Be the browser’s mentor, not its micromanager

All Day Hey - May 2022
I’m going to change how you write CSS
Fluid Type
Fluid Space
Flexible Layouts
Progressive Enhancement
We build for everyone
Not just for ourselves, or our peer groups
Everyone should get an excellent user experience
No one will ever complain about getting a good baseline experience
Let’s take a look at what we are building

buildexcellentwebsites.es
#AllDayHey

buildexcellentwebsites.es
Be the browser’s mentor, not its micromanager.

Key Foundations & Principles

Modern CSS with Methodologies

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.
The “C” in CSS stands for “Come on, Andy, get on with the good stuff”
CUBE CSS is a CSS methodology that’s orientated towards simplicity, pragmatism and consistency. It’s designed to work with the medium that you’re working in—often the browser—rather than against it.
Global CSS
Composition
Utilities
Blocks
Exceptions
Global CSS
Composition
Utilities
Blocks
Exceptions
Global CSS
Composition
Utilities
Blocks
Exceptions
Home is where the HTML is
It gives tools that help others consume the information on websites a head start.
If the CSS doesn’t load, the website still makes sense!
Old browsers will still get a good baseline experience
Get the HTML right and you’ve built a solid foundation. If not, you’re building on sand.
<!DOCTYPE html>
<html lang="en">
<head> </head>
<body>
<main class="flow">
  <header class="section spot-color-primary">...</header>
  <article class="region flow">...</article>
  <article class="region">...</article>
  <article class="section spot-color-primary">...</article>
  <article class="signoff region">...</article>
</main>
</body>
</html>
Why You Should Choose HTML5 section

A few days ago, I was having a chat with some friends, one of whom asked me the difference between `<article>` and `<section>` in HTML. This is one of the eternal mysteries of web development, up there with “why is it white-space: nowrap, not white-space: no-wrap?” and “why is CSS ‘gray’ a darker color than ‘darkgray’?”

Bruce Lawson | JAN 7, 2020 / 27 comments

QUICK SUMMARY

Browsers' visual display of headings nested inside `<section>` elements makes it look as if they are assigning a logical hierarchy to those headings. However, this is purely visual and is not communicated to assistive technologies. What use is `<section>`, and how should authors mark up headings that are hugely important to AT users?

ABOUT THE AUTHOR

Bruce has been working on accessibility, web standards, and browsers since 2001. That's why he looks that bad. You can follow him at @bruce, or read his More about Bruce.©

https://www.smashingmagazine.com/2020/01/html5-article-section/
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

- **Modern CSS with Methodologies**
  Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like [CUBE CSS], [SMACSS] or [BEM] that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

- **Fluid type & Space**
  Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

- **Flexible Layouts**
  Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

- **Progressive Enhancement**
  Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.
  
  Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.
  
  Stick to those principles and making excellent websites that work for everyone suddenly becomes much, much easier.

Why though?

It was in 2010 when [Ethan Marcotte] published the legendary Responsive Web Design article. It completely changed how we built websites for an ever-growing variety of device types and sizes.

The article has aged really well, but the practice of web design has not. Oftentimes, designers and developers get stuck into pixel-pushing a design into shape with rigid methods to ensure it looks exactly like that Figma, Sketch or even Photoshop design. This attitude has stuck around for a long time though, even as far back as the very early days of the web, which [Jeremy Keith] touched on in [Resilient Web Design].

It was as though the web design community were participating in a shared consensual hallucination. Rather than acknowledge the flexible nature of the browser window, they chose to settle on one set width as the ideal ...even if that meant changing the ideal every few years.

Jeremy Keith - Resilient Web Design

We absolutely don’t know what our audience device sizes are going to be or whether or not they have a decent connection speed, fully working browser or even a bright enough screen to present our designs how we want them to be presented. We are all guilty of micromanaging the browser in some aspects, and in turn, are creating an inflexible and fragile user experience.

A better way to approach this is to be the browser’s mentor by setting some base rules and hints, then getting out of its way to let it make decisions based on the challenges it will undoubtedly face. Even looking at this 2015 report on Android device sizes tells us just how fragmented devices are. This was also conducted 5 years ago and at the time of writing, it’s 2023. That’s a long time for even more fragmentation to occur. New factors in all of the other brands and types of devices...
Let’s get stuck into some CSS Programming
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

- **Modern CSS with Methodologies**
  Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBIC CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

- **Fluid type & Space**
  Creating type scales that respond to the viewport, rather than setting explicit values for typography and space, allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

- **Flexible Layouts**
  Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—you it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

- **Progressive Enhancement**
  Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.

  Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.

  Stick to those principles and making excellent websites that work for everyone suddenly becomes much, much easier.

Why though?

It was in 2010 when Ethan Marcotte published the legendary Responsive Web Design article. It completely changed how we built websites for an ever-growing variety of device types and sizes.

The article has aged really well, but the practice of web design has not. Oftentimes, designers and developers get stuck into pixel pushing a design into shape with rigid methods to ensure it looks exactly like that Figma, Sketch or even Photoshop design. This attitude has stuck around for a long time though, even as far back as the very early days of the web, which Jeremy Keith touched on in Resilient Web Design:

> It was as though the web design community were participating in a shared consensual hallucination. Rather than acknowledge the flexible nature of the browser window, they chose to settle on one set width as the ideal ...even if that meant changing the ideal every few years.

Jeremy Keith - Resilient Web Design

We absolutely don’t know what our audience device sizes are going to be or whether or not they have a decent connection speed, fully working browser or even a bright enough screen to present our designs how we want them to be presented. We are all guilty of micromanaging the browser in some aspects, and in turn, are creating an inflexible and fragile user experience.

A better way to approach this is to be the browser’s mentor by setting some base rules and hints, then getting out of its way to let it make decisions based on the challenges it will undoubtedly face. Even looking at this 2015 report on Android device sizes tells us how fragmented devices are. This was also conducted 5 years ago and at the time of writing, it’s 2023. That’s a long time for even more fragmentation to occur. New factor in all of the other brands and types of device...
*::before,
*::after {
  box-sizing: border-box;
}

body, h1, h2, h3, h4, p, figure, blockquote, dl, dd {
  margin: 0;
}

ul[role='list'],
ol[role='list'] {
  list-style: none;
}

html {
  text-size-adjust: none;
  -webkit-text-size-adjust: none;
}

html:focus-within {
  scroll-behavior: smooth;
}

body {
  min-height: 300vh;
  text-rendering: optimizeSpeed;
  line-height: 1.5;
}

a:not([class]) {
  text-decoration-skip-ink: auto;
}

img, picture {
  max-width: 100%;
  display: block;
}

input, button, textarea, select {
  font: inherit;
}
Be the browser’s mentor, not its micromanager.
Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

Modern CSS with Methodologies
Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

Flexible Layouts
Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

Progressive Enhancement
Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.
Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.

Stick to those principles and making excellent websites that work for everyone suddenly becomes much, much easier.

Why though?
It was in 2010 when Ethan Marcotte published the legendary Responsive Web Design article. It completely changed how we built websites for an ever-growing variety of device types and sizes.
The article has aged really well, but the practice of web design has not. Offentimes, designers and developers get stuck into pixel-pushing a design into shape with rigid methods to ensure it looks exactly like that Figma, Sketch or even Photoshop design. This attitude has stuck around for a long time though, even as far back as the very early days of the web, which Jeremy Keith touched on in Resilient Web Design.
It was as though the web design community were participating in a shared consensual hallucination. Rather than acknowledge the flexible nature of the browser window, they chose to settle on one set width as the ideal … even if that meant changing the ideal every few years.

Jeremy Keith - Resilient Web Design
We absolutely don’t know what our audience device sizes are going to be or whether or not they have a decent connection speed, fully working browser or even a bright enough screen to present our designs how we want them to be presented.
We are all guilty of micromanaging the browser in some aspects, and in turn, are creating an inflexible and fragile user experience.

A better way to approach this is to be the browser’s mentor by setting some basic rules and hints, then getting out of its way to let it make decisions based on the challenges it will undoubtedly face. Even looking at this 2015 report on Android device sizes tells us just how fragmented devices are. This was also conducted 7 years ago and at the time of writing, it’s 2022. That’s a long time for even more fragmentation to occur. Now factor in all of the other brands and types of device and you can really see how problematic it is, building websites in a rigid, specific manner.

It makes sense to lose a bit of perceived control and instead, get even greater control by being the browser’s mentor and not its micromanger, right?
Go ahead and open up this website on multiple devices or just resize the browser window. You’ll see it deals with anything that is thrown at it. It also uses progressive enhancement to leverage modern CSS, while providing a solid, base experience for browsers that don’t yet support those features, thanks to the usage of semantic HTML. All in, it’s around 2kb of CSS in total.

Tools of the trade

Tools are just tools. They don’t really matter... especially to the people trying to use the websites you build. The same goes for frameworks too. The most important thing is that you stick to the best practices. Even so, here are some useful tools I...
:root {
  --color-primary: #0042bf;
  --color-primary-glare: #d8e2f4;
  --color-secondary: #ee5141;
  --color-secondary-glare: #ffe3e5;
  --space-s: clamp(1rem, 0.92rem + 0.39vw, 1.25rem);
  --space-m: clamp(1.5rem, 1.38rem + 0.58vw, 1.875rem);
  --space-l: clamp(2rem, 1.84rem + 0.78vw, 2.5rem);
  --size-step-1: clamp(1.1875rem, 1.01rem + 0.87vw, 1.75rem);
  --size-step-2: clamp(1.4375rem, 1.11rem + 1.65vw, 2.5rem);
  --size-step-3: clamp(1.75rem, 1.19rem + 2.82vw, 3.5625rem);
}
1: root {
  --color-primary: #0042bf;
  --color-primary-glare: #d8e2f4;
  --color-secondary: #ee5141;
  --color-secondary-glare: #ffe3e5;
  --space-s: clamp(1rem, 0.92rem + 0.39vw, 1.25rem);
  --space-m: clamp(1.5rem, 1.38rem + 0.58vw, 1.875rem);
  --space-l: clamp(2rem, 1.84rem + 0.78vw, 2.5rem);
  --size-step-1: clamp(1.1875rem, 1.01rem + 0.87vw, 1.75rem);
  --size-step-2: clamp(1.4375rem, 1.11rem + 1.65vw, 2.5rem);
  --size-step-3: clamp(1.75rem, 1.19rem + 2.82vw, 3.5625rem);
}

root {
  --gutter: var(--space-s-m);
  --border-radius: var(--size-step-1);
  --transition-base: 250ms ease;
  --transition-movement: 200ms linear;
  --transition-fade: 200ms ease;
  --transition-bounce: 500ms cubic-bezier(0.5, 0.05, 0.2, 1.5);
  --tracking: -0.05ch;
  --tracking-s: -0.075ch;
}
body {
  color: var(--color-dark);
  background: var(--color-light);
  font-size: var(--size-step-1);
  font-family: var(--font-base);
  line-height: 1.4;
  letter-spacing: var(--tracking);
}
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

Modern CSS with Methodologies
Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

Flexible Layouts
Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

Progressive Enhancement
Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.
Fluid type and fluid space
HTML, CSS

and a sprinkling of

JavaScript

It’s all you need to build a solid, resilient and importantly, inclusive front-end that everyone can enjoy, regardless of their device power, connection speed and environment.
HTML, CSS
and a sprinkling of JavaScript

It's all you need to build a solid, resilient and important, inclusive front-end that everyone can enjoy, regardless of their device power, connection speed and environment.
.header__display-text {
  font-size: 2rem;
}

.header__display-text:first-of-type {
  font-size: 2.3rem;
}

@media (min-width: 768px) {
  .header__display-text {
    font-size: 6rem;
  }
}

.header__display-text:first-of-type {
  font-size: 8rem;
}

@media (min-width: 1100px) {
  .header__display-text {
    font-size: 8rem;
  }
}

.header__display-text:first-of-type {
  font-size: 10rem;
}
@media (min-width: 768px) {
  .header__display-text {
    font-size: 6rem;
  }
}

.header__display-text:first-of-type {
  font-size: 8rem;
}

@media (min-width: 856px) {
  .header__display-text {
    font-size: 7rem;
  }
}

.header__display-text:first-of-type {
  font-size: 9rem;
}

@media (min-width: 1100px) {
  .header__display-text {
    font-size: 8rem;
  }
}

.header__display-text:first-of-type {
  font-size: 10rem;
}
HTML, CSS

and a sprinkling of JavaScript

It's all you need to build a solid, resilient and importanty, inclusive front-end that everyone can enjoy, regardless of their device power, connection speed and environment.
.my-element {
  font-size: clamp(2rem, calc(1rem + 5vw), 10rem);
}

.my-element {
    font-size: clamp(2rem, calc(1rem + 5vw), 10rem);
}

HTML, CSS
and a sprinkling of
JavaScript

It's all you need to build a solid, resilient and importantly, inclusive front-end that everyone can enjoy, regardless of their device power, connection speed and environment.
Size and space scales
Step 0: 1rem
Step 1: 1.25rem \( (1 \times 1.25) \)
Step 2: 1.56rem \( (1.25 \times 1.25) \)
Step 3: 1.95rem \( (1.56 \times 1.25) \)
Step 4: 2.43rem \( (1.95 \times 1.25) \)
**UTOPIA.**

**FLUID TYPE SCALE CALCULATOR**

<table>
<thead>
<tr>
<th>MIN VIEWPORT</th>
<th>MAX VIEWPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Width</td>
</tr>
<tr>
<td>320 px</td>
<td>1140 px</td>
</tr>
<tr>
<td>Font size</td>
<td>Font size</td>
</tr>
<tr>
<td>21 px</td>
<td>24 px</td>
</tr>
<tr>
<td>Type scale</td>
<td>Type scale</td>
</tr>
<tr>
<td>1.2</td>
<td>1.25</td>
</tr>
</tbody>
</table>

**CALculated FONT SIZES**

This table lists font size values in px for your type scales at the min and max viewport widths entered above.

Add a **viewport width** to show its corresponding font size values.

Add a **scale step** to extend your scale up or down.
FLUID TYPE SCALE CALCULATOR

MIN VIEWPORT

<table>
<thead>
<tr>
<th>Width (px)</th>
<th>Font size (px)</th>
<th>Type scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>21</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Calculation: Minor Third

MAX VIEWPORT

<table>
<thead>
<tr>
<th>Width (px)</th>
<th>Font size (px)</th>
<th>Type scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1140</td>
<td>24</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Calculation: Major Third

CALCULATED FONT SIZES

This table lists font size values in px for your type scales at the min and max viewport widths entered above.

Add a viewport width to show its corresponding font size values.

Add a scale step to extend your scale up or down.

<table>
<thead>
<tr>
<th>Scale step</th>
<th>Viewport width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>1140</td>
</tr>
<tr>
<td>MIN VIEWPORT</td>
<td>MAX VIEWPORT</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td><strong>Width</strong></td>
</tr>
<tr>
<td>320 px</td>
<td>1350 px</td>
</tr>
<tr>
<td><strong>Font size</strong></td>
<td><strong>Font size</strong></td>
</tr>
<tr>
<td>16 px</td>
<td>20 px</td>
</tr>
<tr>
<td><strong>Type scale</strong></td>
<td><strong>Type scale</strong></td>
</tr>
<tr>
<td>1.2</td>
<td>1.414</td>
</tr>
</tbody>
</table>

*Minor Third*  *Augmented Fourth*
root {
  --step-1: clamp(0.83rem, calc(0.82rem + 0.08vw), 0.88rem);
  --step-0: clamp(1.00rem, calc(0.92rem + 0.39vw), 1.25rem);
  --step-1: clamp(1.20rem, calc(1.02rem + 0.88vw), 1.77rem);
  --step-2: clamp(1.44rem, calc(1.11rem + 1.65vw), 2.50rem);
  --step-3: clamp(1.73rem, calc(1.17rem + 2.80vw), 3.53rem);
  --step-4: clamp(2.07rem, calc(1.17rem + 4.54vw), 5.00rem);
  --step-5: clamp(2.49rem, calc(1.07rem + 7.11vw), 7.07rem);
  --step-6: clamp(2.99rem, calc(0.81rem + 10.88vw), 9.99rem);
}
<table>
<thead>
<tr>
<th>MIN VIEWPORT</th>
<th>MAX VIEWPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td><strong>Width</strong></td>
</tr>
<tr>
<td>320 px</td>
<td>1350 px</td>
</tr>
<tr>
<td><strong>Font size</strong></td>
<td><strong>Font size</strong></td>
</tr>
<tr>
<td>16 px</td>
<td>20 px</td>
</tr>
<tr>
<td><strong>Type scale</strong></td>
<td><strong>Type scale</strong></td>
</tr>
<tr>
<td>Minor Third</td>
<td>Augmented Fourth</td>
</tr>
<tr>
<td>1.2</td>
<td>1.414</td>
</tr>
<tr>
<td>Multiplier</td>
<td>@min</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>3XS</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2XS</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>XS</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>XL</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

Modern CSS with Methodologies

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Key Foundations & Principles
body {
  color: var(--color-dark);
  background: var(--color-light);
  font-size: var(--size-step-1);
  font-family: var(--font-base);
  line-height: 1.4;
  letter-spacing: var(--tracking);
}
h1, h2, h3 {
  line-height: 1;
  letter-spacing: var(--tracking-s);
}

h1 {
  font-size: var(--size-step-5);
}

h2 {
  font-size: var(--size-step-4);
}

h3 {
  font-size: var(--size-step-3);
}
p, li, blockquote:not([class]) {
  max-width: 50ch;
}

h1, h2, h3 {
  max-width: 20ch;
}
```css
blockquote:not([class]) {
  font-family: var(--font-serif);
  font-size: var(--size-step-2);
}

blockquote:not([class]) p:last-of-type {
  font-family: var(--font-base);
  font-size: var(--size-step-1);
  font-weight: normal;
}

svg {
  height: 2ex;
  width: auto;
  flex: none;
}

a {
  color: currentcolor;
}

a:hover {
  text-decoration: none;
}

:focus {
  outline: 2px solid;
  outline-offset: 0.3ch;
}

:target {
  scroll-margin-top: 2ex;
}
```
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

**Key Foundations & Principles**

**Modern CSS with Methodologies**

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

**Fluid type & Space**

Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of
Global CSS
Composition
Utilities
Blocks
Exceptions
.flow > * + * {
  margin-top: var(--flow-space, 1em);
}
Hello I am a heading

Sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Donec ullamcorper nulla non metus auctor fringilla.

Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Donec id elit non mi porta gravida at eget metus.

A subheading

Sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Donec ullamcorper nulla non metus auctor fringilla.

Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Donec id elit non mi porta gravida at eget metus.
.my-context {
  --flow-space: 10rem;
}

.flow > * + * {
  margin-top: var(--flow-space, 1em);
}
Hello I am a heading

Sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Donec ullamcorper nulla non metus auctor fringilla.

A subheading

Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Donec id elit non mi porta gravida at eget metus.
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles
.region {
    padding-top: var(--region-space, var(--space-l-2xl));
    padding-bottom: var(--region-space, var(--space-l-2xl));
}
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

Modern CSS with Methodologies
Global CSS
Composition
Utilities
Blocks
Exceptions
Key Foundations & Principles

**Modern CSS with Methodologies**
Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

**Fluid type & Space**
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

**Flexible Layouts**
Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size—your front-

**Progressive Enhancement**
Building up with the lowest possible technological solution and enhancing it
CUBE CSS
A CSS methodology that’s orientated towards simplicity, pragmatism and consistency.

Utopia
A handy collection of generators and tools that let you build up various fluid type and space scales depending on viewport sizes to help with responsive design.

Every Layout
A series of simple, composable layouts that can be ported to any project. There’s also heaps of learning material to help you really learn CSS layout.

Design Tokens
Centralised data—almost like a database of design values—that could be consumed by anything that understands a standard, like JSON to help with design consistency.

Polypane
A handy app that lets you spin up multiple viewports in various configurations to help you build truly responsive sites.

Tailwind
A utility-first CSS framework that is very useful for generating utility classes on demand for CUBE CSS.
.grid {
  display: grid;
  grid-template-columns: repeat(
    var(--grid-placement, auto-fill),
    minmax(var(--grid-min-item-size, 16rem), 1fr)
  );
  gap: var(--gutter, var(--space-s-l));
}
Key Foundations & Principles

• **Modern CSS with Methodologies**
  Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

• **Fluid type & Space**
  Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

• **Flexible Layouts**
  Using flexible, flexbox-based layouts, like the ones we provide in *Every Layout*, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way.
  Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where...

• **Progressive Enhancement**
  Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.
  Doing the opposite:
```html
1 <ul class="grid" role="list">
2   ...
3 </ul>
```
[role='list'] {
  padding: 0;
}
Key Foundations & Principles

**Modern CSS with Methodologies**

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

**Fluid type & Space**

Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

**Flexible Layouts**

Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where doing the opposite: building the best...

**Progressive Enhancement**

Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.
.grid[data-layout='50-50'] {
  --grid-placement: auto-fit;
  --grid-min-item-size: clamp(16rem, 50vw, 26rem);
}
Key Foundations & Principles
Modern CSS with Methodologies

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM. This empowers you to write flexible, portable CSS, rather than rigid, inflexible and over-specified CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles
Modern CSS with Methodologies

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM. This empowers you to write flexible, portable CSS, rather than rigid, inflexible and over-specified CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Flexible Layouts
Using flexible, flexbox-based layouts, like those provided by Flexbox, empowers you to make the best use of space across devices.
Global CSS
Composition
Utilities
Blocks
Exceptions
Key Foundations & Principles

Modern CSS with Methodologies
Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

Flexible Layouts
Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size—your front-

Progressive Enhancement
Building up with the lowest possible technological solution and enhancing it
.card {
  background: var(--color-dark);
  color: var(--color-light);
  padding: var(--space-m-l);
  border-radius: var(--border-radius);
  max-width: unset;
}
p,
li,
blockquote:not([class]) {
  max-width: 50ch;
}
.card {
  background: var(--color-dark);
  color: var(--color-light);
  padding: var(--space-m-l);
  border-radius: var(--border-radius);
  max-width: unset;
}
Modern CSS with Methodologies

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Fluid type & Space

Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

Flexible Layouts

Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

Progressive Enhancement

Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time. Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.
Modern CSS with Methodologies
Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

Flexible Layouts
Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

Progressive Enhancement
Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.

Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.
<ul class="grid" role="list" data-rows="masonry" data-layout="50-50">
  ...
</ul>
.grid[data-rows='masonry'] {
  grid-template-rows: masonry;
  align-items: start;
}

Modern CSS with Methodologies
Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

Fluid type & Space
Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

Flexible Layouts
Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.

Progressive Enhancement
Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.

Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.
Key Foundations & Principles

**Modern CSS with Methodologies**

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.

**Fluid type & Space**

Creating type scales that respond to the viewport, rather than setting explicit values for typography and space allows you to set rules once and forget about them, knowing that whatever device, regardless of its available size will be presented with appropriate sizes.

**Progressive Enhancement**

Building up with the lowest possible technological solution and enhancing it where device capability, connection speeds and context conditions allow, helps you build for everyone, not just the minority of people that have fast connections and powerful devices that work well, all the time.

Doing the opposite: building the best experience, then hacking it down for a handful of selected edge-cases means you’re almost certainly going to build an experience that’s excludes a lot of people.

**Flexible Layouts**

Using flexible, flexbox-based layouts, like the ones we provide in Every Layout, ensures that regardless of conditions—be it content or available screen size: your front-end will be able to respond in the most appropriate way. Giving browsers hints and space to do what they do best, helps your front-end handle tricky scenarios where breakpoint-based layouts consistently fail.
CUBE CSS
A CSS methodology that's orientated towards simplicity, pragmatism and consistency.

Utopia
A handy collection of generators and tools that let you build up various fluid type and space scales depending on viewport sizes to help with responsive design.

Every Layout
A series of simple, composable layouts that can be ported to any project. There's also heaps of learning material to help you really learn CSS layout.

Design Tokens
Centralised data—almost like a database of design values—that could be consumed by anything that understands a standard, like JSON to help with design consistency.

Polypane
A handy app that lets you spin up multiple viewports in various configurations to help you build truly responsive sites.

Tailwind
A utility-first CSS framework that is very useful for generating utility classes on demand for CUBE CSS.
.features {
  --grid-placement: auto-fit;
  --grid-min-item-size: clamp(16rem, 33%, 20rem);
  --gutter: var(--space-l-xl);
  --flow-space: var(--space-s);
  text-align: center;
}
.features svg {
  display: block;
  margin-inline: auto;
  height: 4em;
}

.features a {
  text-decoration: none;
}

.features a:hover {
  text-decoration: underline;
  text-decoration-thickness: 0.08ex;
  text-underline-offset: 0.2ex;
}
Be the browser’s mentor, not its micromanager.

Give the browser some solid rules and hints, then let it make the right decisions for the people that visit it, based on their device, connection quality and capabilities. This is how they will get a genuinely great user experience, rather than a fragmented, broken one.

Key Foundations & Principles

Modern CSS with Methodologies

Instead of brute-forcing your designs together with a CSS framework, consider opting for a CSS methodology like CUBE CSS, SMACSS or BEM that empowers you to write flexible, portable CSS, rather than rigid, inflexible and overly-specific CSS.
We’ve never had it better with browsers
Build the whole website with Tailwind

Use Tailwind as a utility generator and lean into CSS
A couple of examples of this approach in the wild
Welcome to Learn CSS!

This course breaks down the fundamentals of CSS into digestible, easy to understand pieces. Over the next few modules, you'll learn the core aspects of CSS work and how to use them effectively in your projects. Use the menu pane by the "Learn CSS" logo to navigate the modules.

You'll learn CSS fundamentals like the box model, cascade and specificity, flexbox, grid and z-index. And, along with these fundamentals, you'll learn about functions, color types, gradients, logical properties and inheritance to make you a well-rounded frontend developer, ready to take on any UI/UX interface.

Each module is full of interactive demos and self-assessments for you to test your knowledge. In addition to learning through reading and demos, there is an accompanying podcast episode for each topic as another way to learn and continue expanding your knowledge.

This course is created for beginner and advanced CSS developers alike. You can go through the series from start to finish to get a general understanding of CSS from top to bottom, or you can use it as a reference for specific styling subjects. For those new to web development overall, check out the [Code In Pro] course from MDN to learn all about how to write markup and link your stylesheets.
CONTENT WARNING

Glitching and flashing
A Human Future is a software development house.

We build products, platforms and prototypes that help ambitious organisations pull their future towards them, faster.

CAMBRIDGE & LONDON
Go forth and build excellent websites
Thank you