{POWER.CODERS}

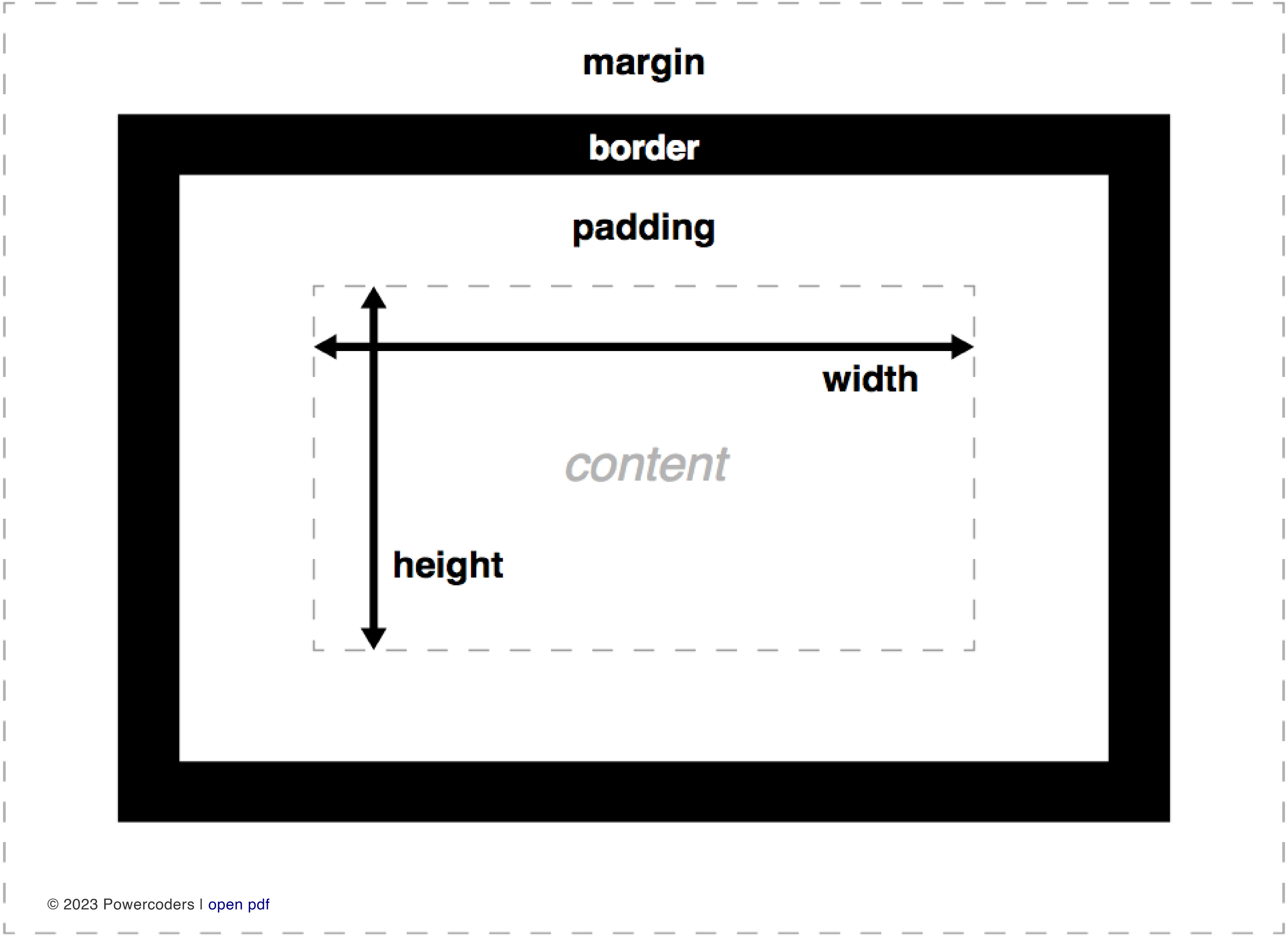
CSS Layout techniques

Agenda

Today we will learn about different CSS layout techniques

- Box model
- Document flow
- CSS Flexbox
- CSS Grid
- CSS Multi columns

Box model


margin
border
padding
width
content
height

Default box rules

- `width` is the width of the content area
- `height` is the height of the content area
- `background` properties apply to padding as well as content
- `padding` adds to the total size of the box
- Like padding, `border` adds to the total size of the box

Default box rules

- `width` is the width of the content area
- `height` is the height of the content area
- `background` properties apply to padding as well as content
- `padding` adds to the total size of the box
- Like padding, `border` adds to the total size of the box

Calculating the total height and width of elements can be difficult. Especially for responsive websites.

`box-sizing: content-box`

Default box rules apply

The total height of an element is the sum of

- content height
- plus padding-top and -bottom
- plus border-top and -bottom

The total width of an element is the sum of

- content width
- plus padding-left and -right
- plus border-left and -right

`box-sizing: border-box`

Best practice to use always this value

`box-sizing: border-box`

Best practice to use always this value

- The **total height** of an element is identical to the **content height** including **padding** and **border**.

`box-sizing: border-box`

Best practice to use always this value

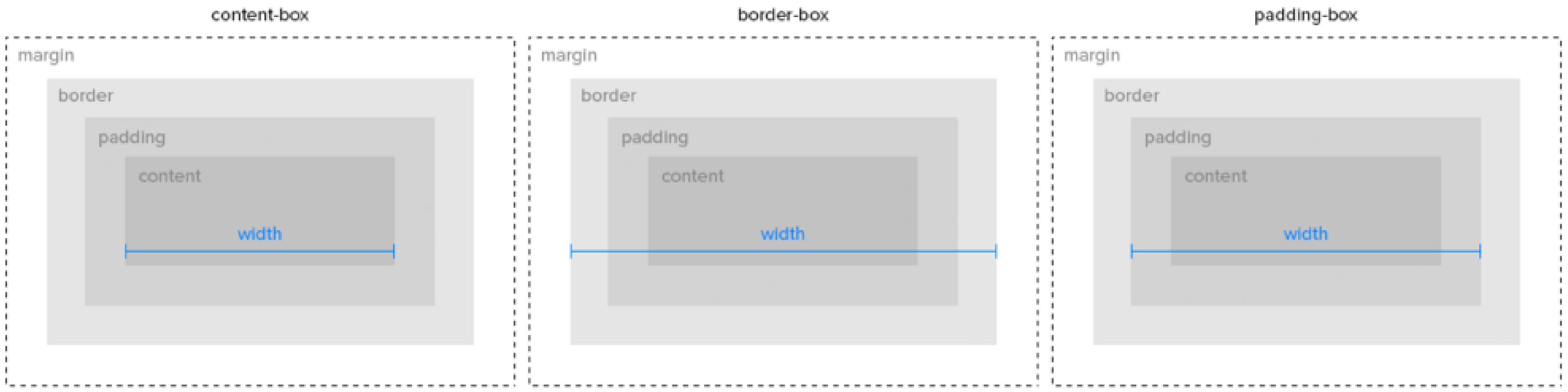
- The **total height** of an element is identical to the **content height** including **padding** and **border**.
- The **total width** of an element is identical to the **content width** including **padding** and **border**.

`box-sizing: border-box`

Best practice to use always this value

- The **total height** of an element is identical to the **content height** including **padding** and **border**.
- The **total width** of an element is identical to the **content width** including **padding** and **border**.

Set border-box **once** on html selector and inherit for all other elements.


content-box
margin
border
padding
content
width
border-box
margin
border
padding
content
width
padding-box
margin
border
padding
content
width

Best practice

```css
html {
  box-sizing: border-box;
}

*, *:before, *:after {
  box-sizing: inherit;
}
```

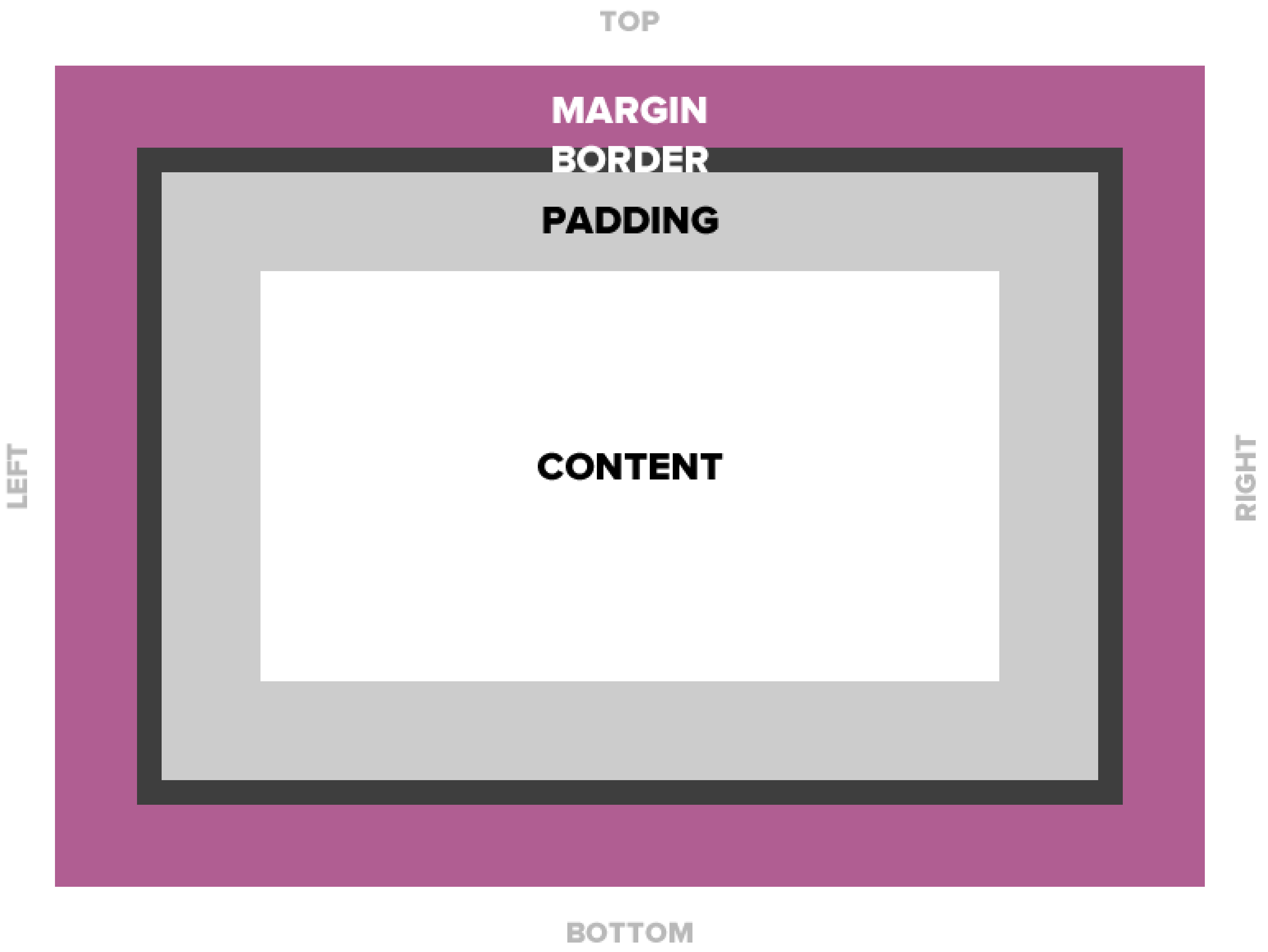
Why not?

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

Why not?

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

Some 3rd party plugins / components might require content-box model. With the best practice solution you ensure that those plugins will still be styled correctly.


TOP
MARGIN
BORDER
PADDING
CONTENT
LEFT
RIGHT
BOTTOM

`margin`

Four values: 10px on top, 5px on right, 3px on bottom, 5px on left

```
margin: 10px 5px 3px 5px; /* clockwise order: top right bottom left */
```

Two values: 10px top and bottom, 15px left and right

```
margin: 10px 15px; /* top/bottom right/left */
```

One value: 15px on all side

```
margin: 15px;
```

One side: 10px only on top

```
margin-top: 10px;
```

`margin: auto`

If a margin is set to auto on a box that has a given width, it will take up as much space as possible.

Centered

```css
margin: auto;
width: 50%;
```

Flush right

```css
margin-left: auto;
margin-right: 0.5rem;
width: 50%;
```

MARGIN COLLAPSE

Collapsing margins happen when two **vertical margins** come in contact with one another. If one margin is **greater** than the other, then that margin overrides the other, leaving **one margin**.
This happens in these 3 cases:

- Adjacent sibling elements: sharing the same parent
- Parent and first / last child
- Empty blocks

An example

```html
<body>
  <h1>Title</h1>
  <p>Paragraph</p>
</body>
```

```css
h1 {
  margin-bottom: 25px;
}

p {
  margin-top: 50px;
}
```

An Example

```html
<body>
  <h1>Title</h1>
  <p>Paragraph</p>
</body>
```

```css
h1 {
  margin-bottom: 25px;
}

p {
  margin-top: 50px;
}
```

You would expect 75px, but instead you get **50px** margin between the `h1` and the `p`.
It's like the bigger margin ate the smaller one: **bigger margin = total vertical margin**

Negative margin

```
h1 {
  margin-bottom: -25px;
}

p {
  margin-top: 50px;
}
```

Negative margin

```
h1 {
  margin-bottom: -25px;
}

p {
  margin-top: 50px;
}
```

50px + (-25px) = 25px

Negative margin

```css
h1 {
  margin-bottom: -25px;
}

p {
  margin-top: 50px;
}
```

50px + (-25px) = 25px

If one margin is negative, the negative margin is subtracted from the positive margin, reducing the total vertical margin.
If both margins are negative, the bigger negative margin eats the smaller one: **bigger negative margin = total negative vertical margin**

`padding`

Four values: 10px on top, 5px on right, 3px on bottom, 5px on left

```
padding: 10px 5px 3px 5px; /* clockwise order: top right bottom left */
```

Two values: 10px top and bottom, 15px left and right

```
padding: 10px 15px; /* top/bottom right/left */
```

One value: 15px on all sides

```
padding: 15px;
```

One side: 10px only on top

```
padding-top: 10px;
```

* `background` properties apply to padding as well as content.

`border`

Borders are specified as "thickness, style, color."
You can specify each property separately, or all three together.

```
border: 1px solid #ff0000;
```

```
border-top: 4px dotted #000000;
```

```
border-width: 10px;
border-style: dashed;
border-color: #666666;
```

Document flow

It's all about the flow

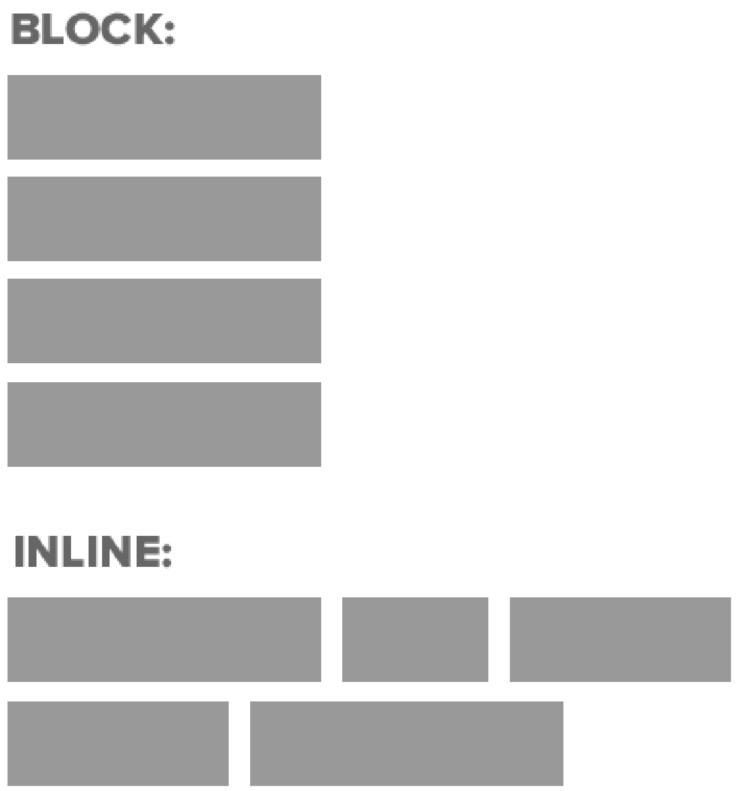
Document flow is the arrangement of page elements, as defined by CSS positioning statements, and the order of HTML elements.

Regarding the order of the HTML elements, their definition as **inline** or **block-level** element defines the space they take up in the document.

Document flow = how each element takes up space and how other elements position themselves accordingly.

Flow of HTML elements

Flow of HTML elements



`display`

defines how an element is displayed. You can turn block-level elements to inline and vice verse.

```css
a {
  display: block; /* block-level element */
}
```

```css
h1 {
  display: inline; /* inline element, will break at end of line */
}
```

```css
li {
  display: inline-block; /* appears inline, does not break across lines */
}
```

```css
#footer {
  display: none; /* hidden */
}
```

`display: inline-block`

Block-level elements are stacked underneath each other in one **column**.

`display: inline-block`

Block-level elements are stacked underneath each other in one **column**.

Changing their `display`-property to `inline-block` results in a **row** of these elements.

`display: inline-block`

Block-level elements are stacked underneath each other in one **column**.

Changing their `display`-property to `inline-block` results in a **row** of these elements.

Is the maximum width of the parent (wrapping) container reached, the elements will automatically wrap into a new line.

Tipps when using inline-block

`inline-block` elements need to have a **width** defined.

Tipps when using inline-block

`inline-block` elements need to have a **width** defined.

Use `vertical-align` to make sure that the elements are aligned properly in one row.

Tipps when using inline-block

`inline-block` elements need to have a **width** defined.

Use `vertical-align` to make sure that the elements are aligned properly in one row.

When two elements with `display: inline-block` are sitting next to each other, whitespace between them becomes a space character. **Remove the whitespace.**

CSS POSITIONING

`position: static`

- Initial value to all elements

`position: static`

- Initial value to all elements
- Static positioned element stay **in-flow**

`position: relative`

- Relative positioned element stay in-flow, but **interact** with out-of-flow elements.

`position: relative`

- Relative positioned element stay in-flow, but **interact** with out-of-flow elements.
- It acts as the container for out-of-flow children. The children respect the box boundaries of the relatively positioned element.

`position: relative`

- Relative positioned element stay in-flow, but **interact** with out-of-flow elements.
- It acts as the container for out-of-flow children. The children respect the box boundaries of the relatively positioned element.
- The content of a relative positioned box can be shifted out-of-flow by offset properties: top, right, bottom, left.

`position: absolute`

- Absolute positioned element is removed from the flow entirely = **out-of-flow** element.

`position: absolute`

- Absolute positioned element is removed from the flow entirely = **out-of-flow** element.
- Their position is **assigned to the first parent** element, which has a non-static position (relative, absolute, fixed or sticky).

`position: absolute`

- Absolute positioned element is removed from the flow entirely = **out-of-flow** element.
- Their position is **assigned to the first parent** element, which has a non-static position (relative, absolute, fixed or sticky).
- The offset properties (top, right, bottom, left) are based on the **top left corner** of that parent.

`position: absolute`

- Absolute positioned element is removed from the flow entirely = **out-of-flow** element.
- Their position is **assigned to the first parent** element, which has a non-static position (relative, absolute, fixed or sticky).
- The offset properties (top, right, bottom, left) are based on the **top left corner** of that parent.
- Each absolute positioned elements get its **own layer**. You can stack the layer with the CSS property `z-index`.

`position: fixed`

- Fixed positioned element is removed from the flow entirely = out-of-flow element.

`position: fixed`

- Fixed positioned element is removed from the flow entirely = out-of-flow element.
- It is assigned a position to the viewport (browser window) and creates a new layer.

`position: fixed`

- Fixed positioned element is removed from the flow entirely = out-of-flow element.
- It is assigned a position to the viewport (browser window) and creates a new layer.
- The offset properties (top, right, bottom, left) are based on the top left corner of the viewport.

`position: sticky`

- Mix between relative positioned element and fixed positioned element.

`position: sticky`

- Mix between relative positioned element and fixed positioned element.
- It acts like relative positioned until it is scrolled beyond a specific offset, then it turns to fixed position.

`position: sticky`

- Mix between relative positioned element and fixed positioned element.
- It acts like relative positioned until it is scrolled beyond a specific offset, then it turns to fixed position.
- Can I use position sticky?

Tricky sticky

- If any parent/ancestor of the sticky element has any of the following overflow properties set, position: sticky won't work: `hidden`, `auto`, `scroll`

Tricky sticky

- If any parent/ancestor of the sticky element has any of the following overflow properties set, position: sticky won't work: `hidden`, `auto`, `scroll`
- If the parent element has no `height` set then the sticky element won't have any area to stick to when scrolling. This happens because the sticky element is meant to stick/scroll within the height of a container.

Tricky sticky

- If any parent/ancestor of the sticky element has any of the following overflow properties set, position: sticky won't work: `hidden`, `auto`, `scroll`
- If the parent element has no `height` set then the sticky element won't have any area to stick to when scrolling. This happens because the sticky element is meant to stick/scroll within the height of a container.

More info and issues in this tutorial

`float`

- A float is a box that is shifted to the left or right on the current line.

`float`

- A float is a box that is shifted to the left or right on the current line.
- Content flows around the shifted box, down the right side of a left-floated box and vice verse.

`float`

- A float is a box that is shifted to the left or right on the current line.
- Content flows around the shifted box, down the right side of a left-floated box and vice verse.
- Floated elements are **out-of-flow**. The parent container loses its content height and width.

`float`

- A float is a box that is shifted to the left or right on the current line.
- Content flows around the shifted box, down the right side of a left-floated box and vice verse.
- Floated elements are **out-of-flow**. The parent container loses its content height and width.

Floated elements are still often used for typical website layouts.
DO NOT USE THEM.

Clearfix

Is used to solve the parent height problem of floated elements

```
.clearfix:before,
<!-- .clearfix:after {
 content:"";
 display:table;
} -->

.clearfix:after {
 clear:both;
}
<!--
.clearfix {
 *zoom:1; -->
}
```

Tipps when using float

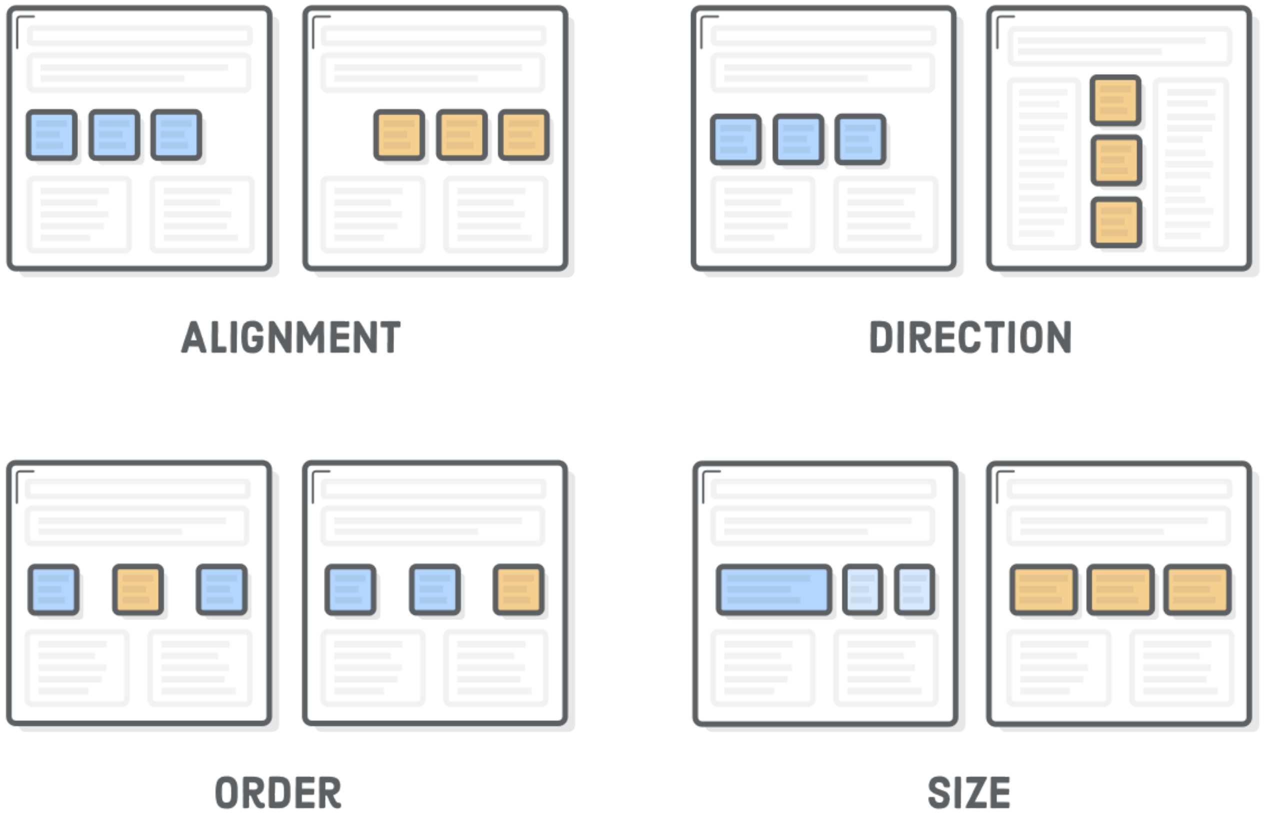
Use the `.clearfix` snippet to ensure the parent element takes up enough space in the document flow.

Tipps when using float

Use the `.clearfix` snippet to ensure the parent element takes up enough space in the document flow.

Use `clear` if you want following elements to move below the floated element.

CSS Flexbox



`display: flex`

- Use `display: flex;` to create a flex container.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.
- Use `flex-direction` if you need columns instead of rows.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.
- Use `flex-direction` if you need columns instead of rows.
- Use the `row-reverse` or `column-reverse` values to flip item order.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.
- Use `flex-direction` if you need columns instead of rows.
- Use the `row-reverse` or `column-reverse` values to flip item order.
- Use `order` to customize the order of individual elements.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.
- Use `flex-direction` if you need columns instead of rows.
- Use the `row-reverse` or `column-reverse` values to flip item order.
- Use `order` to customize the order of individual elements.
- Use `align-self` to vertically align individual items.

`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.
- Use `flex-direction` if you need columns instead of rows.
- Use the `row-reverse` or `column-reverse` values to flip item order.
- Use `order` to customize the order of individual elements.
- Use `align-self` to vertically align individual items.
- Use `flex` to create flexible boxes that can stretch and shrink.

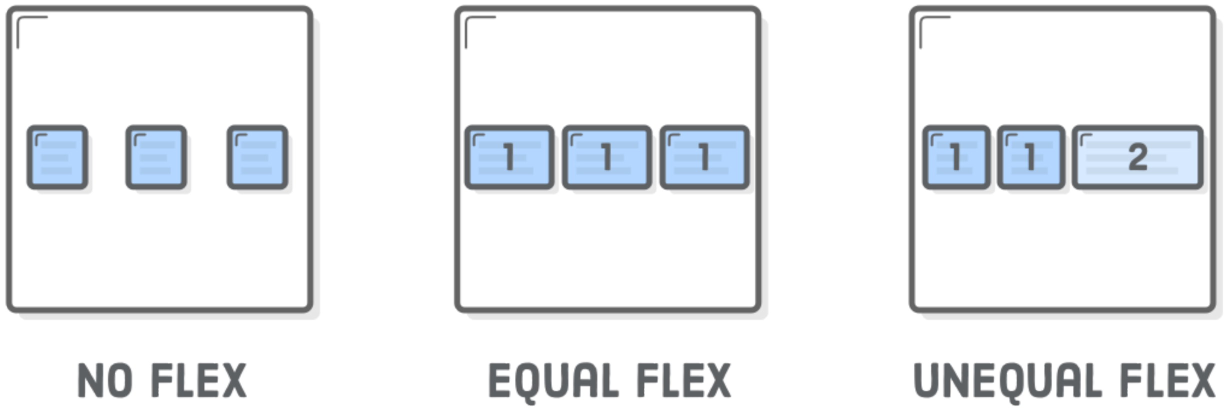
`display: flex`

- Use `display: flex;` to create a flex container.
- Use `justify-content` to define the horizontal alignment of items.
- Use `align-items` to define the vertical alignment of items.
- Use `flex-direction` if you need columns instead of rows.
- Use the `row-reverse` or `column-reverse` values to flip item order.
- Use `order` to customize the order of individual elements.
- Use `align-self` to vertically align individual items.
- Use `flex` to create flexible boxes that can stretch and shrink.

Flexbox is an easy way to create **responsive websites** as scalability is built-in.

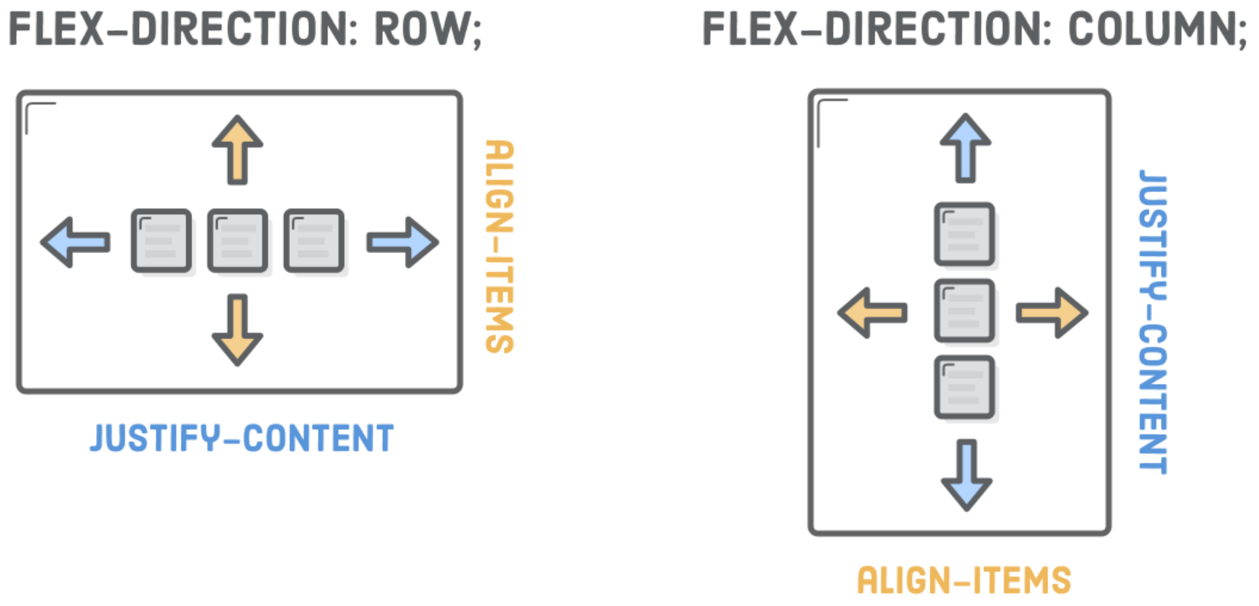
Flexible containers

With the property `flex` on the items you have the first step for a responive website.



TIPPS WHEN USING FLEX

Depending on the `flex-direction` the properties **justify-content** and **align-items** switch meaning.



CSS Grid

It is the latest CSS layout technique.
All major web browsers support it though, **so use it**.

- Use `display: grid;` to create a grid container.

CSS Grid

It is the latest CSS layout technique.
All major web browsers support it though, **so use it**.

- Use `display: grid;` to create a grid container.
- Items are placed in rows by default and span the full width of the grid container.

CSS Grid

It is the latest CSS layout technique.
All major web browsers support it though, **so use it**.

- Use `display: grid;` to create a grid container.
- Items are placed in rows by default and span the full width of the grid container.
- Use `grid-template-rows` to define the number (and height) of rows.

CSS Grid

It is the latest CSS layout technique.
All major web browsers support it though, **so use it**.

- Use `display: grid;` to create a grid container.
- Items are placed in rows by default and span the full width of the grid container.
- Use `grid-template-rows` to define the number (and height) of rows.
- Use `grid-template-columns` to define the number (and width) of columns.

CSS GRID

It is the latest CSS layout technique.
All major web browsers support it though, **so use it**.

- Use `display: grid;` to create a grid container.
- Items are placed in rows by default and span the full width of the grid container.
- Use `grid-template-rows` to define the number (and height) of rows.
- Use `grid-template-columns` to define the number (and width) of columns.
- Use `grid-gap` or `grid-row-gap` / `grid-column-gap` to define the gutter between grid items.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.
- You can combine `fr` units with other units like px, em or %.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.
- You can combine `fr` units with other units like px, em or %.
- Use `grid-row-start` and `grid-row-end` accordingly.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.
- You can combine `fr` units with other units like px, em or %.
- Use `grid-row-start` and `grid-row-end` accordingly.
- Use `grid-template-areas` to define names for your grid, e.g. header, content, sidebar and footer.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.
- You can combine `fr` units with other units like px, em or %.
- Use `grid-row-start` and `grid-row-end` accordingly.
- Use `grid-template-areas` to define names for your grid, e.g. header, content, sidebar and footer.
- Place the items in the grid by using `grid-column-start` and `grid-column-end`.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.
- You can combine `fr` units with other units like px, em or %.
- Use `grid-row-start` and `grid-row-end` accordingly.
- Use `grid-template-areas` to define names for your grid, e.g. header, content, sidebar and footer.
- Place the items in the grid by using `grid-column-start` and `grid-column-end`.
- Auto-place items by using `grid-auto-rows`, `grid-auto-columns` and `grid-auto-flow`.

`display: grid`

- Use `fr` unit to create flexible grid tracks. It represents a fraction of the available space in the grid container.
- You can combine `fr` units with other units like px, em or %.
- Use `grid-row-start` and `grid-row-end` accordingly.
- Use `grid-template-areas` to define names for your grid, e.g. header, content, sidebar and footer.
- Place the items in the grid by using `grid-column-start` and `grid-column-end`.
- Auto-place items by using `grid-auto-rows`, `grid-auto-columns` and `grid-auto-flow`.
- Use `justify-items` and `align-items` to align the items inside your grid.

Grid vs Flex: When to use which?

- Grid puts layout first: structure and predictability
- Flex puts content first: more flexibility

It is not one or the other. Mix them, use them both.

CSS Multi columns

Newspaper-style columns, often used as fallback for `flex` and `grid` layouts or for masonary-like layouts (like pinterest).

- Use `column-count` to define the number of columns.

CSS Multi columns

Newspaper-style columns, often used as fallback for `flex` and `grid` layouts or for masonary-like layouts (like pinterest).

- Use `column-count` to define the number of columns.
- Use `column-width` to define the width of each column.

CSS Multi columns

Newspaper-style columns, often used as fallback for `flex` and `grid` layouts or for masonary-like layouts (like pinterest).

- Use `column-count` to define the number of columns.
- Use `column-width` to define the width of each column.
- Use `column-gap` to define the gutter/margin between the columns.

CSS Multi columns

Newspaper-style columns, often used as fallback for `flex` and `grid` layouts or for masonary-like layouts (like pinterest).

- Use `column-count` to define the number of columns.
- Use `column-width` to define the width of each column.
- Use `column-gap` to define the gutter/margin between the columns.
- Use `column-rule` to display a vertical line between the columns.

CSS Multi columns

Newspaper-style columns, often used as fallback for `flex` and `grid` layouts or for masonary-like layouts (like pinterest).

- Use `column-count` to define the number of columns.
- Use `column-width` to define the width of each column.
- Use `column-gap` to define the gutter/margin between the columns.
- Use `column-rule` to display a vertical line between the columns.
- Use `column-span` on child elements you want to span all columns.

CSS Multi columns

Newspaper-style columns, often used as fallback for `flex` and `grid` layouts or for masonary-like layouts (like pinterest).

- Use `column-count` to define the number of columns.
- Use `column-width` to define the width of each column.
- Use `column-gap` to define the gutter/margin between the columns.
- Use `column-rule` to display a vertical line between the columns.
- Use `column-span` on child elements you want to span all columns.
- Use `break-inside` and similar properties on children to control content breaks.

REFERENCE SHEETS

- CSS intro
- CSS positioning

Online ressources for CSS Grid

- Complete Guide to Grid on CSS Tricks
- Grid by Example by Rachel Andrew
- The CSS Workshop by Jen Simmons
- Spring Into CSS Grid by Joni Trythall
- Grid cheat sheet

ONLINE RESSOURCES

- Youtube Channel: Layout Land by Jen Simmons
- Flexbox - a friendly tutorial for modern CSS Layouts
- CSS multiple column layout by Rachel Andrew
- Responsive CSS columns
- Visual guide for flexbox, grid and positioning
- Ten modern layouts in one line of CSS
- Sketching with CSS Cheatsheet

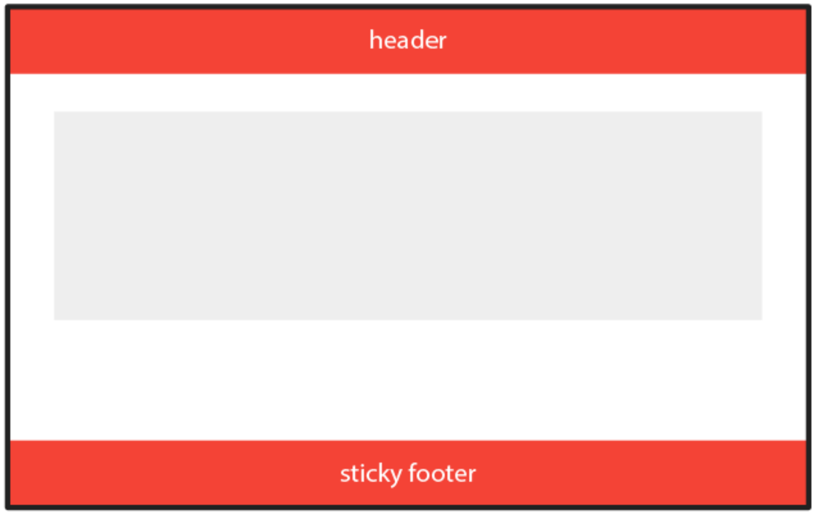
Exercises

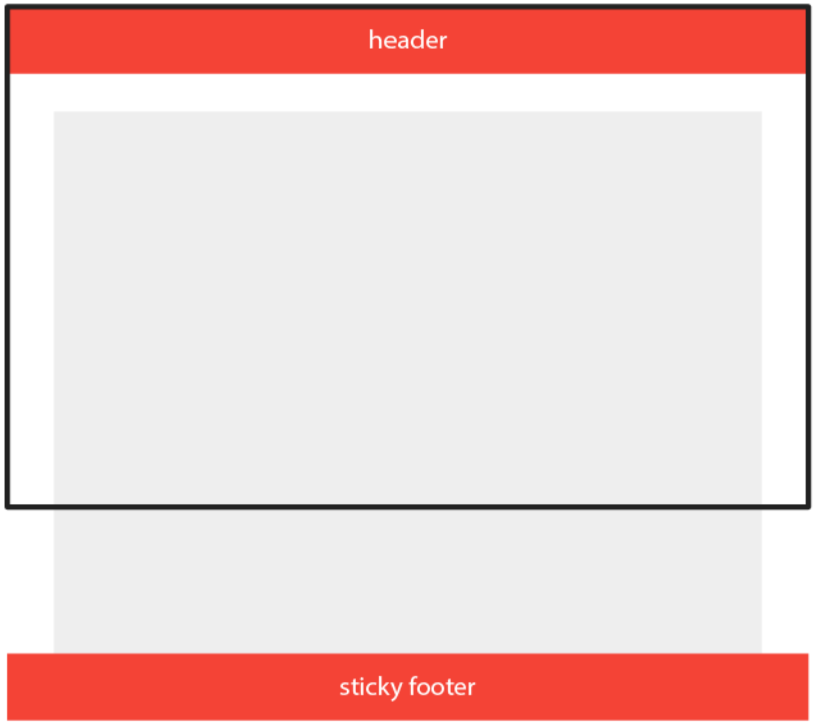
1. Sticky footer

Definition: A sticky footer pattern is one where the footer of your page "sticks" to the bottom of the viewport in cases where the content is shorter than the viewport height.

Assignment: Create a basic website layout with a sticky footer at the bottom. Once with CSS grid, once with flex.

Don't forget to push it to GitHub


header
sticky footer


header
sticky footer

2. Responsive layout in grid and flex

3. Play games

- CSS grid garden
- Flexbox Froggy

Work on Sample Shop

Can you already add the first styles to our sample shop?

On repl.it

Try to implement the new elements you learned this morning.