Week 05: CSS

Agenda

- CSS properties
- Selectors recap
- Flexbox vs grid
- BEM revisited
- Hands on

Let's start!

Main CSS properties: divided by purpose

- Background: solid colors, gradient, images, positioning, repetition
- Box model: width and height, padding and margin, border color, style, and width
- Positioning: left, right, top, and bottom, z-index
- Typography: color, font-size, -family, -weight, line-height, text-align, -transform
- Transitions
- Animations
- **Flex parents: flex-direction, -wrap, (-flow), align-items, justify-content**
- **Flex children: flex-basis, -grow, -shrink, order**
- **Grid parents: grid-template-rows, -template-columns, -template-areas, -column/row-gap, ...**
- **Grid children: -column-start and -end (-column shorthand), ditto for column, ...**

Property reference

- CSSreference.io
- MDN

Let's take a look together.

Selector recap

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 { }
```

Selector lists

- The CSS selector list is denoted by a comma (`,`) and selects all matching nodes

```
a:hover {
  color: red;
}

#navbar {
  color: red;
}
```

```
a:hover, #navbar {
  color: red;
}
```

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

Demo: CSS selector game

https://flukeout.github.io/

Can you reach level 17?

Flexbox and grid

- Which one to use? It depends
- Flexbox is useful for one-dimensional layouts
  - Can change orientation based on viewport width
  - Order of children can change as well
  - Easy to distribute and align space between elements
- Grid is better suited for two-dimensional layouts
  - Essentially behaves like a table

Understanding flex properties

- **For parent**: flex-direction, flex-wrap, align-items, justify-content, align-content
- **For items**: align-self, flex-grow, flex-shrink, flex, *order*
- Interactive examples

Helpful tip: knowing how to use flex order may come in handy in the iteration.

Understanding grid: part 1

Get started by defining a container:

```
.container {
  display: grid | inline-grid;
}
```

Understanding grid: part 2

Lay out the layout:

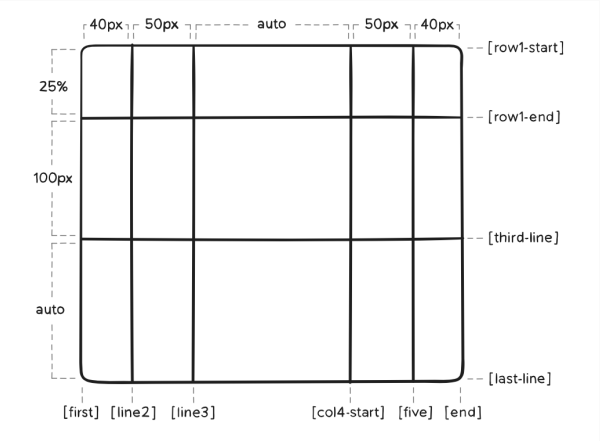
- `grid-template-columns` or `grid-template-rows` takes
  - Track-size (length, percentage, free space portion `fr`, or `auto`)
  - Arbitrary name to label this section (optional)

```
.container {
  grid-template-columns: 1fr 50px 1fr 1fr;
  // four 50px columns
}
```

Understanding grid: part 2.5

Lines between rows and columns can be explicitly named (square bracket notation):

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



Tip: repeating parts in column/row definition can be streamlined with `repeat(n, ...)`

Understanding grid: part 3

- Define where slots start/end by referring to line numbers or names
- Slots can span across multiple tracks (`span <number>`) or until they hit a specific line (`span <name>`)

```
.item {
  grid-column-start: <number> | <name> | span <number> | span <name> | auto;
  grid-column-end: <number> | <name> | span <number> | span <name> | auto;
  grid-row-start: <number> | <name> | span <number> | span <name> | auto;
  grid-row-end: <number> | <name> | span <number> | span <name> | auto;
}
```

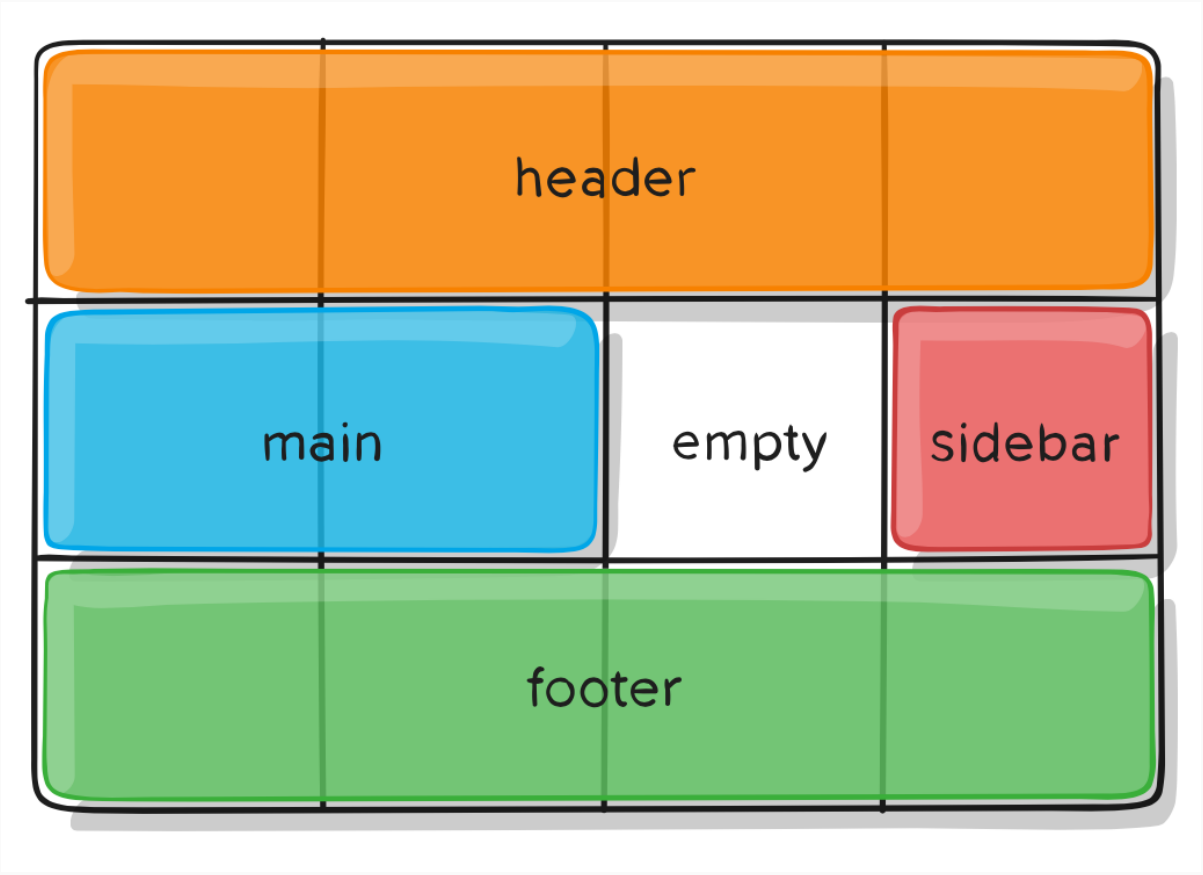
- `grid-column: a b` = shorthand for `grid-column-start: a` and `grid-column-end: b`
- ditto for rows

Understanding grid: part 4

- Assign "grid areas" to items
- Define layout on grid element
- Dots signify empty cells

```
.item-a {grid-area: header}
.item-b {grid-area: main}
.item-c {grid-area: sidebar}
.item-d {grid-area: footer}

.container {
  display: grid;
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
```


header
main
empty
sidebar
footer

Congratulations on understanding CSS Grid!

For more thorough explanations, refer to the Complete Grid Guide.

Let's talk BEM

Block

- An independent page **component** that can and should be reused
- Its **name describes its purpose** (button), not its appearance (not red, not big)
- Blocks can be nested in each other

```
<!-- `search-form` block -->
<form class="search-form">
    <!-- `input` element in the `search-form` block -->
    <input class="search-form__input">

    <!-- `button` element in the `search-form` block -->
    <button class="search-form__button">Search</button>
</form>
```

Element

- A semantical part of a block, **unable** to stand on its own
- Separated from the block name with a double underscore (`block-name__element-name`)
- Can be nested, but only the outermost block is projected into element name (so **never** `block__elem1__elem2`)

When to use a block and when an element?

- If a section of code might be reused and it doesn't depend on other page components being implemented => **block**
- If a section of code can't be used separately without the parent entity => **element**

Modifier

- Defines the appearance, state or behavior of its parent (block or element)
- Separated with a double hyphen (`block-name--modifier`)
- Can never be used alone (is semantically tied)

```
<!-- The `search-form` block has the `focused` Boolean modifier -->
<form class="search-form search-form--focused">
    <input class="search-form__input">

    <!-- The `button` element has the `disabled` Boolean modifier -->
    <button class="search-form__button search-form__button--disabled">Search</button>
</form>
```

Questions?

Hands on: Iteration 04

You can find the assignment in GitLab issues.

Let's take a look together.

An important iteration tip

Some (but very few) HTML elements may appear more than once. It is nearly impossible to achieve the desired result without some repetition – but use it sparsely.

Before you start:

- Please check whether your tutor has already accepted your MR
- If they have, make sure you have merged your solution from the previous week

*Note: if your tutor has **not** seen your MR, it's completely ok. You do **not** need to have the previous iteration merged to be able to work on a new one - **iterations are independent**. However, if you **do** have an accepted MR that still has not been merged, make sure to merge it first.*