CSS mindset

Demystification course with Lateral Nord

Lateral Nord.

Jerry Jäppinen

Product design consultant

jerryjappinen@lateralnord.com

+358 40 7188776

@jerryjappinen

The short version

CSS has absolutely nothing to do with programming

CSS is not intended for creating applications

...or even layouts

1. CSS for programmers
2. **Learning to walk**
3. History lesson
4. CSS is declarative
5. HTML structure
6. Worth knowing
7. Laying things out
8. Measurements
9. Best practices and exercises

CSS can be frustrating

- There are missing features in CSS
- We have to battle with browser incompatibilities
- Some parts of the spec are just crazy

But that's not why CSS is hard

It's hard because you never learned the basics

To run, we need to learn how to walk

To walk, we must understand principles

Just like you need to understand the principles of object-oriented programming before writing good Java, you need to understand something about fundamentals before writing good CSS

If Java is about OO programming...

Clojure is about functional programming...

Then what is CSS about?

design

CSS is about design

O_O

CSS is a way to talk about design abstractions

Alignments, dimensions, text, fonts, paragraphs, lists, colors and containers...

When you use CSS, you break down a design to these fundamental properties

Views can always be described as a combination of these four key components

structure, styles, behavior, content

Simply put: you write structure in HTML, behavior in JS, and styles in CSS. Content is merged into structure when delivered to browser, but usually stored separately

All of this is generic. Not just web.

Try out InDesign, you'll find tools similar to web technologies there. ID is for creating print materials, but it's a tool for implementing design just the same.

This is why some things in CSS might seem weird to programmers. Rules of graphic design are very different from programmer math!

1. CSS for programmers
2. Learning to walk
3. **History lesson**
4. CSS is declarative
5. HTML structure
6. Worth knowing
7. Laying things out
8. Measurements
9. Best practices and exercises

CSS is about styling HTML documents.

HTML documents had, pretty much, paragraphs, lists, tables. Images came in at some point.

Originally, semantics in HTML were awful.

When CSS came ubiquitous, poor semantics were the biggest hindrance to using it efficiently.

(Probably still is. Make sure this isn't the case with you.)

Prepare to feel stupid.

To train (which you need to do), write and style plain HTML documents.

Create a single HTML document with base styles, and then write 10 different themes for it.

You'll find lots of practical ways to organize your CSS and see how graphic design breaks down into border widths, background colors and font styles.

CSS is applied magically by the browser

Even for elements created dynamically

You don't have to actually **do** anything.

The people who actually write rendering engines are heroes who do what they do so we don't have to.

CSS doesn't tell a browser what to do.

You don't command with CSS, you declare what kind of arrangements, dimensions, text and colors you recommend as an author.

But you're not the only one who gets a say

User, browser, vendor and element styles cascade.

To author good CSS, pay less attention to applied styles of individual elements

You can also use this idea of cascading "layers" of CSS to organize your styling.

For example, simply write your multiple stylesheets:

- reset/normalization stylesheet(s)
- default stylesheet(s)
- global brand stylesheet(s)
- view-specific stylesheet(s)

Browsers handle more calculations and conversions to render stuff on screen than you ever want to know.

Read the CSS spec. It's crazy and it needs to be.

Browsers are mean rendering machines. Declaration is awesome.

But you just told us: structure and styles are separate!

They are.

But good structure and semantics are a prerequisite to authoring good CSS. Just like painting a bumpy surface is hard.

(That's all I had to say about structure)

HTML elements don't really exist in CSS, only arbitrary elements do.

Browsers decide how to render HMTL elements (like native-looking input fields), but CSS doesn't really know anything about that.

Things that don't exist in CSS:

Functions, class inheritance, grids or columns, mixins, variables, negation, ifs and elses, cancelling a declaration, centering.

The real power of CSS is **flexibility**.

Like we discussed, you shouldn't really worry about reproducing a pixel-perfect replication of a mock-up on a user's screen. Your vision is just one of the things that affect the result.

```
element#id.class child:pseudoclass .childclass,
anotherElement.class {
    attribute: value;
    another-attribute: another-value;
}
```

You'll get pretty far with nothing but this. This is one block, one declaration.

So that's it.

Start of gross simplifications

Like I said, we learn to walk first

```
em { display: inline; }
p { display: block; }
```

Basically, we have text and containers.

That's **inline** and **block.**

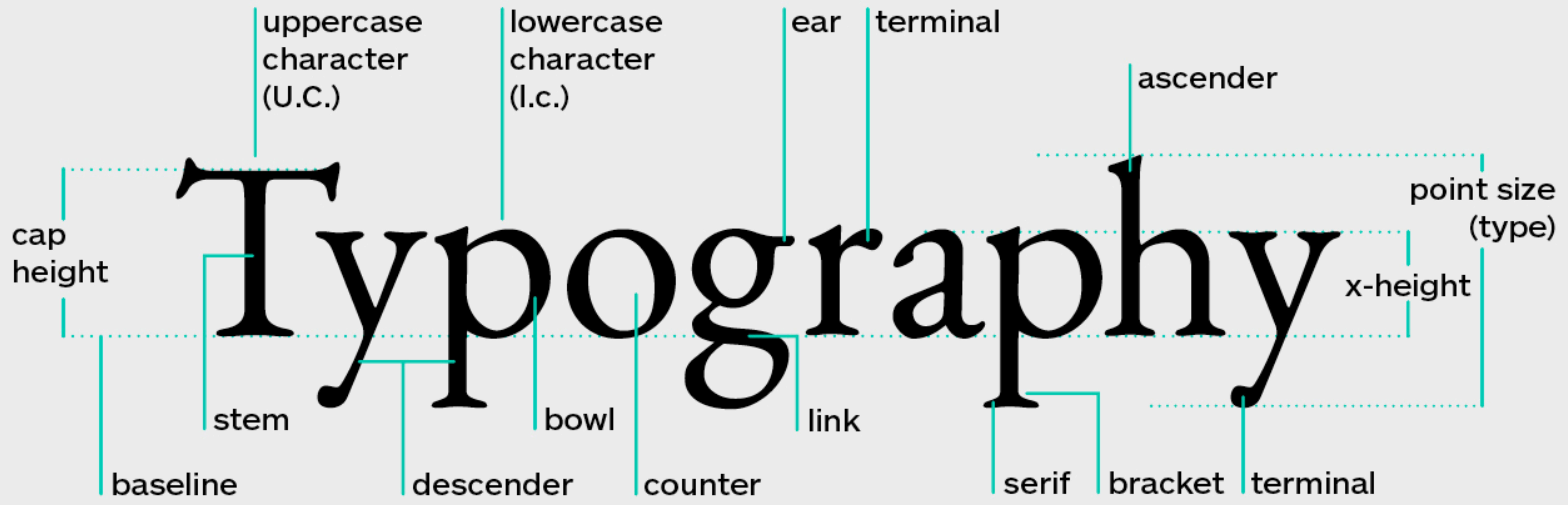
Inline elements are arranged in a natural text flow, and are laid out according to basic typographic rules.

Things like line-height, baseline and indenting are familiar from print designs. After this session, go get a Wired Magazine from the kitchen and look at the page.

Take an example: **vertical-align**

It's not what you think it is, but it's pretty rad.
It's about aligning inline items on a line of text.

And also content in table cells, because the same attribute has different uses depending on context. Sorry about that.



CSS actually understands what's going on here!

Cool typo in CSS

- letter-spacing,
- word-spacing
- direction
- text-align
- text-justify
- text-indent
- font-variant
- text-tranform
- word-break
- word-wrap
- white-space
- baseline-shift
- drop-initial-after-align
- ruby annotation styling

Block elements contain inline elements, and reserve room for them. Blocks have dimensions and can be arranged in a few ways.

Box model governs the rules for the dimensions of block elements. Blocks are extensively nested.

CSS is great at getting any content of a document to render sensibly

Layouts impose limitations rendering content, e.g. limiting dimensions of containers

Increasing these limitations lowers the threshold of breaking content rendering

In CSS content naturally flows from top to bottom

You control the horizontal dimensions of elements, and let browser handle the vertical dimensions based on what fits the screen on any given context

End of gross simplifications

Units of measure are awesome

Browsers handle all the hassle of converting everything to pixels for you

%: relative to available space or font size

em: the height of the font in context

ex: x-height of the font

px: dependent on canvas resolution

mm: millimeters

cm: centimeters; 1cm=10mm in: inches; 1in=2.54cm

pt: points; 1pt=1/72in

pc: picas; 1pc=12pt

vh: viewport height (percentage)
vw: viewport width (percentage)

vmin: vh or vw, whichever is lower
vmax: vh or vw, whichever is higher

Box model allows you to mix all the available units

Some things are a little tricky, some things are not doable, but many many cool things can be done

When you want a box of given size on the screen, you must always think why it should be of that size, and use that rationale in your declarations

1. CSS for programmers
2. Learning to walk
3. History lesson
4. CSS is declarative
5. HTML structure
6. Worth knowing
7. Laying things out
8. Measurements
9. **Best practices and exercises**

When starting out, break down your design into global components

Don't start replicating mock-ups or individual views

Each layout is a **system**

A **set of rules** that govern how UI objects look and flow

Express these rules in CSS

Avoid moving code from developer console to CSS

In dev console, you only see a change from one perspective. You're focused in applied styles

Never add more than the minimum declarations. Touch only the attributes you really need and want to specify.

Shorthands are short but express more:

background: green; means **background-image: none;**

Shorthands are magical and make it harder to iterate!

Don't think in hierarchies. **Classes are labels**, and allow for all kinds of permutations

Individual views are not a good way to organize stylesheets by

Try having separate stylesheets for separate concerns like body frame, columns, color scheme, text styling, button styles etc.

Organize declarations in logical chunks

Think about how you will be looking for a style declaration when you spot something on the page and want to change it

Understand **similarity vs. sameness**

Only include things in one declaration when you want the content to apply for everything described in selector, by definition

If two blocks just happen to share characteristics, but could just as well not, there's a good chance you want to change one independent of another

Repetition **(similarity vs. sameness)**

```
#header, #footer {
  background-color: black;
}
#header {
  color: white;
}
#footer {
  color: grey;
}
```

```
#header {
  color: white;
  background-color: black;
}
#footer {
  color: grey;
  background-color: black;
}
```

Ids are usually not good

```
#header, #body {
  width: 80%;
  max-width: 50em;
  min-width: 10em;
}
```

```
.frame-container {
  max-width: 80%;
}
```

Avoid excess nesting (and tools that encourage it)

```
.features .row1 .column2 .detail
s h1 {
  margin-top: 1em;
}
```

```
.title {
  margin-top: 1em;
}
```

Don't touch elements created by plugins

```
/* Override plugin styles to
give more padding to graphs */

#graph-from-plugin {
  padding: 30px;
}
```

```
/* Override plugin styles to
give more padding to graphs */

.my-graph-container {
  padding: 10px;
}
```

Inheritance vs. fake reset

```css
body {
  font-weight: normal;
}

a {
  font-weight: bold;
}

a.discreet  {
  font-weight: normal;
}
```

```css
body {
  font-weight: normal;
}

a {
  font-weight: bold;
}

a.discreet  {
  font-weight: inherit;
}
```

Trust the cascade

```css
/* Bad */
* {
  font-size: 16px;
}
```

```css
/* Good */
bold {
  font-size: 16px;
}
```

Don't try to fix simple problems with unnatural and error-prone CSS:

- display
- position
- top, right, bottom, left
- negative margins
- z-index

That's it!

Jerry Jäppinen

Product design consultant

DCMN elisa DIGITA OLX GROUP ProductBeat

jerryjappinen@lateralnord.com

+358 40 7188776

@jerryjappinen

Lateral Nord.