knew

D0K3H9

flexbox

- paragraph, a
- I do this pretty regularly
- outer element,
- margin auto, align items is clearer.
- better semantics
- padding rem is a bit funny. could have made a flex.
- margin won't do anything

css grid

- new layout
- fraction unit. Setting up a grid where columns take up fractions.
- goes into the second column. you can have areas. big fan. from usability, flex is great for 1 dimensional layout

table

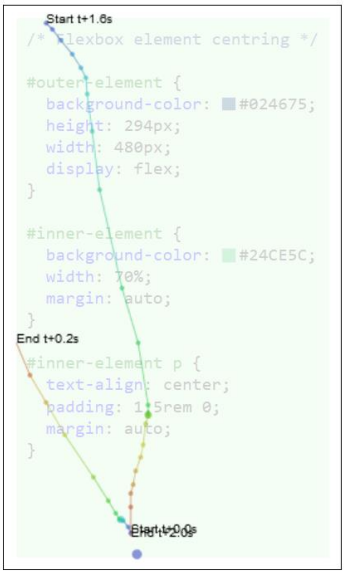
- personally, I don't use it a lot.

Gss

- okay, I have not seen it before.
- height... top... window...
- first impression: not sure about the double equals. I can see what top and left. Problem, is it always on the top. ::window is confusing, but its not position fixed. I am not sure what the expected behaviour is. always,
- intrinsic-height,
- assignmnet, but usually boolean operators.
- interesting idea, unfamiliar syntax. == window top
- relationships between

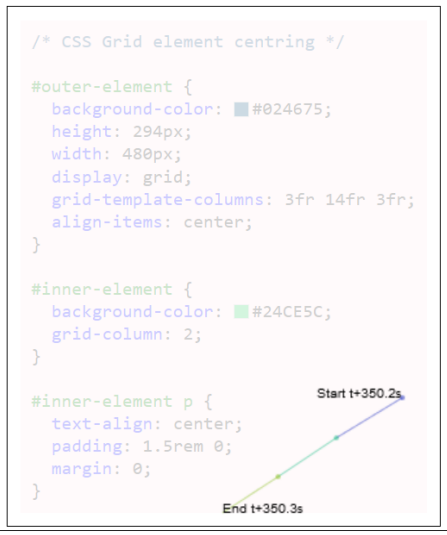
Appendix C: Cursor Movements

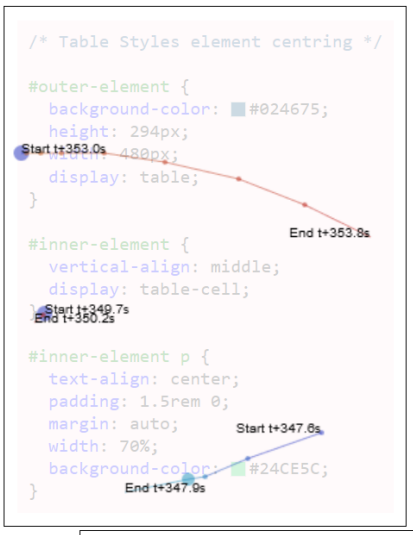
M9Q0L5d

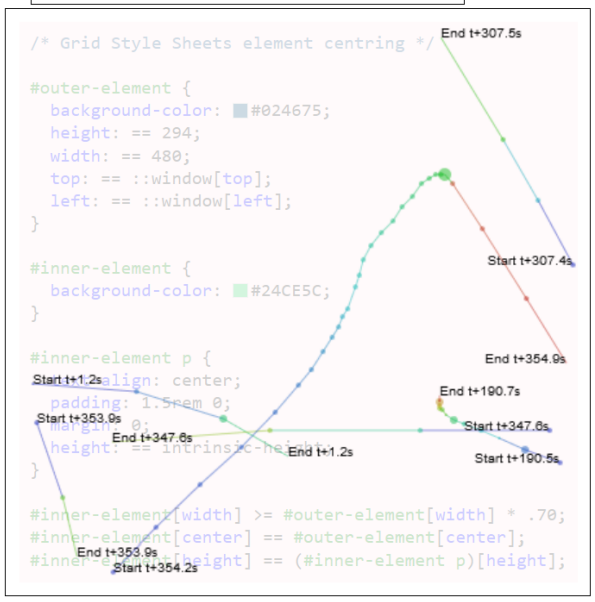


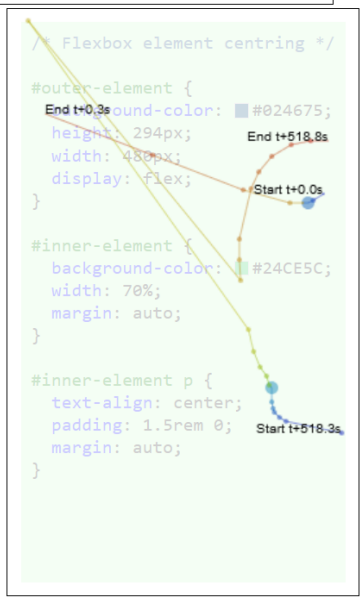
Note: Flexbox showed movement in this session.

X0Z3C0









Note: no activity on image or HTML in this session.

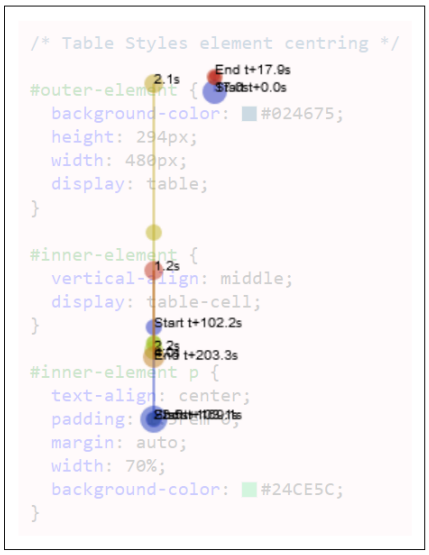
K1C5C8

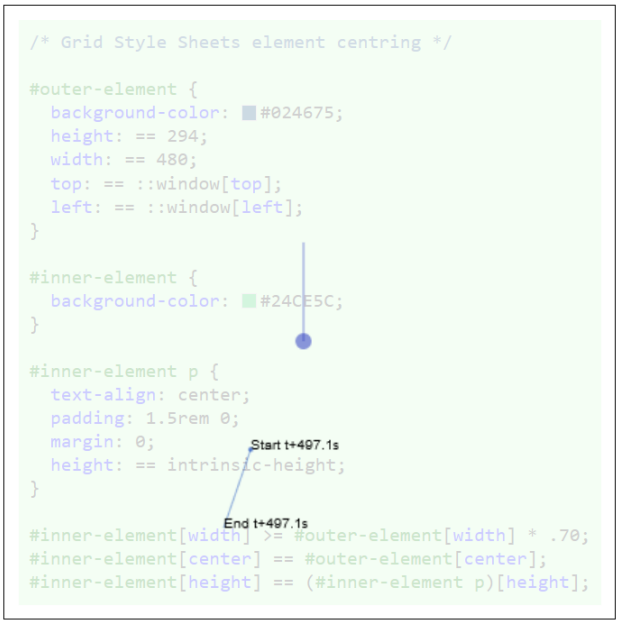
```
/* Flexbox element centring */

#outer-element {
  background-color: #024675;
  height: 294px;
  width: 480px;
  display: flex;
}

#inner-element {
  background-color: #24CE5C;
  width: 70%;
  margin: auto;
}

#inner-element p {
  text-align: center;
  padding: 1.5rem 0;
  margin: auto;
}
```





Note: no activity on Flexbox snippet or CSS Grid snippet.

M706W1

Figure 1: A basic layout where an element is aligned centrally within another element on both the horizontal axis as well as the vertical axis.

The HTML of Figure 1 looks like the following:

In Figure 1, `#inner-element` and `#outer-element` only label the elements and are not intended as part of the design. The "Content." text shows a paragraph tag centred within the inner element. The height and width of the green and blue elements is the same in each of the layouts produce by the code snippets below.

Choose the code snippet that makes the most sense

Each code snippet below produces a layout like that shown in Figure 1. Which of the following code snippets do you feel most intuitively (with the least amount of thought) represents the result of centring one div within another, as shown in Figure 1?

Z4T2D3

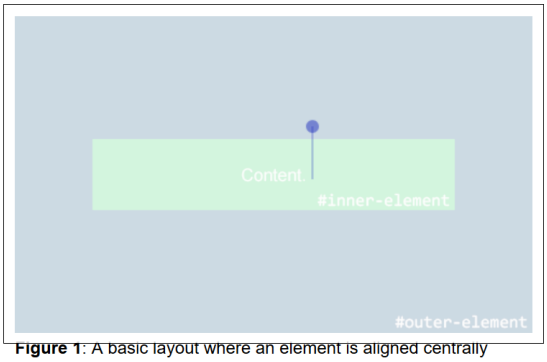


Figure 1: A basic layout where an element is aligned centrally within another element on both the horizontal axis as well as the vertical axis.

The HTML of Figure 1 looks like the following:

```
<div id="outer-element">
  <div id="inner-element">
    <p>Content.</p>
  </div>
</div>
```

In Figure 1, `#inner-element` and `#outer-element` only label the elements and are not intended as part of the design. The "Content." text shows a paragraph tag centred within the inner element. The height and width of the green and blue elements is the same in each of the layouts produce by the code snippets below.

Choose the code snippet that makes the most sense

Each code snippet below produces a layout like that shown in Figure 1. Which of the following code snippets do you feel most intuitively (with the least amount of thought) represents the result of centring one div within another, as shown in Figure 1?

```
/* Table Styles element centring */

#outer-element {
  background-color: #024675;
  height: 294px;
  width: 480px;
  display: table;
}

#inner-element {
  vertical-align: middle;
  display: table-cell;
}

#inner-element p {
  text-align: center;
  padding: 1.5rem 0;
  margin: auto;
  width: 70%;
  background-color: #24CE5C;
}
```

```
/* CSS Grid element centring */

#outer-element {
  background-color: #024675;
  height: 294px;
  width: 480px;
  display: grid;
  grid-template-columns: 3fr 14fr 3fr;
  align-items: center;
}

#inner-element {
  background-color: #24CE5C;
  grid-column: 2;
}

#inner-element p {
  text-align: center;
  padding: 1.5rem 0;
  margin: 0;
}
```

```
/* Flexbox element centring */

#outer-element {
  background-color: #024675;
  height: 294px;
  width: 480px;
  display: flex;
}

#inner-element {
  background-color: #24CE5C;
  width: 70%;
  margin: auto;
}

#inner-element p {
  text-align: center;
  padding: 1.5rem 0;
  margin: auto;
}
```

```
/* Grid Style Sheets element centring */

#outer-element {
  background-color: #024675;
  height: 294;
  width: 480;
  top: ::window[top];
  left: ::window[left];
}

#inner-element {
  background-color: #24CE5C;
}

#inner-element p {
  text-align: center;
  padding: 1.5rem 0;
  margin: 0;
  height: intrinsic-height;
}
```

Note: no activity on Flexbox code snippet or HTML.

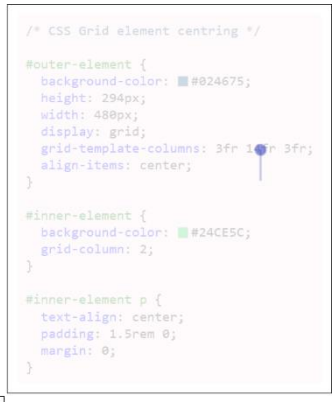
M9T5F6

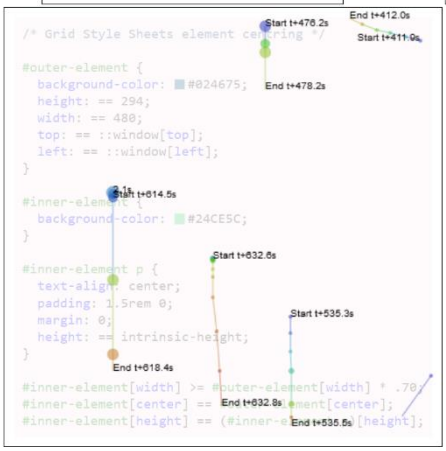
```
/* Flexbox element centring */

#outer-element {
  background-color: #024675;
  height: 294px;
  width: 480px;
  display: flex;
}

#inner-element {
  background-color: #24CE5C;
  width: 70%;
  margin: auto;
}

#inner-element p {
  text-align: center;
  padding: 1.5rem 0;
  margin: auto;
}
```





Note: no activity on Flexbox snippet, image or HTML.

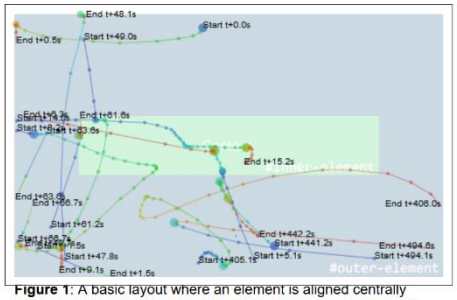
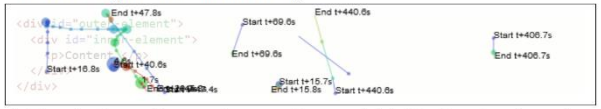


Figure 1: A basic layout where an element is aligned centrally within another element on both the horizontal axis as well as the vertical axis.

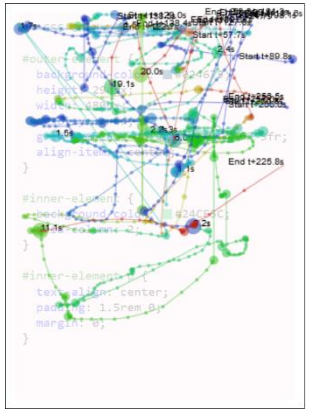
The HTML of Figure 1 looks like the following:

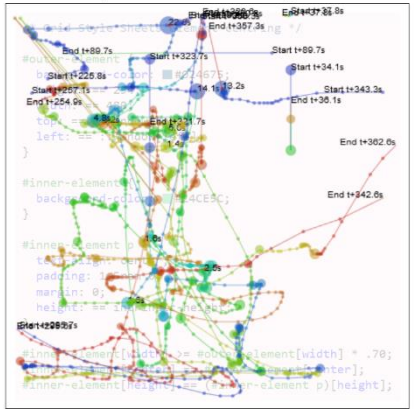


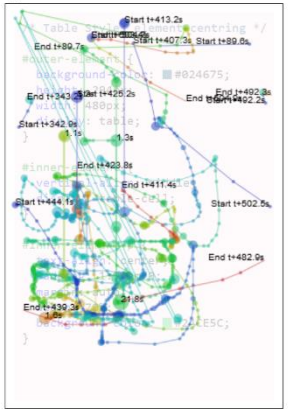
In Figure 1, `#inner-element` and `#outer-element` only label the elements and are not intended as part of the design. The "Content." text shows a paragraph tag centred within the inner element. The height and width of the green and blue elements is the same in each of the layouts produce by the code snippets below.

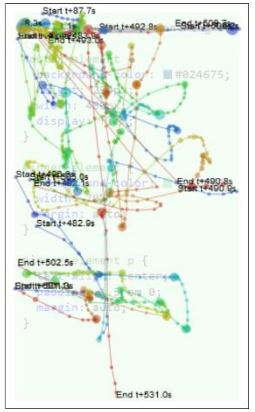
Choose the code snippet that makes the most sense

Each code snippet below produces a layout like that shown in Figure 1. Which of the following code snippets do you feel most intuitively (with the least amount of thought) represents the result of centring one div within another, as shown in Figure 1?









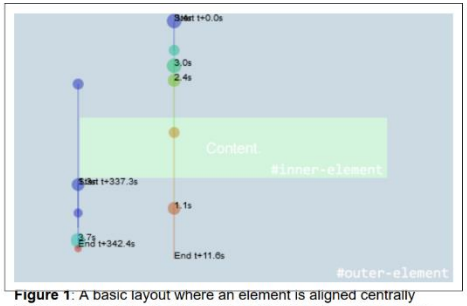


Figure 1: A basic layout where an element is aligned centrally within another element on both the horizontal axis as well as the vertical axis.

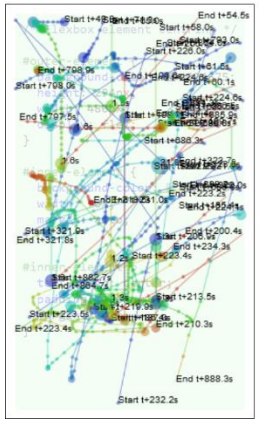
The HTML of Figure 1 looks like the following:

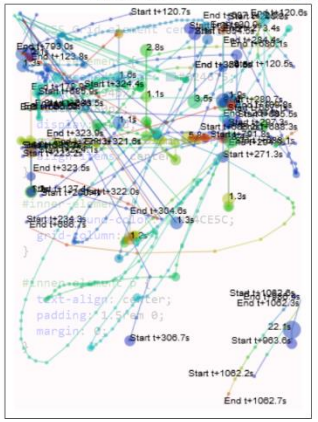
```
<div id="outer-element">
  <div id="inner-element">
    <p>Content.</p>
  </div>
</div>
```

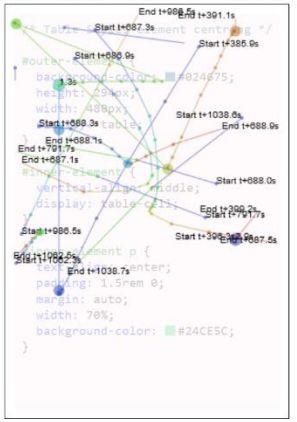
In Figure 1, `#inner-element` and `#outer-element` only label the elements and are not intended as part of the design. The "Content." text shows a paragraph tag centred within the inner element. The height and width of the green and blue elements is the same in each of the layouts produce by the code snippets below.

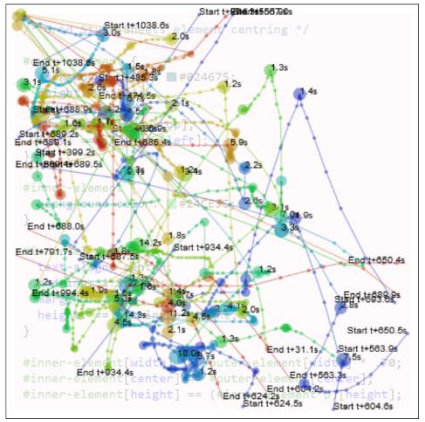
Choose the code snippet that makes the most sense

Each code snippet below produces a layout like that shown in Figure 1. Which of the following code snippets do you feel most intuitively (with the least amount of thought) represents the result of centring one div within another, as shown in Figure 1?









Appendix D: Declaration

I declare that I have independently written this Master's Thesis and have not used other sources than the sources and means indicated and stated in quotations.

William Clear 29 July 2016