Week 04 — Intro to CSS

Lukáš Grolig et al.

Outline

- First steps
- Selectors
- Cascading and inheritance
- Browser support
- The box model, flexbox vs grid
- Layouting
- Intro to BEM

What is CSS?

- Stands for **Cascading Style Sheets**
- Basic HTML is readable in a browser, but isn't aesthetically pleasing
- CSS can change how elements look in a browser using custom rules
- Clear separation of concerns: content in HTML, look and feel in CSS
- Has means for basic styling as well as advanced
  - Colors, fonts
  - Animations, 3D transforms
- Allows to style different viewport widths (mobile vs desktop) within a document

An example of CSS

```
.actionbutton {
    font-size: 1.15rem;
    color: white;
    padding: 1rem 1.5rem;
    text-align: center;
    background-color: #36393F;
}
```

Bridging HTML and CSS

- Inline CSS using `style` attributes
- Inline CSS using a `<style>` tag
- Linking an external CSS stylesheet
  - The only correct way

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" type="text/css" href="theme.css" />
    <link rel="stylesheet" type="text/css" href="theme-override.css" />
    <link rel="stylesheet" type="text/css" href="custom-styles.css" />
  </head>
  <body>

  </body>
</html>
```

CSS selectors

- **Used to specify which elements to target with a particular set of rules**
- A sort of a filter for elements
- "Markings" (classes, IDs) are added to HTML to allow for easier targeting with CSS
- These can be combined arbitrarily

1. Element, ID and class selectors
2. Attribute selectors
3. Pseudo-class selectors
4. Combinators

Element, ID and class selectors

- They target
  - whole elements
  - HTML classes (dot prefix)
  - HTML identifiers (should be unique, hash prefix)

```
h1 { }
```

```
.box { }
```

```
#unique { }
```

Attribute selectors

- They give you the option to target
  - the presence of an attribute, or
  - its value

```
a[title] { }
```

```
a[href="https://example.com"] { }
```

Pseudo-class selectors

- Can target pseudo-classes – these match certain states of an element
- For example `hover`, `visited`, or `focus`
- They also include means to target elements based on their ancestor relationship
- `first-child`, `last-child`, `only-child`, `nth-of-type`, `empty`, etc.

```
a:hover { }
```

Combinators

- Lining up selectors behind one another implies the latter being a descendant of the former
  - The so-called "descendant selector"
  - Represented with a space character
- **Direct** children can be targeted using the `>` combinator
- **Adjacent siblings** can be targeted using the `+` combinator
- **Any siblings** in general can be targeted using the `~` combinator

Combinator example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
  <article>
    <h1 class="header header-blue">
      Lorem Ipsum
    </h1>
    <p>
      Lorem Ipsum is simply dummy
      text of the printing and
      typesetting industry. Lorem
      Ipsum has been the industry's
      standard dummy text ever
      since the 1500s
    </p>

    <p>
      Secondary text
```

Combinator example

```
article > p + p {
    font-style: italic;
}
```

CSS Evaluation

```
body .container #hero-container .button a { }
<-------------------------------------
```

1. Things are evaluated from right to left
2. This means we get a list of all `<a>` s in document
3. In this list there is filter applied and only subset with parrent .button is returned
4. In this subset, we apply filter again and get those with parrent #hero-container
5. ...

The three horsemen of CSS

- Cascading
- Specificity
- Inheritance

Cascading

- The **cascading** in **Cascading Style Sheets** carries immense importance
- Stylesheets are applied in-order:
  - Default browser styles
  - External CSS files and `<style>` contents
  - Inline `style` attributes
- Styles in stylesheets are applied **top-to-bottom**
- Latter styles override former ones

Specificity

- Different rules may apply to the same elements
- Browser calculates a score for each rule -> higher score wins
- Order of specificity, ascending:
  - Element selectors (lowest score)
  - Class, pseudoclass, attribute selectors
  - ID selectors
  - Inline `style` attributes

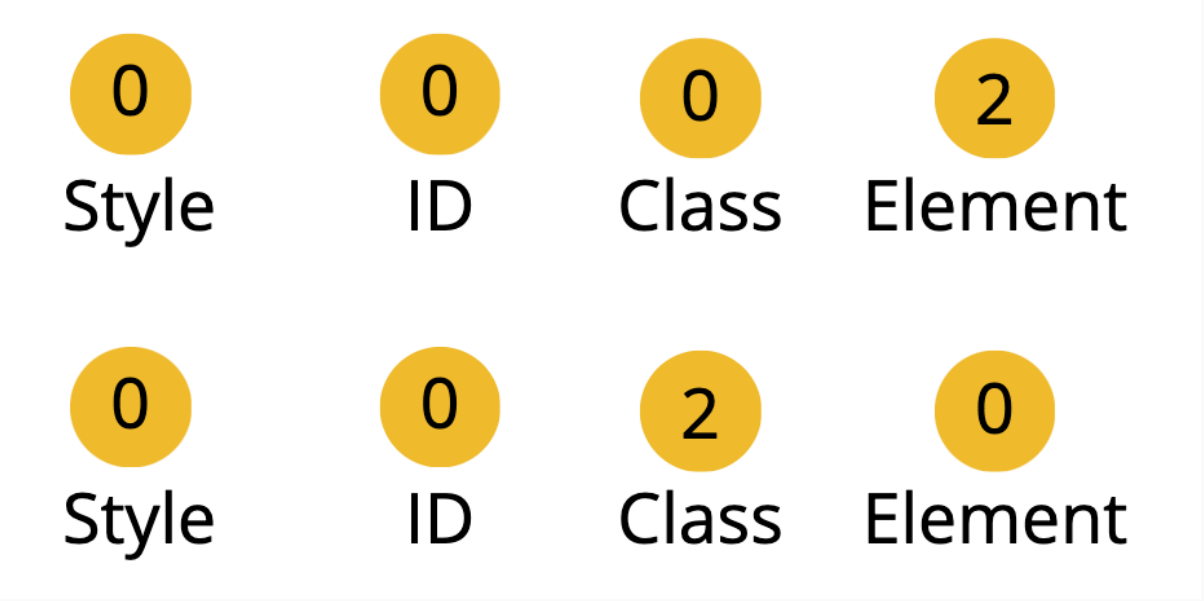
This is my heading.

```
.main-heading {
    color: red;
}

h1 {
    color: blue;
}
```

```
<h1 class="main-heading">This is my heading.</h1>
```

Selector specificity example

```
ul > li {
  color: white;
}

.list > .list-item {
  color: black;
}
```



= selector 2 wins

Inheritance

- Simply put, most CSS property values applied to an element are inherited by their descendants
- For example, `color`ing a document section will color all text paragraphs within
- **Not all properties are inherited**: `width` and `height`, among others

Browser support

- Writing CSS is nice, but browsers have to be able to parse our styles
- Different browsers support different rules (at different times)
- Developing for WebKit (or Google's fork Blink) usually works well
    - Firefox uses Gecko

Basic CSS concepts

- Two types of boxes make up the websites we see on the daily
  - **Block boxes**
  - **Inline boxes**

Inline boxes

- Are flush with previous content (inline)
- Do not respect `width` and `height` properties
- Padding and margin does not affect other elements
- Set with `display: inline` CSS property

Block boxes

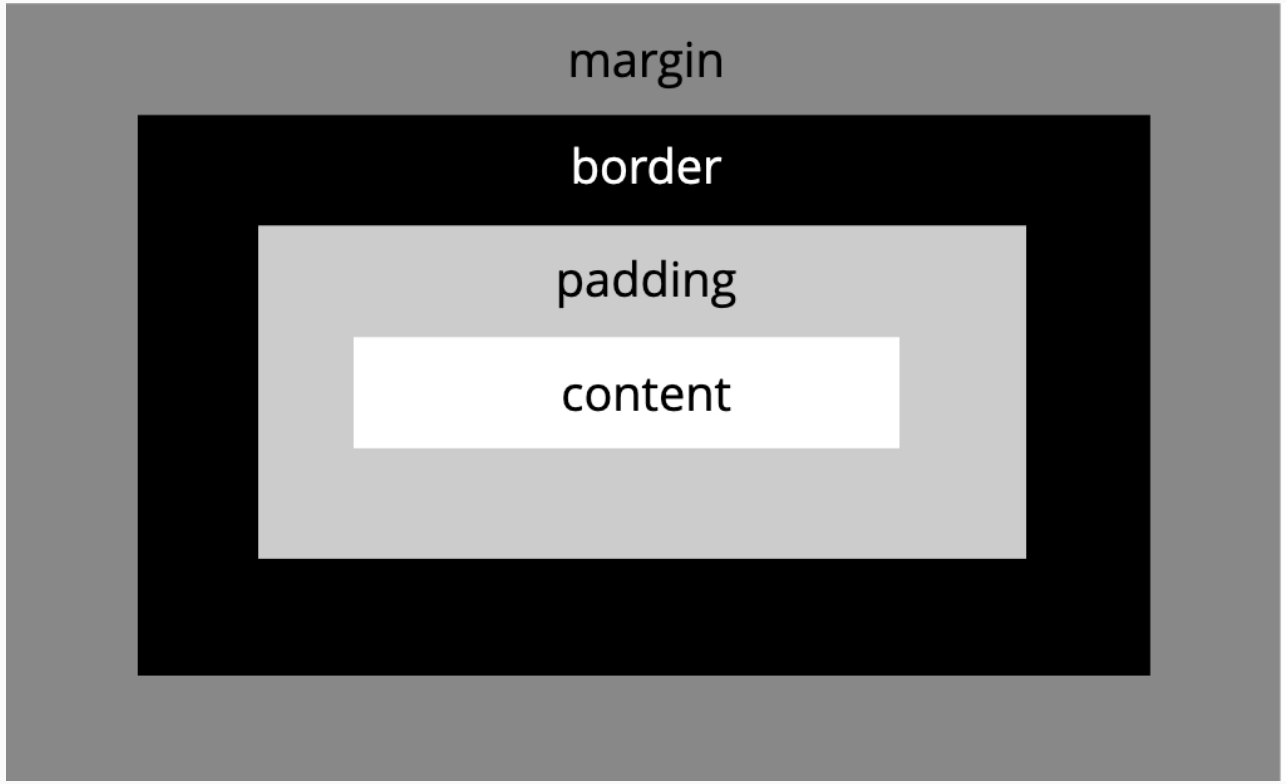
- Are placed on a new line from previous content
- Take up the full width of their container
- Respect their `width` and `height` properties
- Padding and margin properties push other elements away
- Set with `display: block` CSS property, implicit for `div` elements

Inline block boxes

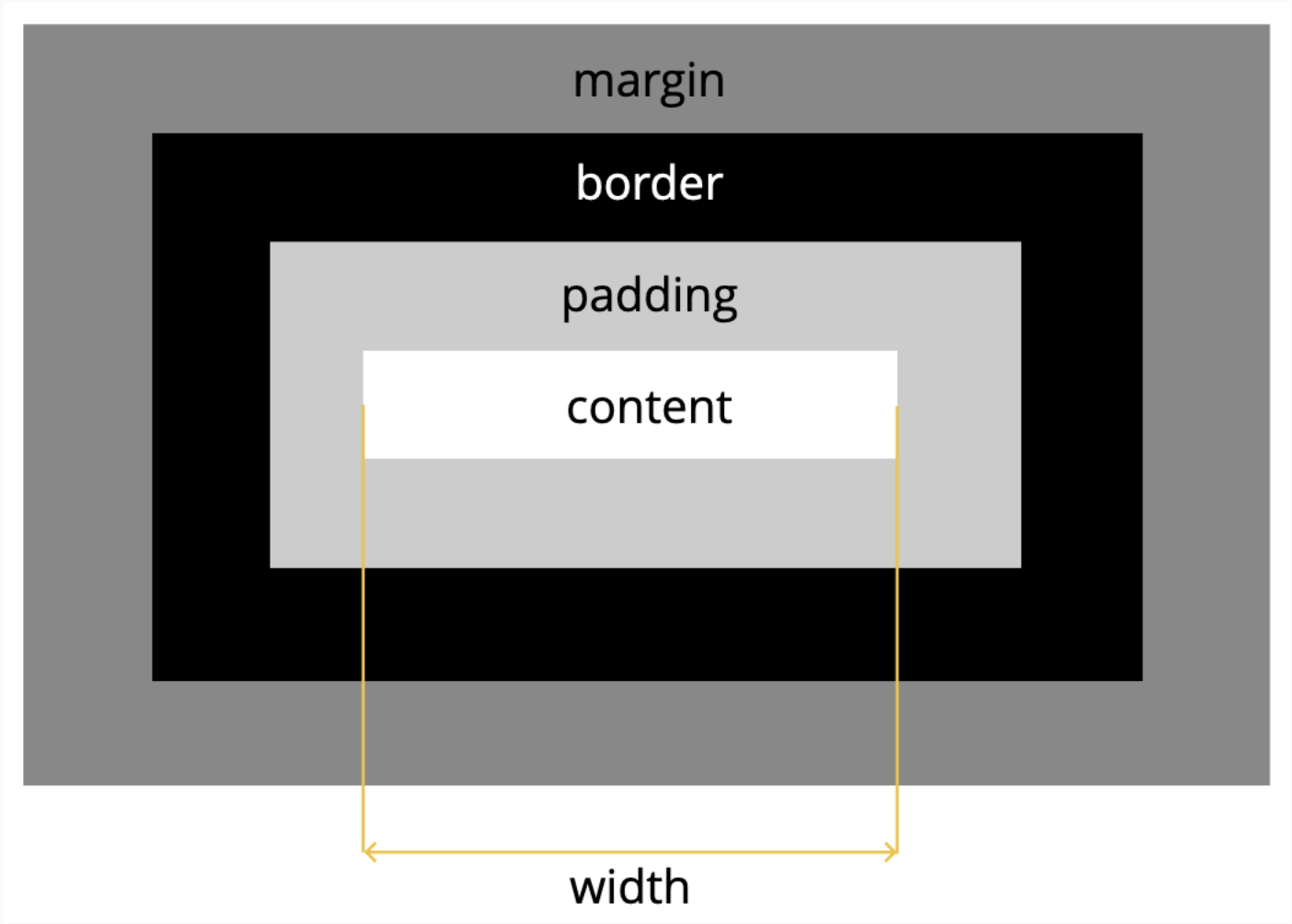
- Combination of inline boxes and block boxes
- Respect `width` and `height`, margin and padding works
- Are still flush with surrounding content
- Set with `display: inline-block` CSS property

The box model

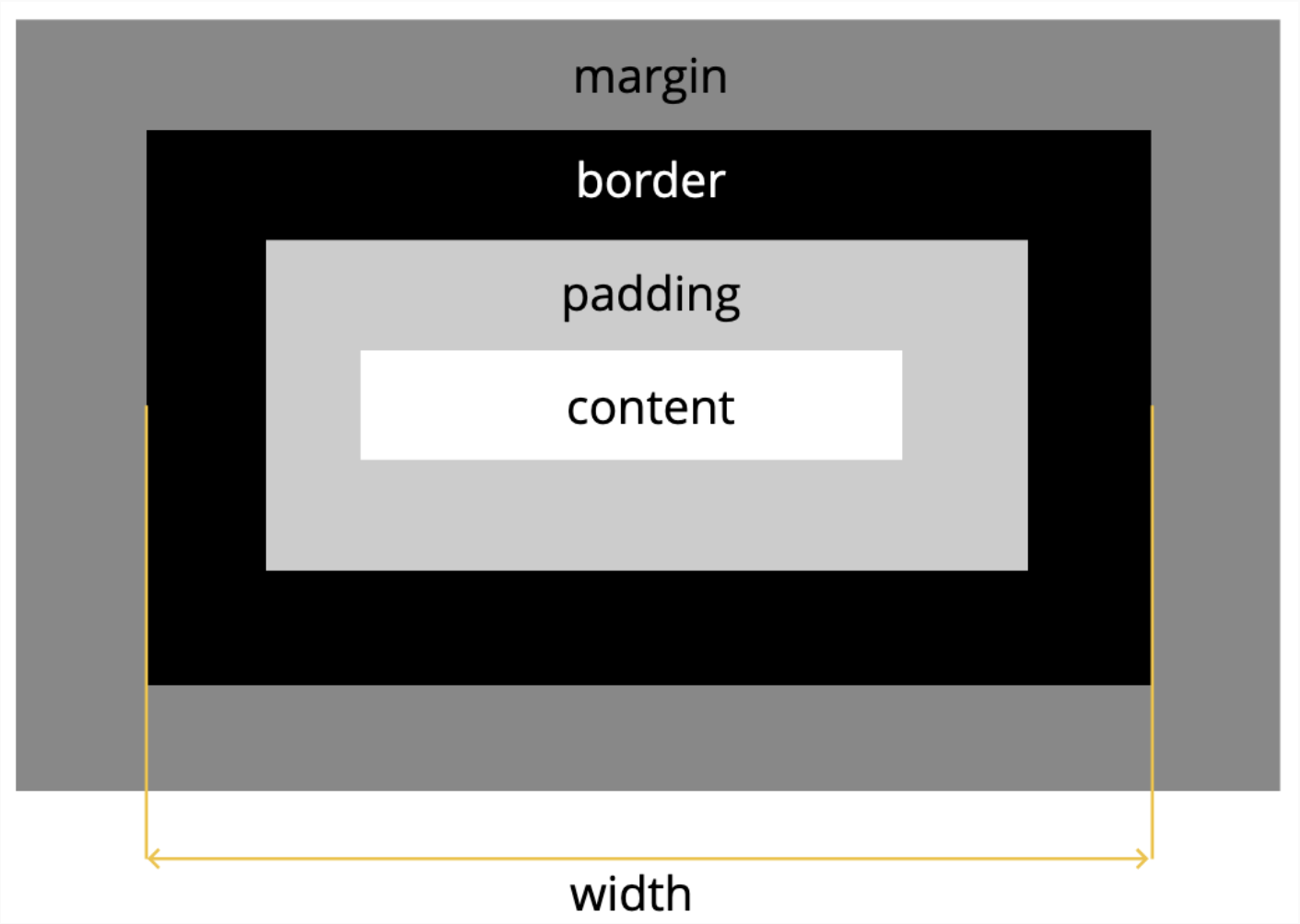
- Margin is invisible space around the box
  - Pushes elements away
  - Can be negative (element overlap)
- The border is drawn between margin and padding
  - Can be styled (width, style, color)
- Padding sits between the border and box contents
  - Pushes content away from border (inwards)
  - Can not be negative



Content box model



Border box model



Box model comparison

- Some elements default to content box, others to border box
- Border box is **superior** (easier to work with) and as such should be set globally

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

Setting margin/padding/border values: individually

```
.box {
  margin-top: 30px;
  margin-right: 30px;
  margin-bottom: 40px;
  margin-left: 4rem;
}
```

```
.box {
  margin: 30px 30px 40px 4rem;
}
```

Setting margin/padding/border values: in pairs

```
.box {
  margin: 30px 20px;
//          TOP  LEFT
//       BOTTOM RIGHT
}
```

Setting margin/padding/border values: altogether

```
.box {
  margin: 30px;
}
```

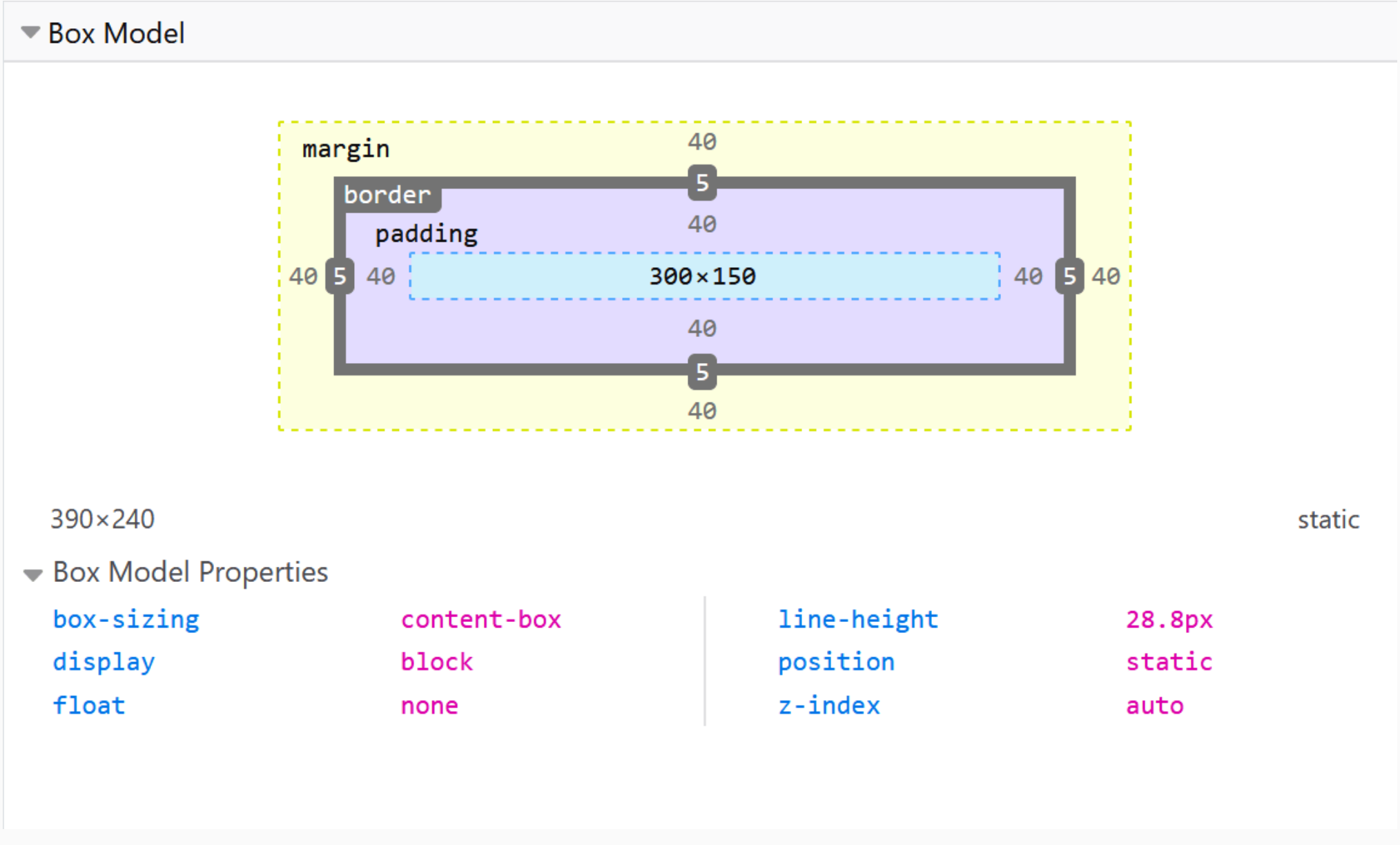
Hiding an element: discarding it

```
.box {
  display: none;
}
```

Hiding an element: keeping layout space reserved

```
.box {
  visibility: hidden;
}
```

Viewing the box model in browser devtools



Overflow

```
.no-overflow {
   overflow: hidden;
}

.visible-overflow {
  overflow: auto; // same as visible
}

.display-scrollbar {
  overflow: scroll;
}
```

Truncation

```
.truncate {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Responsive design

- Using a set of practices to allow having a single page/stylesheet
- When smartphones were introduced, companies would have to maintain a separate mobile version of their site
- Today's websites should be developed **mobile-first**
- Using browser dev tools to change viewport to simulate a phone screen
- Tablet and desktop styles come later
- But how do we distinguish which ones to use when?
- **Media queries**

Media queries

- **Media queries** were introduced in 2009
  - Allow for quicktesting of users viewport (and its size)
  - Screen size values are referred to as **breakpoints**

```
@media only screen and (min-width: 768px) {
    body {
        background-color: lightblue;
    }
}
```

Typical breakpoints

- Obviously not set in stone but fairly common among different device manufacturers

1. Extra small devices (phones, 600px and down)
2. Small devices (tablets in portrait mode, large phones, 600px and up)
3. Medium devices (landscape tablets, 768px and up)
4. Large devices (laptops or small desktops, 992px and up)
5. Extra large devices (large laptops and dekstops, 1200px and up)

Device orientation

```
@media only screen and (orientation: landscape) { /* or portrait */
  body {
    background-color: lightblue;
  }
}
```

Layouting with CSS

- Historically, using tables for content layout was the common way
- Sadly not very responsive and **semantically wrong**
- Using `float` properties works, not easy to get right
- New ways have emerged: **flexbox** and **grid**

Flexbox

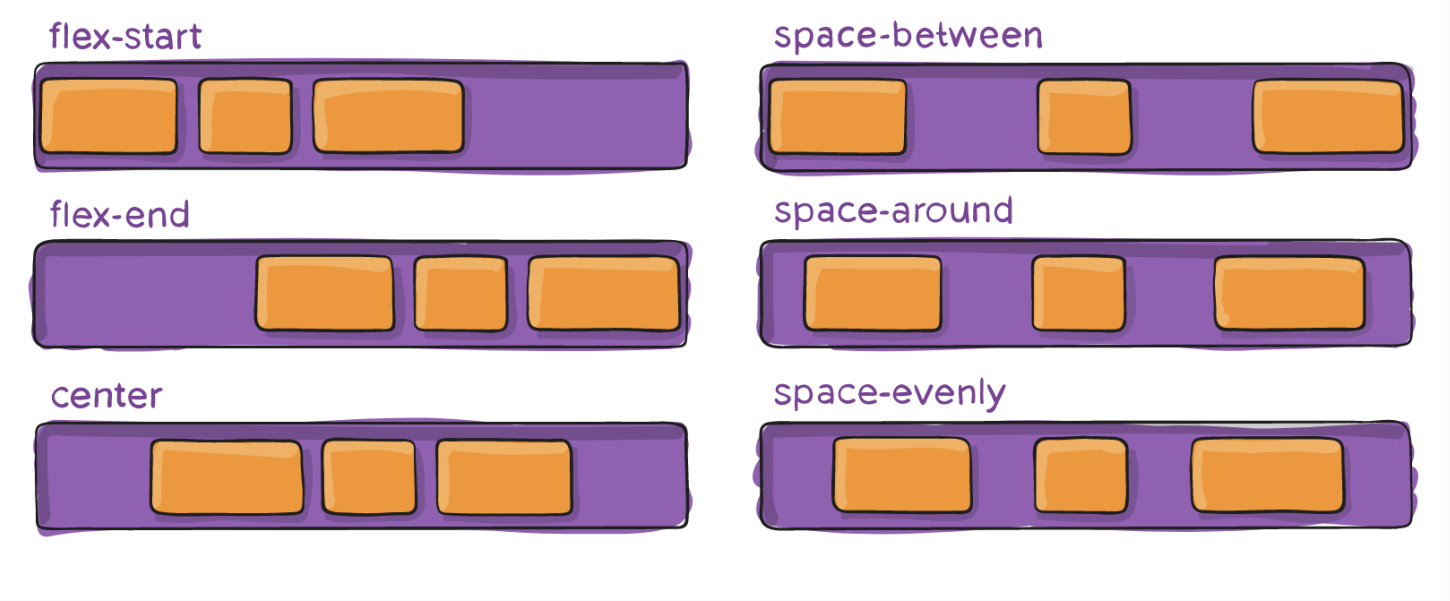
- A one-dimensional container for arranging items in rows or columns
- Items flex (expand) to fill additional space or shrink to fit into smaller spaces
- It is a reliable solution that works cross-browser

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
</div>
```

```
.flex-container {
  display: flex;
  flex-direction: row | column;
}
```

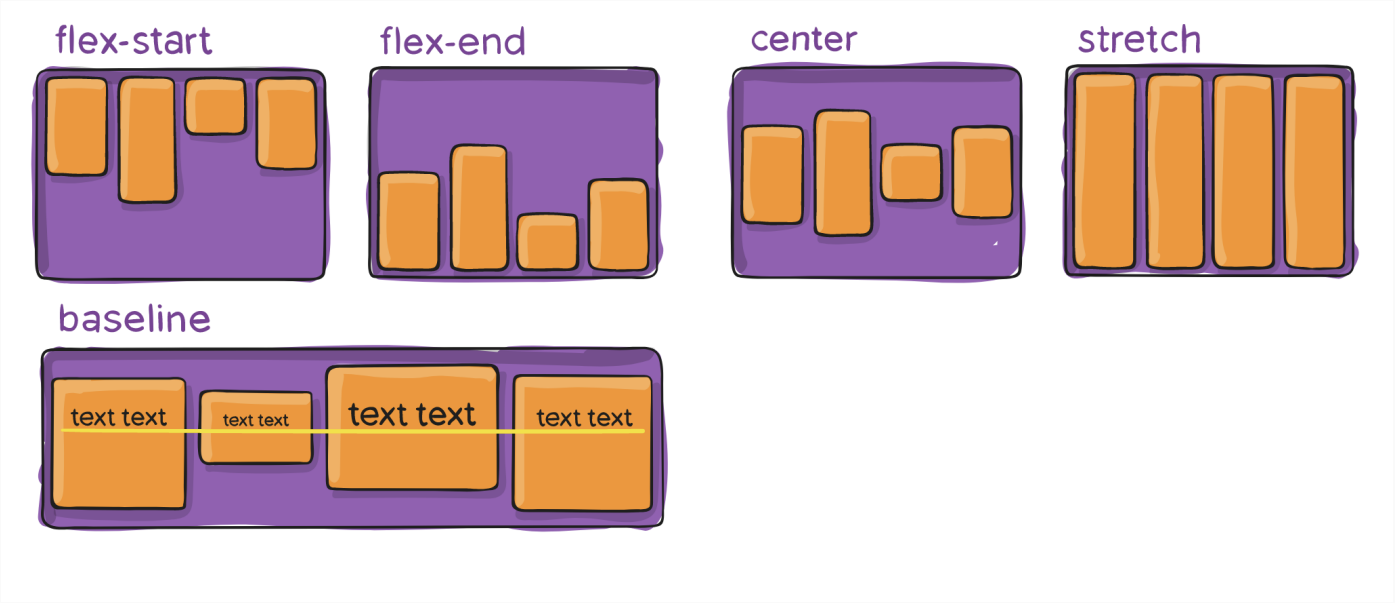
Aligning content along the main axis

- Using `justify-content`
- Examples for `flex-direction: row`:



Align content along the secondary axis

- Using `align-items`
- Examples for `flex-direction: row`:

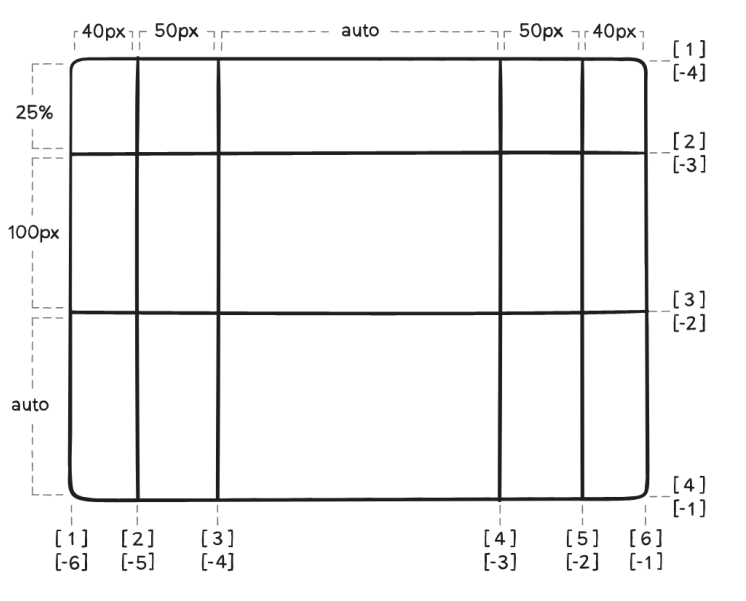


Assigning properties to individual flex items

- Can control whether an item takes up residual space (`flex-grow`)
- Can specify how much an item shrinks if it were to overflow (`flex-shrink`)
- Can specify minimum size of item value (it cannot shrink below it) with `flex-basis`
- These can and **should** be shorthanded (`flex: 0 1 300px`)
- Items can also be reordered using the `order` property

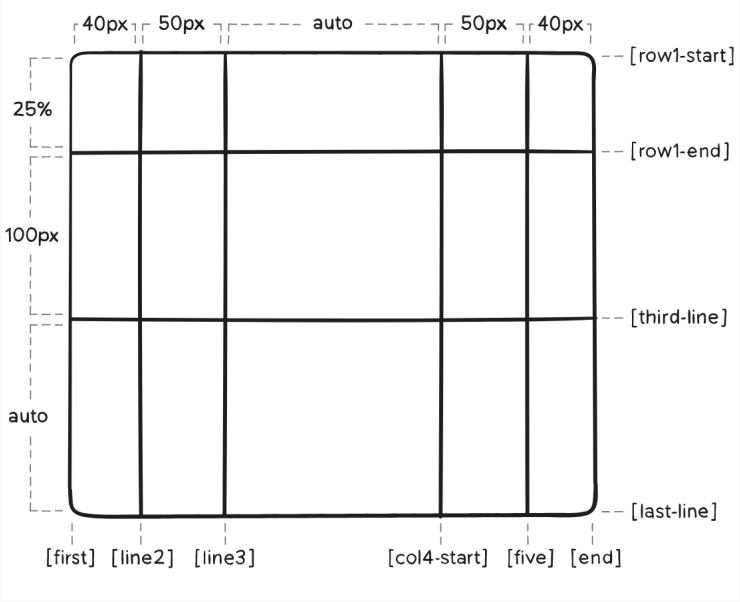
CSS grid: part 1

- Useful for two-dimensional layout where flexbox does not suffice
- Basically does what tables used to be used for - but much better - and more




```
.container {
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```

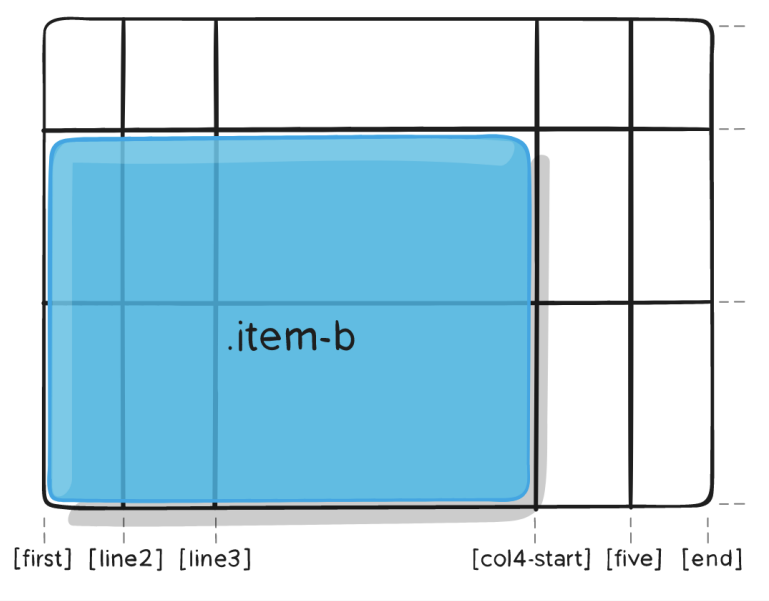
CSS grid: part 2




```
.container {
  grid-template-columns:
    [first] 40px [line2] 50px [line3]
    auto [col4-start] 50px [five] 40px [end];
  grid-template-rows:
    [row1-start] 25% [row1-end]
    100px [third-line] auto [last-line];
}
```

CSS grid: part 3

- Note: items can overlap each other, stacking order can be controlled with `z-index`



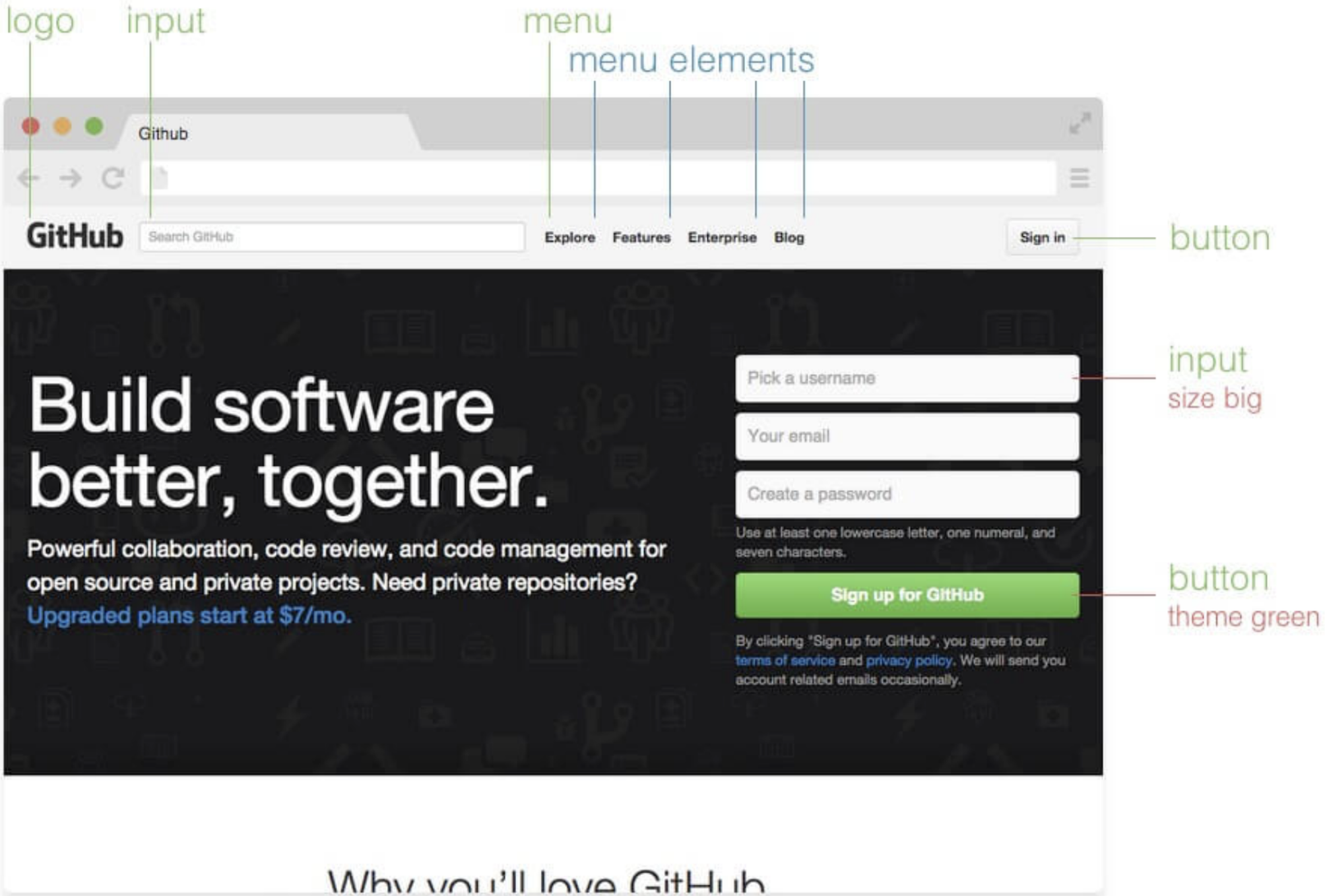

```
.item-b {
  grid-column-start: 1;
  grid-column-end: col4-start;
  grid-row-start: 2;
  grid-row-end: span 2;
}
```

Practices on organizing CSS

- Keeping it consistent -> **using a methodology**
- Avoiding overly-specific selectors
- Commenting CSS (labelling sections like typography)
- Breaking each property on a new line
- Using selector lists to avoid duplicite styles

What is BEM?

- A way of defining CSS classes to use with HTML elements
- BEM methodology helps to think with components – important to understand
- **Block** - a standalone entity that is meaningful on its own
- **Element** - a part of a block with no meaning on its own, semantically tied to block
- **Modifier** - a flag on block/element used to modify appearance or behavior


logo
input
menu
menu elements
Github
GitHub
Search GitHub
Explore
Features
Enterprise
Blog
Sign in
button
Build software better, together.
Powerful collaboration, code review, and code management for open source and private projects. Need private repositories?
Upgraded plans start at $7/mo.
Pick a username
Your email
Create a password
Use at least one lowercase letter, one numeral, and seven characters.
Sign up for GitHub
By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.
input
size big
button
theme green
Why you'll love GitHub

Adhering to BEM

```
<button class="button">Normal button</button>
<button class="button button--state-success">Success button</button>
<button class="button button--state-danger">Danger button</button>
```

```
.button {
        display: inline-block;
        border-radius: 3px;
        padding: 7px 12px;
        border: 1px solid #D5D5D5;
        background-image: linear-gradient(#EEE, #DDD);
}
.button--state-success {
        color: #FFF;
        background: #569E3D linear-gradient(#79D858, #569E3D) repeat-x;
        border-color: #4A993E;
}
.button--state-danger {
        color: #900;
}
```

A word on CSS preprocessors

- Sass, Less, PostCSS (and others) augment regular CSS
- Add variables, mixins, computed values
- Need to compile files to regular CSS before using in production
- Over time, variables and other features were introduced to CSS itself
- The need to use preprocessors has decreased

SCSS

```scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

SCSS nesting

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

SCSS mixins

```scss
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}

.box {
  @include transform(rotate(30deg));
}
```

SCSS inheritance

```scss
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}
```

A word on CSS frameworks

- They make a developer's job easier
- Many to choose from, different approaches
- Emphasis on mobile-first, long-term popular: Bootstrap
- Based on flexbox: Bulma
- Based on grid: Foundation (used by IS MU)
- Utility-first and modern: **Tailwind**

Tailwind.css example

```
<button
    type="button"
    class="inline-flex items-center px-3 py-2 border
    border-transparent text-sm leading-4 font-medium
    rounded-md shadow-sm text-white bg-indigo-600
    hover:bg-indigo-700 focus:outline-none focus:ring-2
    focus:ring-offset-2 focus:ring-indigo-500">
    Button text
</button>
```

Resources

- https://slides.com/lukasgrolig/pb138-introduction-to-css
- https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/
- https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes
- https://www.smashingmagazine.com/2019/02/css-browser-support/
- https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/
- https://www.browserstack.com/guide/top-css-frameworks