CS0060

Philipp Eichmann

HTML/CSS/JS

BASICS

HTML DOCUMENT STRUCTURE

```
<!DOCTYPE html>
<html>

    <head>
        <title>Page Title</title>
    </head>

    <body>
        <h1>My First Heading</h1>
        <p>My first paragraph.</p>
    </body>

</html>
```

HTML5 marker

Contains the entire HTML document

Contains metadata, stylesheet/javascript imports

Contains the actual HTML content

COMMON TAGS

```
<h1>Heading 1</h1>

<h2>Heading 2</h2>

...
```

The **HTML `<h1>`–`<h6>` elements** represent six levels of section headings. `<h1>` is the highest section level and `<h6>` is the lowest.

COMMON TAGS

```
<p>Lorem ipsum dolor est ...</p>
```

The **HTML `<p>` element** represents a paragraph. HTML paragraphs can be any structural grouping of related content, such as images or form fields

COMMON TAGS

```
<a href="http://www.brown.edu">Brown University</a>
```

The **HTML `<a>` element** (or *anchor* element), with its `href` attribute, creates a hyperlink to web pages, files, email addresses, locations in the same page, or anything else a URL can address

COMMON TAGS

```
<img src="logo.png" />

<video src="movie.mp4" />
```

The **HTML `<img>` element** embeds an image into the document.

The **HTML Video element** (**`<video>`**) embeds a media player which supports video playback into the document.

COMMON TAGS

```
<table>
    <thead>
        <tr>
            <th>A</th>
            <th>B</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>C</td>
            <td>D</td>
        </tr>
    </tbody>
</table>
```

The **HTML `<table>` element** represents tabular data — that is, information presented in a two-dimensional table comprised of rows and columns of cells containing data.

COMMON TAGS

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ul>
```

The **HTML** `<ul>` and `<ol>` **element** represent unordered and ordered lists of items.

COMMON TAGS

```
<section>
  ...
</section>
```

The **HTML `<section>` element** represents a standalone section — which doesn't have a more specific semantic element to represent it — contained within an HTML document. Typically, but not always, sections have a heading.

COMMON TAGS

```
<header>
  ...
</header>
```

The **HTML `<header>` element** represents introductory content, typically a group of introductory or navigational aids. It may contain some heading elements but also a logo, a search form, an author name, and other elements.

COMMON TAGS

```
<nav>

  ...

</nav>
```

The **HTML `<nav>` element** represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. Common examples of navigation sections are menus, tables of contents, and indexes.

COMMON TAGS

```
<footer>

  ...

</footer>
```

The **HTML `<footer>` element** represents a footer for its nearest sectioning content or sectioning root element. A footer typically contains information about the author of the section, copyright data or links to related documents.

COMMON TAGS

```
<div>

  ...

</div>
```

The **HTML Content Division element** (`<div>`) is the generic container for flow content. It has no effect on the content or layout until styled using CSS. It should be used only when no other semantic element is appropriate.

COMMON TAGS

```
<span>normal text</span>

<span>highlighted text<span>
```

The **HTML `<span>` element** is a generic inline container for phrasing content, which does not inherently represent anything. It should be used only when no other semantic element is appropriate. `<span>` is very much like a `<div>` element, but `<div>` is a block-level element whereas a `<span>` is an inline element.

HTML DOCUMENT EXAMPLE

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <header>
            <nav>
                <ul>
                    <li>Page 1</li>
                    <li>Page 2</li>
                    <li>Page 3</li>
                </ul>
            </nav>
        </header>
        <h1>Heading</h1>
        <p>Paragraph with a <a href="#">link</a>.</p>
        <img src="brown.png" />
        <footer>
            <span>Copyright by Brown University</span>
        </footer>
    </body>
</html>
```

HTML RENDERING

How does a browser know how to render a html document?

It uses default styles for every tag

DEMO

OVERRIDING THE DEFAULT STYLES

Method 1: Inline styles

```
<a style="color: #ff0000; text-decoration: none;">Link</a>
```

Generally bad practice because

- styles are not re-useable
- convolutes HTML markup
- Cannot be used to style pseudo elements (not covered in this lecture)

OVERRIDING THE DEFAULT STYLES

Method 2: CSS Stylesheets in HTML

```
<html>
    <head>
        <style>
            a {
                color: #ff0000;
                text-decoration: none;
            }
        </style>
    </head>
    ...
<html>
```

Generally bad practice because

- styles are not re-useable in other HTML documents
- convolutes HTML markup

OVERRIDING THE DEFAULT STYLES

Method 3: Importing CSS Stylesheets

```
<html>
    <head>
        <link href=“stylesheet.css" rel="stylesheet">
    </head>
    ...
<html>
```

This is the recommended way of using CSS!

CSS STYLE SHEETS

HTML

```
<body>
    <a href="…">Click me</a>
    <p>Lorem ipsum dolor est…<p>
</body
```

CSS Style sheet

Selectors	Properties	Values
a {		
	color:	#ff0000;
	text-decoration:	none;
}		
p {		
	line-height:	1.5;
	margin:	10px 10px;
}		

DISAMBIGUATING HTML ELEMENTS USING CSS

```
<body>
    <a href="…">Click me</a>
    <p>Lorem ipsum dolor est…</p>
    <footer>
        <a href="…">Back</a>
    </footer>
</body
```

```
a {
    color:              red;
    text-decoration:    underline;
}

footer a {
    color:              green;
}
```

SELECTOR SPECIFICITY MATTERS

An HTML Element's final CSS property is defined

by the CSS rule with the most specific selector

DEMO

DISAMBIGUATING HTML ELEMENTS USING CSS

```
<body>
    <p>Lorem ipsum dolor est…</p>
    <header>
        <a href="…">Email me</a>
        <a href="…">Back</a>
    </header>
</body
```

Tags of the same kind cannot (easily) be disambiguated if they have same parent.

REFERENCING HTML ELEMENTS DIRECTLY

```
<body>
    <header>
        <a id="link-a" href="…">Home</a>
        <a class="link-b" href="…">Back</a>
    </header>
    <a href="…">Click me</a>
    <p>Lorem ipsum dolor est…</p>
</body
```

```
a {
    text-decoration: underline;
}

#link-a {
    color:              red;
}

.link-b {
    color:              green;
}
```

ID VS CLASS

`id`

The `id` attribute assigns a unique identifier to an element

`class`

The `class` attribute, assigns one **or more** class names to an element.
A class name may be shared by several element instances.

Unless otherwise required, always use `class`

(even if there's only a single element using it)

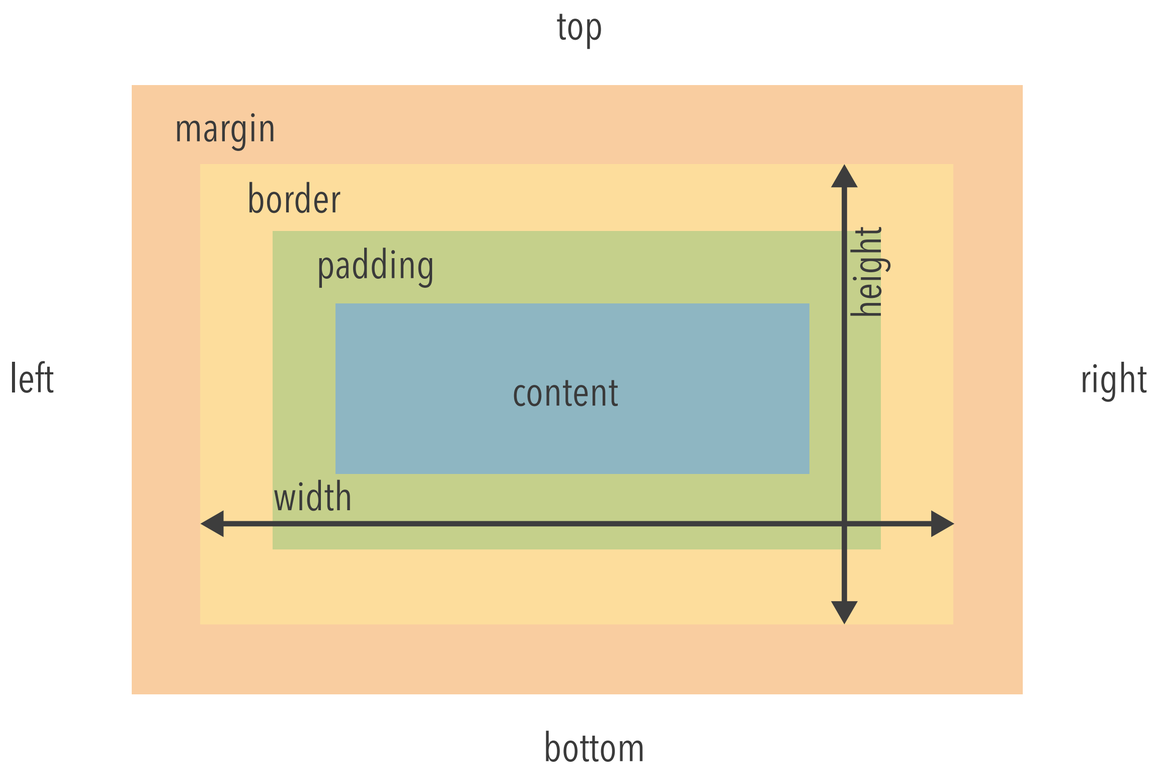
USING CSS TO LAY OUT HTML DOCUMENTS

HTML RENDERING

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <header>
            <nav>
                <ul>
                    <li>Page 1</li>
                    <li>Page 2</li>
                    <li>Page 3</li>
                </ul>
            </nav>
        </header>
        <h1>Heading</h1>
        <p>Paragraph with a <a href="#">link</a>.</p>
        <img src="brown.png" />
        <footer>
            <span>Copyright by Brown University</span>
        </footer>
    </body>
</html>
```

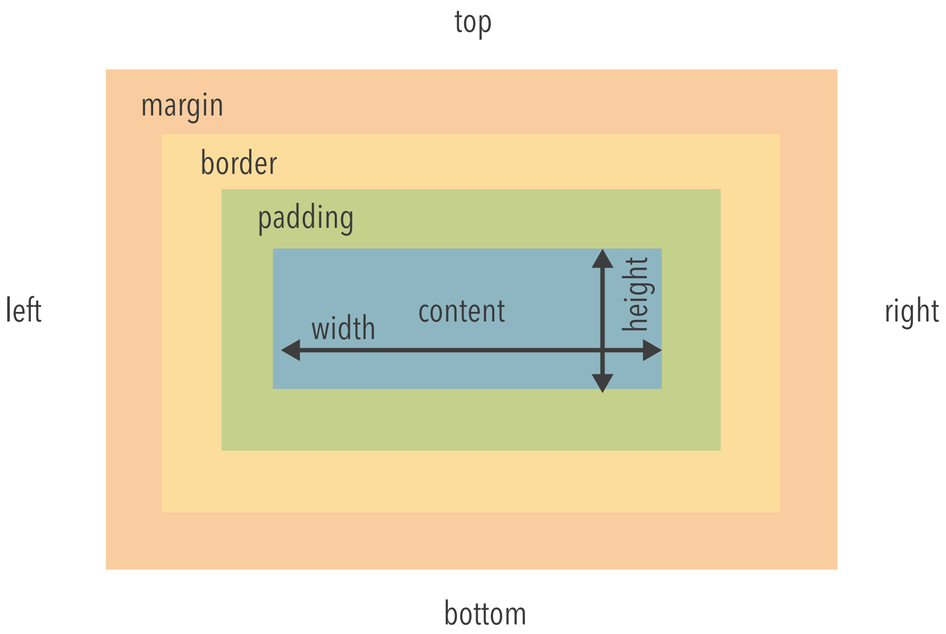
DEMO

CSS BOX MODEL

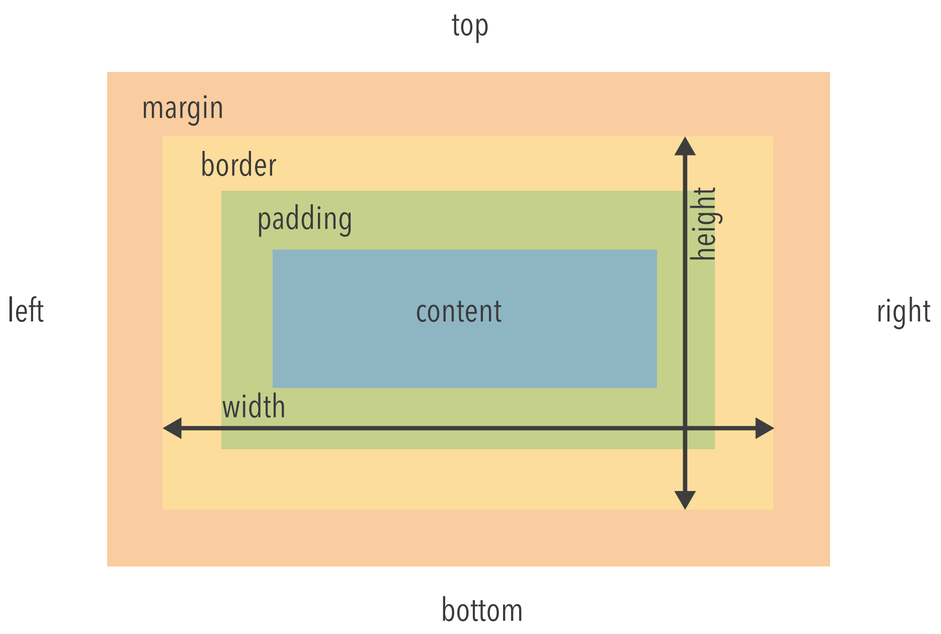


CSS BOX MODEL

`box-sizing: content-box;`



`box-sizing: border-box;`



THE TWO FUNDAMENTAL LAYOUT PROPERTIES

How can we arrange these boxes to build a custom layout?

`display`

`position`

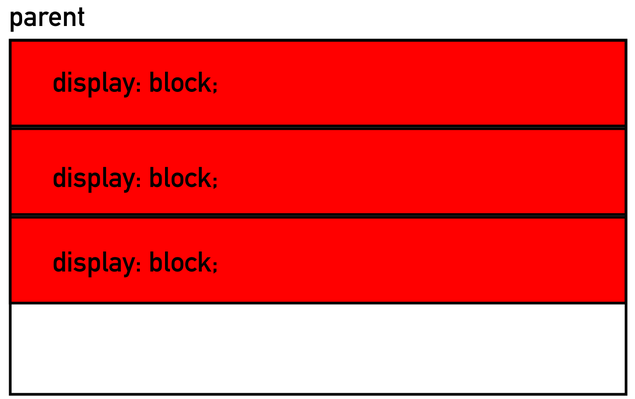
THE "DISPLAY" PROPERTY

The `display` CSS property sets whether an element is treated as a block or inline element, and the layout used for its children, such as flow layout, grid or flex.

THE "DISPLAY" PROPERTY

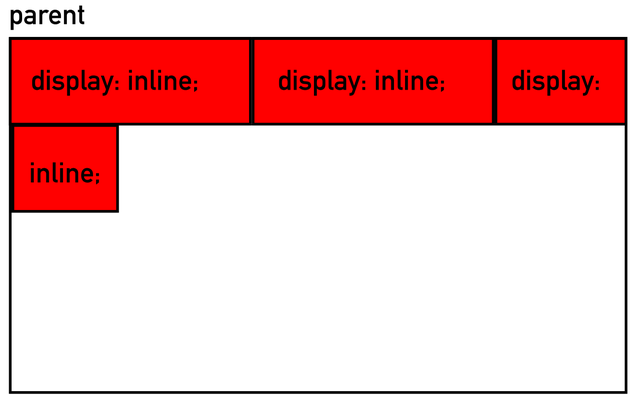
`display: block;`

- Takes up parent's full width
- Consecutive block elements are stacked vertically



`display: inline;`

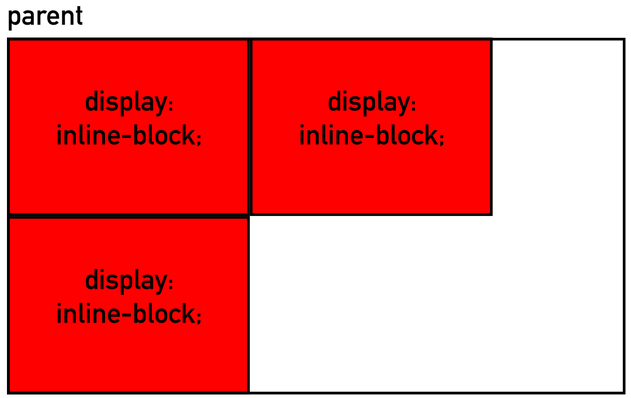
- Width based on content's width
- Consecutive inline elements are placed on same line, if possible, and automatically break to the next line
- Can only contain other inline elements, no block elements



THE "DISPLAY" PROPERTY

`display: inline-block;`

- Arranges block elements like inline elements on the same line
- Breaks on next line if necessary



There are other display values, such as `table`, `flex`, `grid`, etc.

THE "POSITION" PROPERTY

The `position` CSS property sets how an element is positioned in a document. The `top`, `right`, `bottom`, and `left` properties determine the final location of positioned elements.

THE "POSITION" PROPERTY

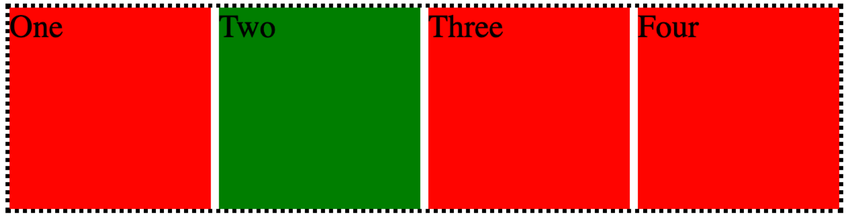
`position: static;`	→	*Un-positioned*
`position: relative;`	→	Position element relative to its `static` position
`position: absolute;`	→	Position element relative to the position of its closest *positioned* ancestor.
`position: fixed;`	→	Position element relative to the viewport

THE "POSITION" PROPERTY

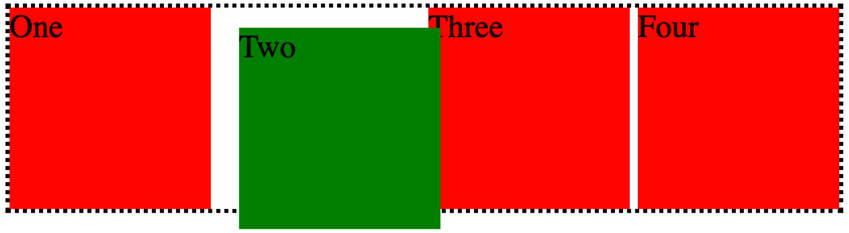
```
<div class="parent">
     <div class="box" class="box one">One</div>
     <div class="box" class="box two">Two</div>
     <div class="box" class="box three">Three</div>
     <div class="box" class="box four">Four</div>
</div>
```

```
.parent {
    position: relative;
}

.box {
    display: inline-block;
    background: red;
    width: 100px;
    height: 100px;
}

.two {
    background: green;
    left: 15px;
    top: 15px;
}
```

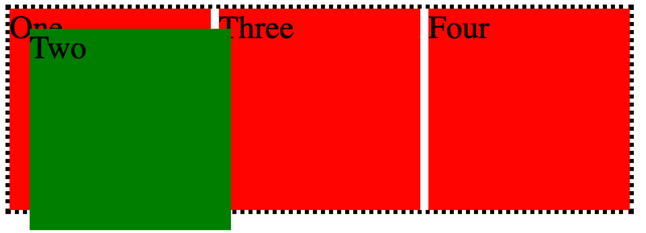
`position: static;`



`position: relative;`



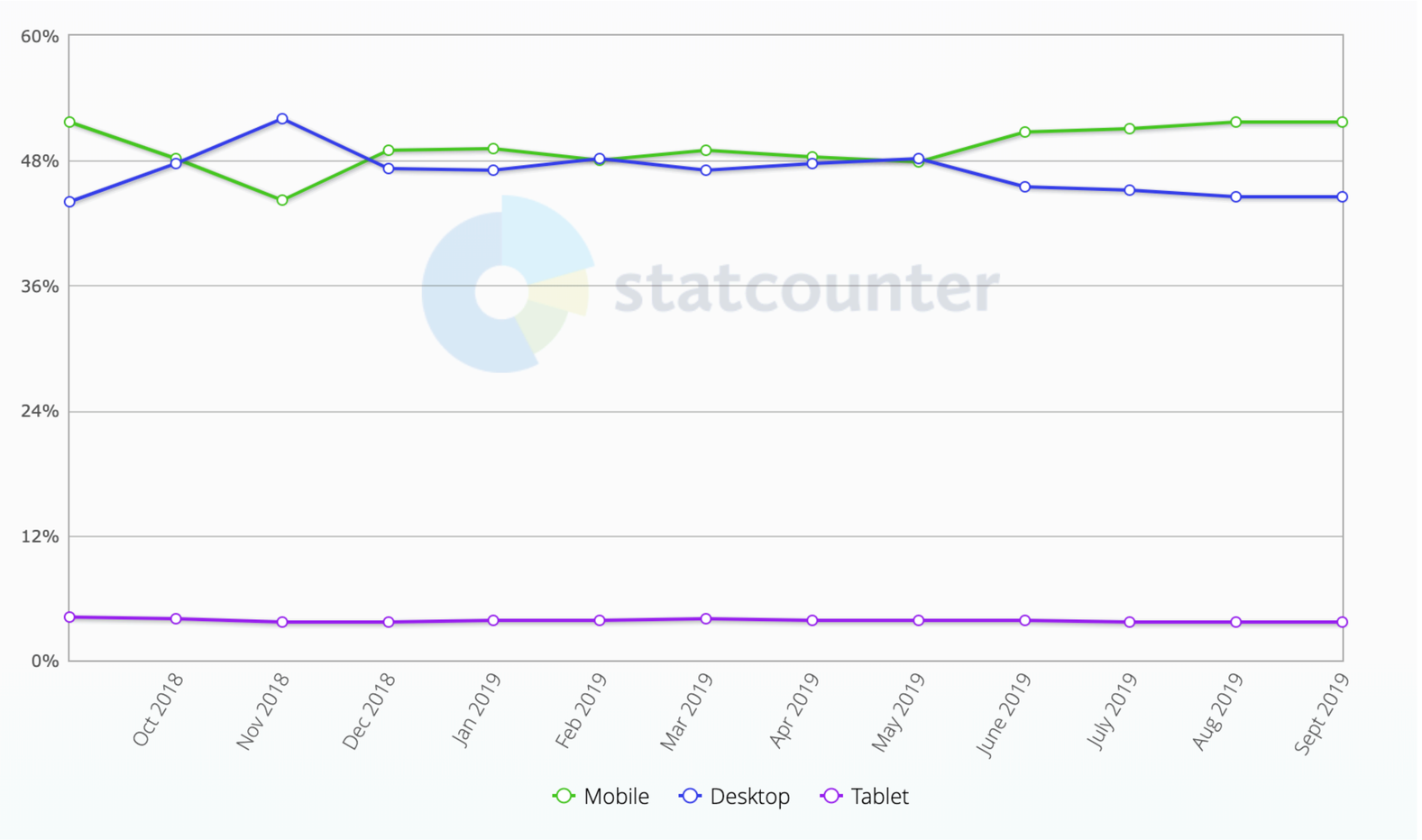
`position: absolute;`



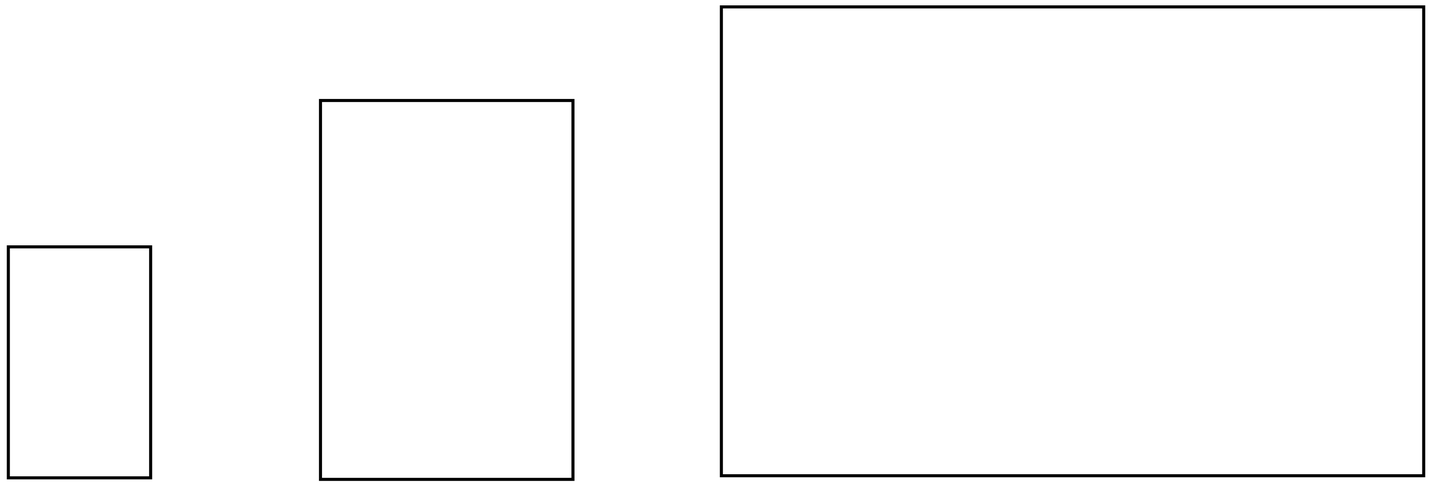
DEMO:
CREATING A WEBSITE FROM SCRATCH

RESPONSIVE WEB DESIGN

RESPONSIVE LAYOUTS



DESIGNING FOR DIFFERENT FORM FACTORS



Do we need to build different websites for each form factor?

CSS MEDIA QUERIES

Original CSS:

```
section {
    padding: 10px 10px;
}

img {
    display: inline-block;
}
```

If the screen size is <= 600px, also apply:

```
@media screen and (max-width: 600px) {

    section {
        padding: 20px 20px;
    }

    img {
        display: inline-block;
    }

}
```

CHANGING CSS THROUGH CODE

CHANGING CSS THROUGH CODE

- Use `body.querySelector(".{className}")` to select elements by their class name
- Use `element.style` to change CSS properties

```
let button = document.body.querySelector(".button");
let otherElement = document.body.querySelector(".other-element");

button.addEventListener("click", (e)=> {
     otherElement.style.background = "red";
});
```

CHANGING CSS THROUGH CODE

- Use `classList` to add/remove classes

```
let button = document.body.querySelector(".button");
let otherElement = document.body.querySelector(".other-element");

button.addEventListener("click", (e)=> {
    if (!otherElement.classList.contains("is-selected")) {
        otherElement.classList.add("is-selected");
    }
    else {
        otherElement.classList.remove("is-selected");
    }
});
```

FURTHER TOPICS

- Flexbox
  [https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox]
- Pseudo elements
  [https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements]
- CSS Units
  [https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units]
- CSS Preprocessors
  [https://sass-lang.com/]