Responsive Web Design

History lesson

The idea behind Responsive Web Design is using the same HTML at different screen sizes, such as mobile devices and widescreen desktops.

This technique has been an industry best practice for over a decade because it avoids the need to create separate HTML for different screen sizes.

In the past, some websites would create a “mobile version” that would be entirely different from the regular site.

Problems with Mobile-version Websites

These caused two main issues:

1. The amount of work needed to maintain two separate websites.
2. Users could not always find the same content or perform the same actions when they switched between mobile and desktop. Mobile versions were rarely at “feature parity” with the regular site.

The fluid nature of web content

In an HTML document without CSS, browsers automatically:

- Wrap lines of text
- Allow for scrolling

These are based on the available screen space. If the user changes the window size, the browser will automatically readjust to fit.

Static representations of web layouts are always misleading

With more complex layouts, designers must make decisions about how the content should adjust to fit at different screen sizes.

At least two design mockups are needed to build a web page — horizontal (desktop) and vertical (mobile).

Even with two mockups, designers cannot foresee every possible screen size, combined with other factors like zoom level and system settings. Defer to the browser's natural behavior whenever possible.

Adapative vs Responsive

Designers must also decide whether the layout will be Adaptive or fully Responsive.

1. Adaptive Layouts
    - Have fixed widths at different breakpoints
2. Responsive Layouts
    - No fixed widths, mostly percentage-based

Mobile considerations

On mobile, there are a few differences from desktop:

- Small overall area (“real estate”)
  - Hamburger menus hide navigation behind a button
- Narrow screen
  - Most things are 100% width, not in columns
- Finger as pointer
  - Not as precise, so the element sizes should be at least 48px x 48px
- Data plans and possibly poor data connection
  - Don’t send unnecessary desktop styles until needed

Don't Forget This One Line of Code!

The viewport must be set manually in the HTML.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

CSS Box Model

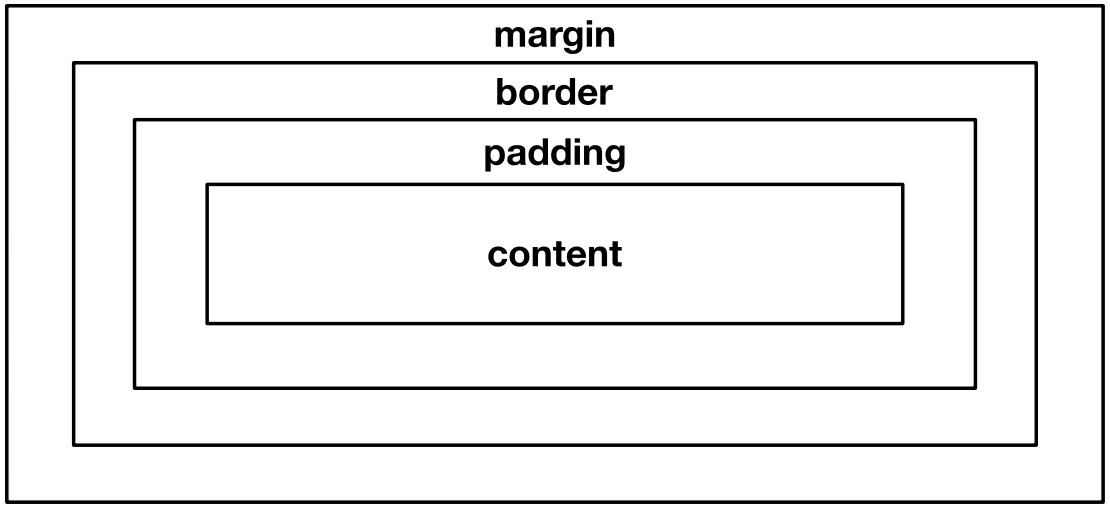
Web layout is the arrangement of boxes

Everything in web design is a box. Not everything necessarily looks like a box, but everything takes up a box-like space.

The size and shape of an HTML element are determined by its contents, unless given an explicit height or width.

Margin, Border, and Padding

Diagram of the Box Model:



An HTML element can have:

- Border
- Outer margin
- Inner padding

Debug the box model using the web inspector

1. Right-click
2. Inspect element
3. Select Elements tab
4. Select an HTML element
5. See Styles panel

CSS Syntax

Anatomy of a CSS rule

HTML has elements. CSS has rules.

CSS rules have 3 parts:

- Selector: Targets one or more HTML elements.
- Property: A known CSS keyword that controls some aspect of the style
- Value: What the property is being set to

```
/* <selector></selector> */

selector {
  property: value;
}
```

Targeting HTML elements for styling

By HTML element name:

```
/* <bobs-element>Hello world!</bobs-element> */

bobs-element {
  background-color: red;
}

h1 {
  font-size: 16px;
}
```

With a class:

```
/* <div class="bobs-class"></div> */

.bobs-class {
  background-color: red;
}
```

Overrides

If there are duplicates, the last one down will apply:

```
bobs-element {
  font-size: 64px;
}

bobs-element {
  font-size: 24px;
}
```

Specificity

Selectors are weighted (by “specificity”) when you combine them:

```
div.bobs-class {
  background-color: green;
}

.bobs-class {
  background-color: red;
}
```

Variables/Custom Properties

Define custom properties at the top level:

```
:root {
  --my-custom-property: 12px;
}
```

Use multiple times with the `var()` keyword:

```
p {
  margin-top: var(--my-custom-property);
}

img {
  padding: var(--my-custom-property);
}
```

Calculations

Use custom properties in calculations with the `calc()` keyword:

```
h1 {
  font-size: calc(var(--my-custom-property) * 2);
}
```

CSS Resets

Overriding Browser Styles

Web browsers apply default styles to built-in elements. Web developers tend to override some of these defaults to make development easier.

```
* {
  box-sizing: border-box;
}

body {
  margin: 0;
  -webkit-text-size-adjust: 100%;
}

img {
  max-width: 100%;
}
```

Inspecting Browser Styles

You can use the Web Inspector to see the browser's default styles.

```
button {
  padding: 1px 6px;
  /* ... */
  /* See inspector for all properties. */
}
```

You can override the defaults, or remove them with the `unset` keyword.

Here, with `unset`, all <button>s will appear like s:

```
button {
  background: unset;
  border: unset;
  color: unset;
  font: unset;
  padding: unset;
}
```

CSS Layout

Block Layout

- `display: block`
- Default for and most other elements
- Takes up all of the available horizontal space
- Block elements stack on top of each other
- Best for blocks of flowing text

Flex Layout

- `display: flex`
- The parent element continues to behave like a block element, but it changes the way its child elements are laid out — into rows OR columns.
- Children flow in one dimension.
  - By default, with just `display: flex` applied to the parent, the children will be distributed horizontally.
  - The flow direction can be switched from horizontal back to vertical with `flex-direction: column`.
- Children size themselves with the `flex` property.
- Children can wrap if there's too many to fit in one row/column.
  - `flex-wrap: wrap`

Grid Layout

- `display: grid`
- The parent element continues to behave like a block element, but it changes the way its child elements are laid out — into rows AND columns like a table.
- Children flow in two dimensions, determined by parent.
- Parent defines the size of children.

Centering

```
<div class="centered">Hello world!</div>
<style>
  .centered {
    display: flex; /* or display: grid */
    height: 100%;
    place-items: center;
    place-content: center;
  }
</style>
```

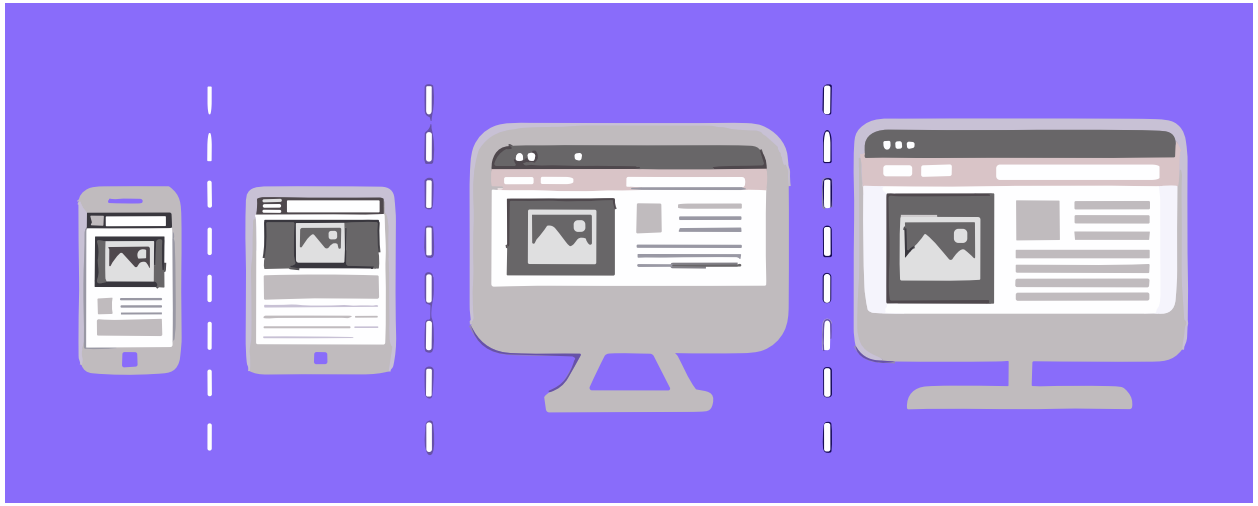
Centering Block Elements Horizontally

```
<div class="horizontally-centered">Hello world!</div>
<style>
  .horizontally-centered {
    margin-left: auto;
    margin-right: auto;
    width: 50%;
  }
</style>
```

CSS Media Queries

Targeting different screen widths

The point at which the layout switches from one design to another is called a breakpoint.



Building Mobile-First

Building mobile-first means mobile styles are not in a media query.

```
/* Mobile */
body {
  font-size: 16px;
}

/* Tablet */
@media (min-width: 800px) {
  body {
    font-size: 18px;
  }
}

/* Wide screen */
@media (min-width: 1200px) {
  body {
    font-size: 20px;
  }
}
```

Handling user preferences

When possible, design to fit the user's system preferences.

Some users prefer darker colors on their screen due to light fatigue or other reasons.

Accomodate those users whenever possible by providing an alternate design.

```
body {
  background-color: lightyellow;
}

@media (prefers-color-scheme: dark) {
  body {
    background-color: darkpurple;
  }
}
```

Handling Accessibility Preferences

Some system preferences define accessibility settings.

Animations should always be gated so as not to disturb those with motion disorders.

```
@media (prefers-reduced-motion: no-preference) {
  .spinner {
    animation: spin-around-the-page 2s;
  }
}
```

Browser Support

Where does CSS come from?

CSS is changing all the time.

Over the years, new CSS has given web developers new powers:

- Media queries
- Animations and transitions
- Photoshop-style filters

New CSS properties and syntax are introduced through an open process that includes the companies that design and code web browser engines:

Chromium

- Chrome (Google)
- Edge (Microsoft)

WebKit

- Safari (Apple)

Gecko

- Firefox (Mozilla)

Google, Microsoft, Apple, and Mozilla listen to developer input and decide together whether to make and implement changes to the CSS specification.

Cross-Browser Testing

Unfortunately, the browsers may have slight differences in their imlementations.

It's important to check the apperance and functionality of a web page in all browsers, because there might be differences or even bugs.

Automatic Updates

Browsers used to require users to download new versions, or were tied to operating systems that updated every few years.

This caused problems because browsers were adopting new CSS at a very slow rate.

Now all browsers are "evergreen" one-time downloads that update themselves automatically, or are tied to more frequent operating system updates.

There are some cases where older browsers need to be considered, but now it's easier to assume that most users are on the latest technology.

Example: CSS Nesting

This year, a new CSS syntax was introduced into browsers.

To target these elements:

```
<div class="a">
  Red
  <div class="b">
    Green
    <div class="c">Blue</div>
  </div>
</div>
```

Old Style of CSS Nesting

```
.a {
  color: red;
}

.a .b {
  color: green;
}

.a .b .c {
  color: blue;
}
```

CSS Nesting Since 2023

```
.a {
  color: red;

  .b {
    color: green;

    .c {
      color: blue;
    }
  }
}
```

caniuse.com

For browser support, see: https://caniuse.com/css-nesting

When you discover new CSS, you can check the website caniuse.com to see which browsers have implemented it.