JONAS.IO
SCHMEDTMANN

Subscribe here

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

Follow me here

@JONASSCHMEDTMAN

SLIDES FOR THEORY LECTURES

(DON'T SKIP THEM, THEY ARE SUPER IMPORTANT 🤓)

📖 TABLE OF CONTENTS: THEORY LECTURES (CLICK THE TITLES)

SECTION 01 — WELCOME AND FIRST STEPS

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

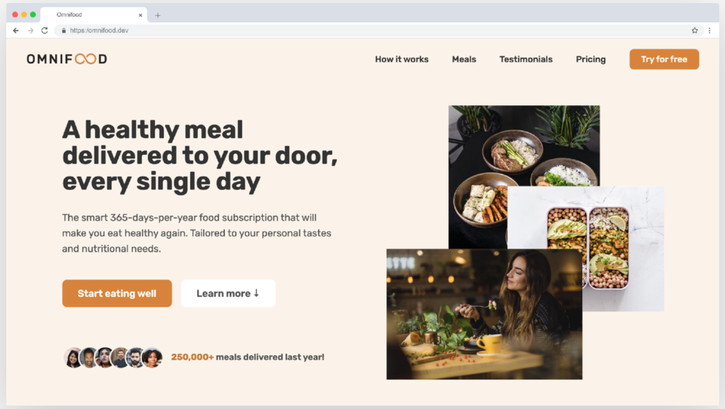
SECTION

WELCOME AND FIRST STEPS

LECTURE

A HIGH-LEVEL OVERVIEW OF WEB DEVELOPMENT

FRONT-END VS. BACK-END DEVELOPMENT



STATIC WEBSITE

Website where files are simply sent to browser as they are

index.html | style.css | style.css

script.js | image.jpg | image.jpg

HTML CSS JS

The 3 languages that **browsers** understand

The process of writing

FRONT-END DEVELOPMENT

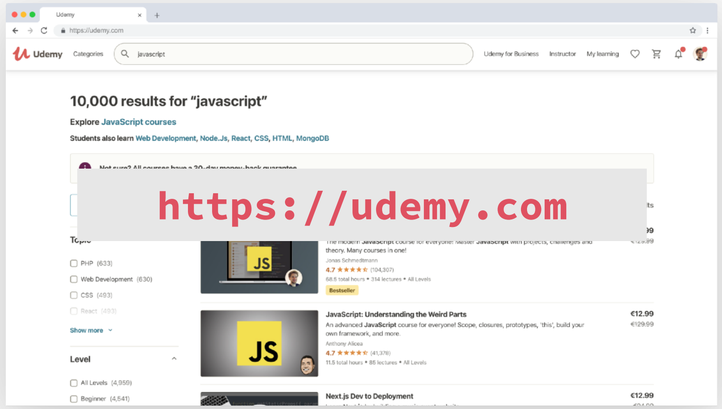
BROWSER

REQUEST

RESPONSE

WEB SERVER

FRONT-END VS. BACK-END DEVELOPMENT



DYNAMIC WEBSITE

Website files are "assembled" on the server

`index.html` `style.css` `script.js` `image.jpg`

HTML CSS JS

The 3 languages that **browsers** understand

BROWSER

REQUEST

RESPONSE

WEB SERVER

APP

DB

node.js php

etc...

Back-end languages that run on **servers**

The process of writing

FRONT-END DEVELOPMENT

BACK-END DEVELOPMENT

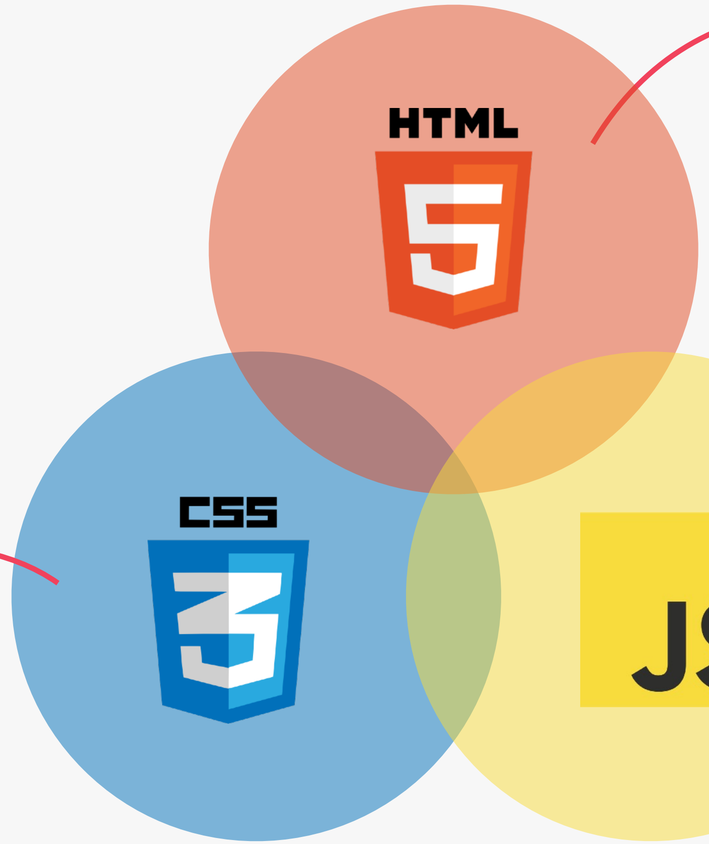
THE 3 **LANGUAGES** OF THE FRONT-END

CONTENT

NOUNS

`<p></p>`

means “***paragraph***”



ADJECTIVES

`p {color: red}`

means “the paragraph text is ***red***”

VERBS

`p.hide();`

means “***hide*** the paragraph”

PRESENTATION

PROGRAMMING LANGUAGE: DYNAMIC EFFECTS AND WEB APPLICATIONS

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

WELCOME AND FIRST STEPS

LECTURE

WATCH BEFORE YOU START!

SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

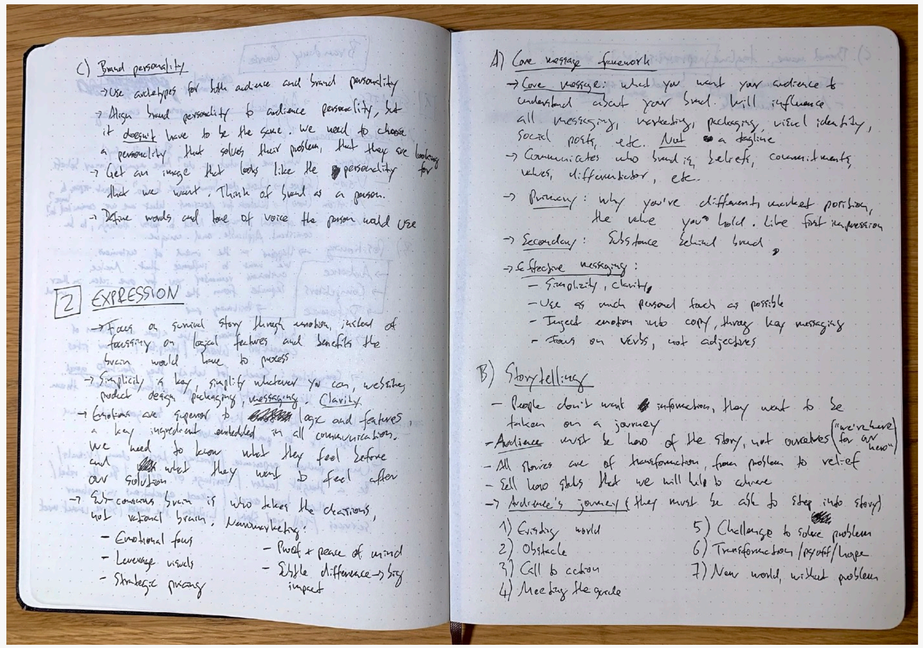
😱 **If this is your first time ever writing code, please don't get overwhelmed.** It's 100% normal that **you will not understand everything** at the beginning. ***Just don't think "I guess coding is not for me"!***

SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

👩‍💻 **You need to code along with me!** You will learn **ZERO** HTML and CSS skills by just sitting and watching me code. You really have to write code **YOURSELF**!

SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

📝 **If you want the course material to stick, take notes.** Notes on code syntax, notes on theory concepts, notes on everything!



Totally non-coding... Try to understand a single word 😂

SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

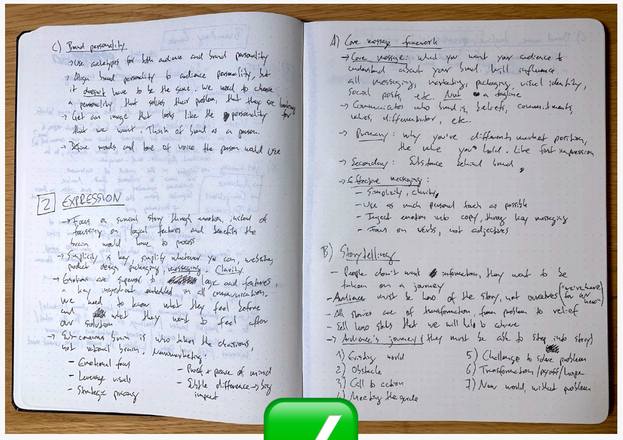
🤓 **Try all the coding challenges!** Try to do your best, but if you get stuck for too long, watch the solution. **Don't beat yourself up if you can't figure it out!** Just rewatch the lectures that were covered in the challenge, try to understand them better, and move on.

Watch for this sign!

PAUSE THE VIDEO
FOR CHALLENGE

SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

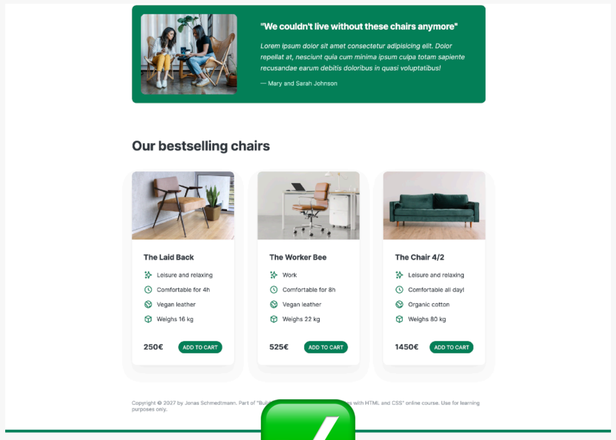
⏳ **Before moving on from a section, make sure that you understand exactly what was covered.** Take a break, review the code we wrote, review your notes, review the projects we built, and maybe even write some code yourself.



✅

```
.chair-details li:not(:last-child) {
  /* margin-bottom: 24px; */
  margin-bottom: 16px;
}

.chair-icon {
  width: 24px;
  height: 24px;
  stroke: #087f5b;
}

.chair-price {
  display: flex;
  justify-content: space-between;

  align-items: center;
  font-size: 20px;
}

footer {
```

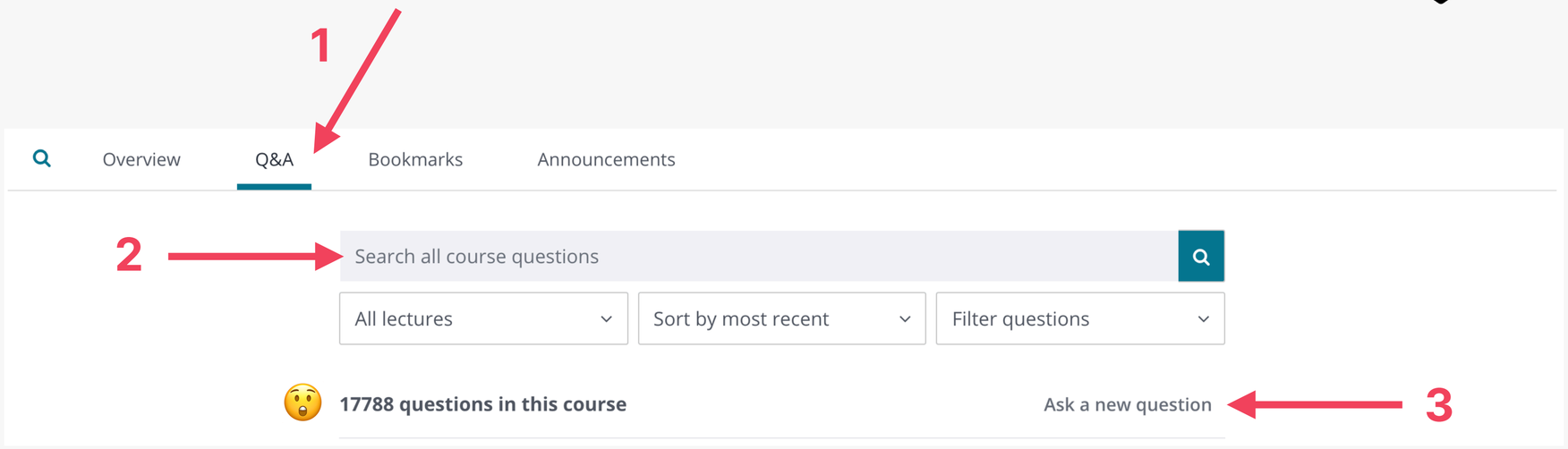
✅



✅

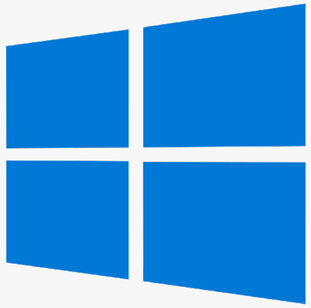
SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

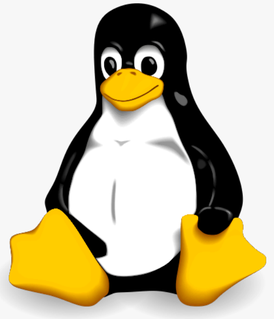
⁉️ If you have an error or a question, **start by trying to solve it yourself! This is essential for your progress. If you can't solve it, check the Q&A section**. If that doesn't help, you can **ask a new question**. Use a short description, and post code on codepen.io.



SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

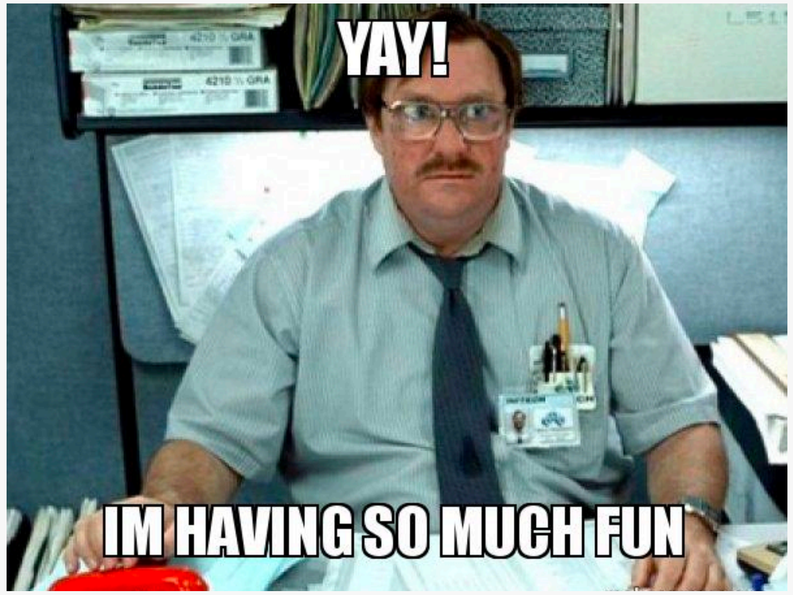
🖥️ **I recorded this course on a Mac, but everything works the exact same way on Windows or Linux.** If something doesn't work on your computer, it's **NOT** because you're using a different OS.





SOME QUICK CONSIDERATIONS BEFORE WE START... 🚀

😍 **Most importantly, have fun!** It's so rewarding to see something that **YOU** have built **YOURSELF**! So if you're feeling frustrated, stop whatever you're doing, and come back later!



And I mean **REAL** fun 😅

SECTION 02 — HTML FUNDAMENTALS

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

HTML FUNDAMENTALS

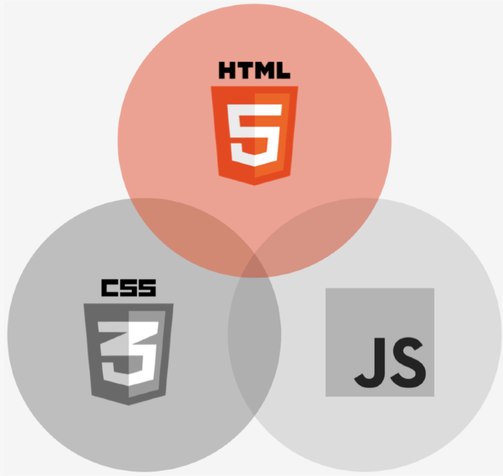
LECTURE

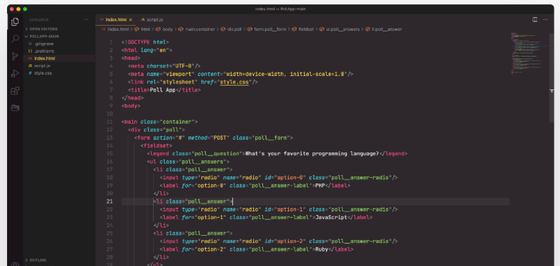
INTRODUCTION TO HTML

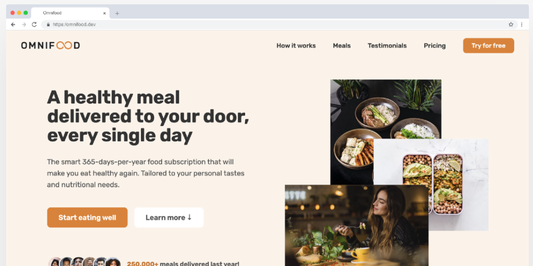
WHAT IS **HTML**?

HTML

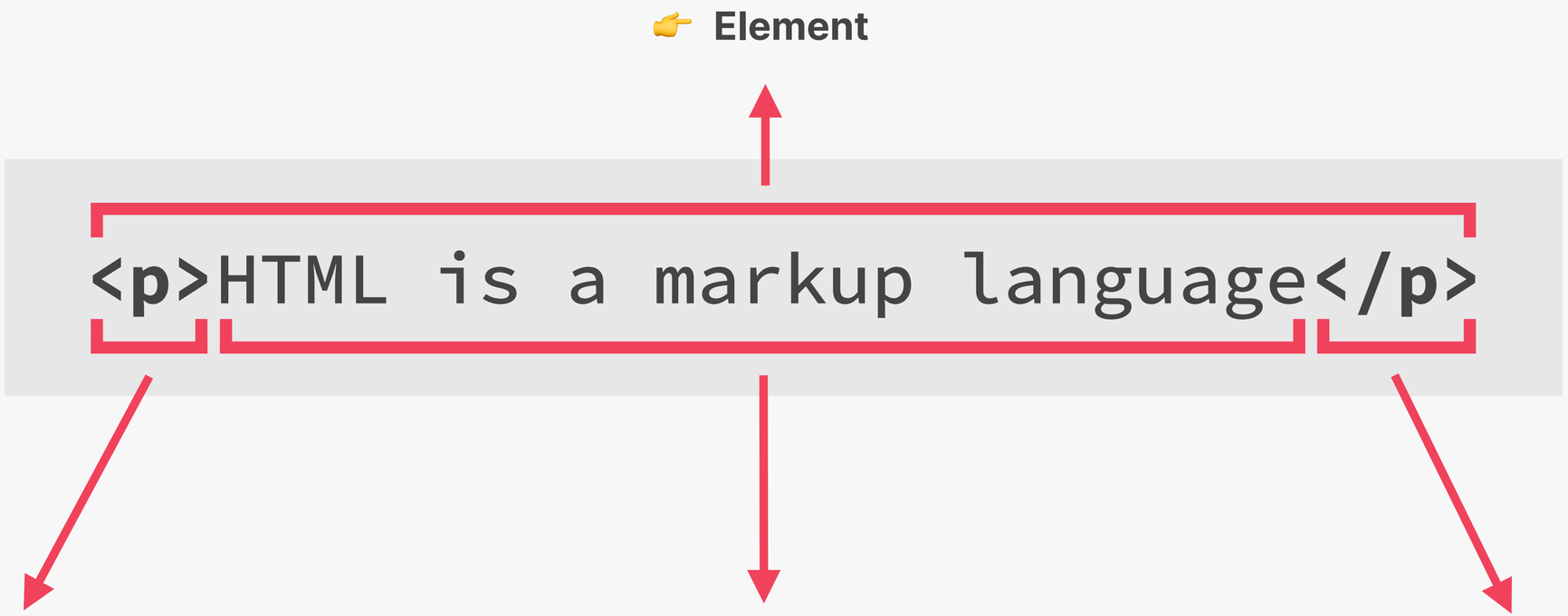
- 👉 **H**yper**T**ext **M**arkup **L**anguage
- 👉 HTML is a markup language that web developers use to **structure and describe the content** of a webpage (*not a programming language*)
- 👉 HTML consists of **elements** that describe different types of content: paragraphs, links, headings, images, video, etc.
- 👉 Web browsers understand HTML and **render HTML code as websites**







ANATOMY OF AN HTML ELEMENT



👉 **Opening tag:** Name of the element, wrapped in < and >

👉 **Content:** Content of the element, in this example text. But it might be another element (**child element**). Some elements have **no content** (e.g. `<img>`)

👉 **Closing tag:** Same as opening tag, but with a `/`. When element has no content, it's omitted

SECTION 03 — CSS FUNDAMENTALS

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

CSS FUNDAMENTALS

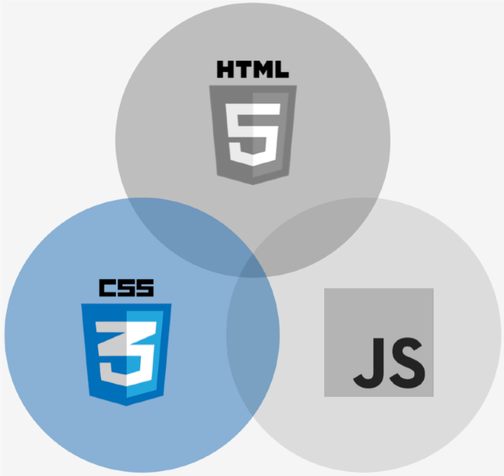
LECTURE

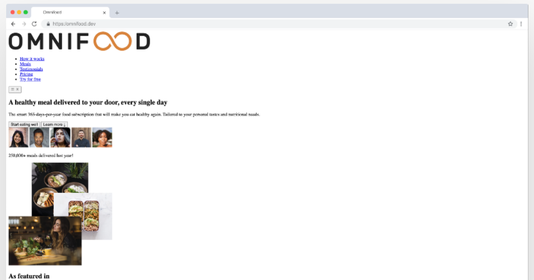
INTRODUCTION TO CSS

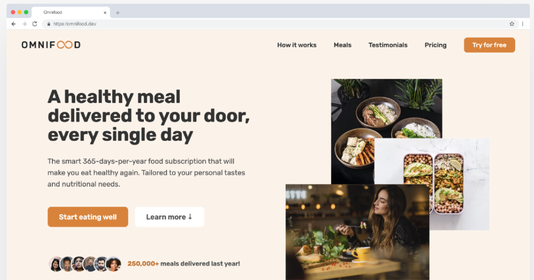
WHAT IS **CSS**?

CSS

- 👉 **C**ascading **S**tyle **S**heets
- 👉 CSS describes the **visual style and presentation** of the **content written in HTML**
- 👉 CSS consists of countless **properties** that developers use to format the content: properties about font, text, spacing, layout, etc.

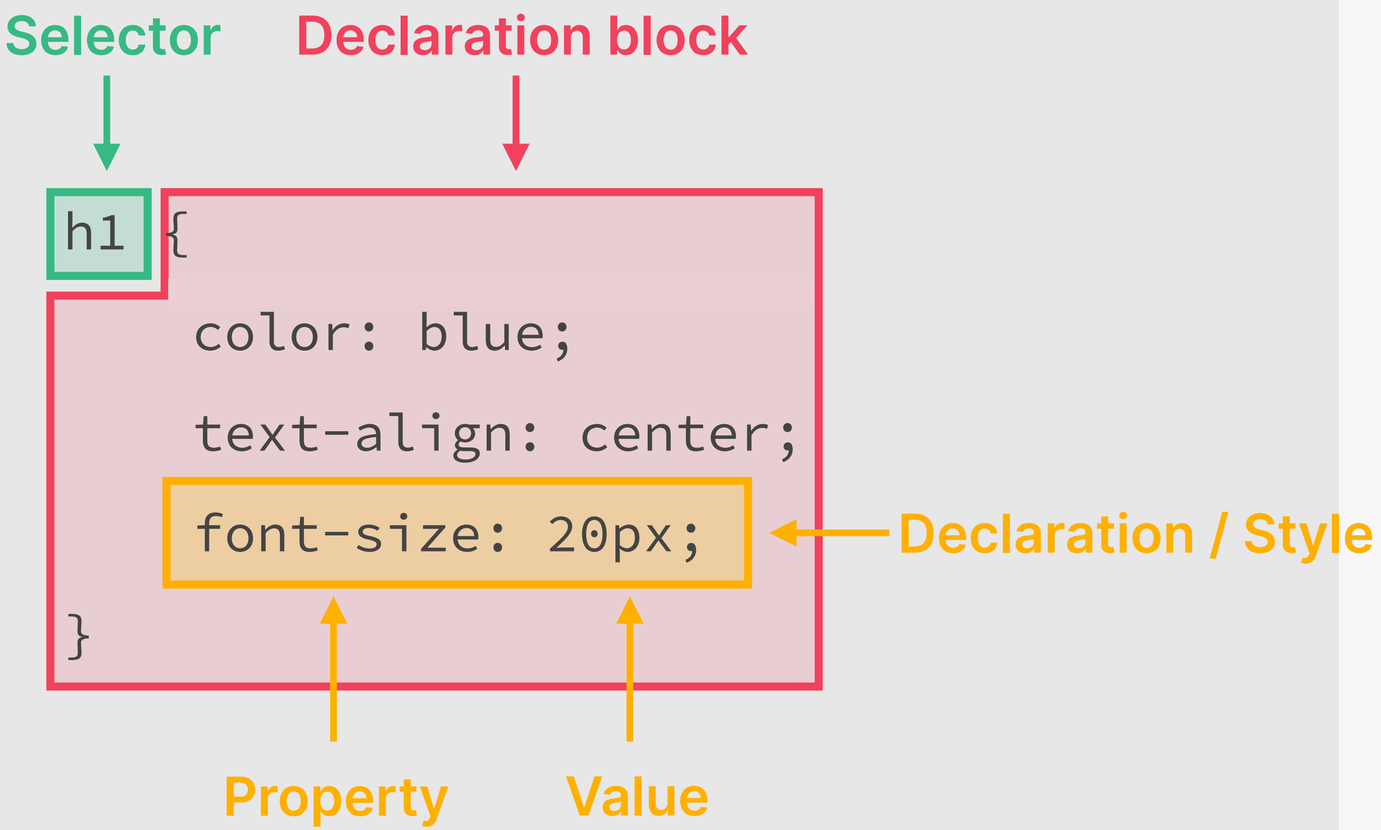






HOW WE **SELECT** AND **STYLE** ELEMENTS

A CSS RULE



JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

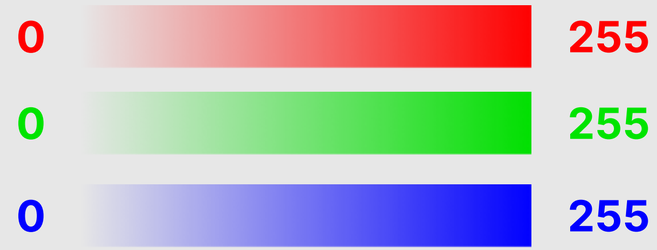
CSS FUNDAMENTALS

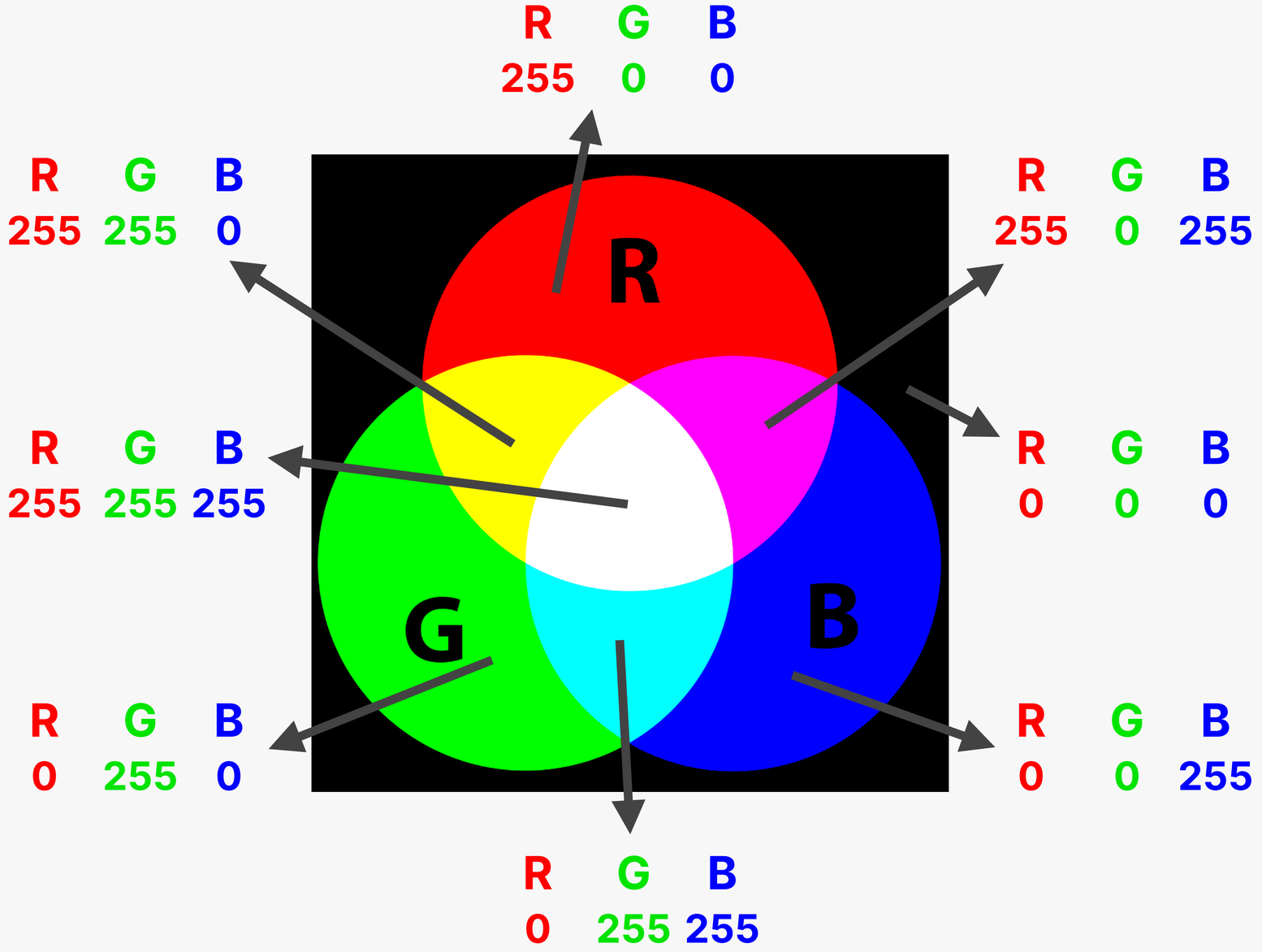
LECTURE

WORKING WITH COLORS

THE **RGB** MODEL

- 👉 **RGB Model:** Every color can be represented by a combination of **RED**, **GREEN** and **BLUE**
- 👉 Each of the 3 base colors can take a value between **0** and **255**, which leads to 16.8 million different colors





DEFINING COLORS IN CSS

1 RGB / RGBA NOTATION

👉 Regular RGB model

```
rgb(0, 255, 255)
```

👉 RGB with **transparency** (“alpha”)

```
rgba(0, 255, 255, 0.3)
```

```
#f4b33f
```

```
rgb(244, 179, 63)
```

```
rgba(244, 179, 63, 0.7)
```

2 HEXADECIMAL NOTATION

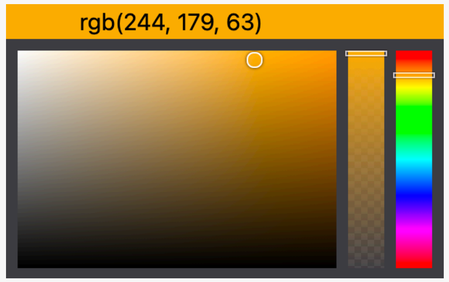
👉 Instead of using a scale from 0 to 255, we go from **0 to ff** (255 in hexadecimal numbers)

```
#00ffff
```

👉 Shorthand, when all colors are identical pairs

```
#0ff
```

💡 In practice, we mostly use **hexadecimal** colors, and **rgba** when we need transparency



👉 Color picker in VS Code

SHADES OF **GREY**

- 👉 When colors in all 3 channels are the same, we get a **grey color**
- 👉 There are 256 pure grays to choose from

`rgb(0, 0, 0) / #000000 / #000`

`rgb(69, 69, 69) / #444444 / #444`

`rgb(183, 183, 183) / #b7b7b7`

`rgb(255, 255, 255) / #ffffff / #fff`

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

CSS FUNDAMENTALS

LECTURE

CSS THEORY #1: CONFLICTS BETWEEN SELECTORS

CONFLICTING SELECTORS AND DECLARATIONS

```html
<p id="author-text" class="author">
  Posted by Laura Jones on Monday, June 21st 2027
</p>
```

```css
.author {
  font-style: italic;
  font-size: 18px;
}

#author-text {
  font-size: 20px;
}

p,
li {
  font-family: sans-serif;
  color: #444444;
  font-size: 22px;
}
```

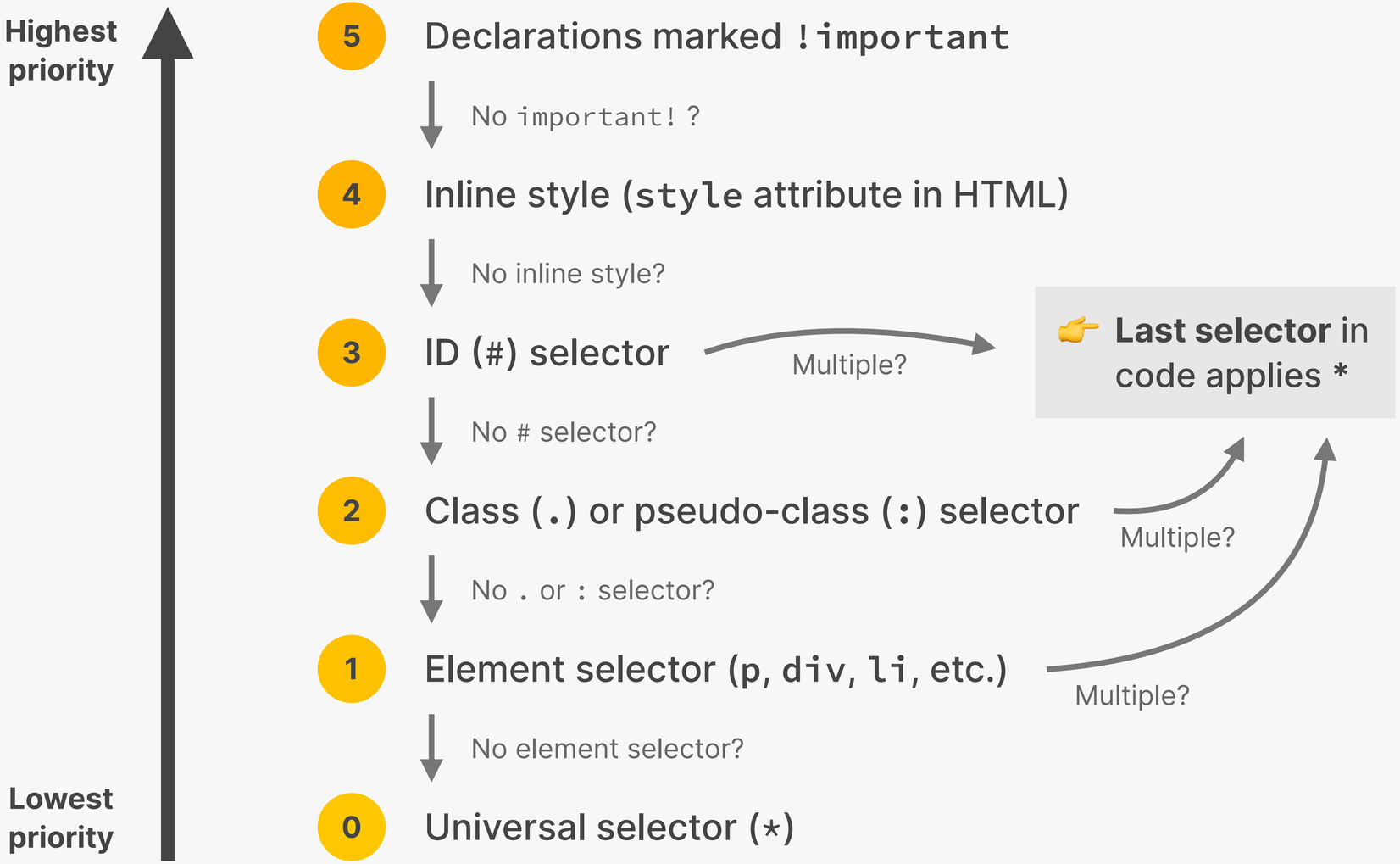
🤔 There are **multiple selectors** selecting the same element. **Which one of them applies?**

🤓 ***All of them. All rules and properties are applied!***

🤔 But there are **conflicting** `font-size` declarations! Is it `18px`, or `20px`, or `22px`?

🤓 *Let's see how it works...*

RESOLVING CONFLICTING DECLARATIONS




```css
.author {
  font-style: italic;
  font-size: 18px;
}

#author-text {
  font-size: 20px;
}

p,
li {
  font-family: sans-serif;
  color: #444444;
  font-size: 22px;
}
```

👉 There is an ID selector (`#author-text`), so **for the conflicting `font-size` property**, this is the selector that applies

☝️ * It's a bit more complicated in reality

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

CSS FUNDAMENTALS

LECTURE

CSS THEORY #2: INHERITANCE AND THE UNIVERSAL SELECTOR

HOW **INHERITANCE** WORKS

Parent element

```
<body>
  <nav>
    This is the navigation
  </nav>

  <h1>My website</h1>

  <p>
    The text is this paragraph
    is completely irrelevant 😄
  </p>

</body>
```

```
color: #444444;
font-size: 16px;
font-family: sans-serif;
```

```
color: #444444;
font-size: 16px;
font-family: sans-serif;
```

```
color: #444444;
font-size: 16px;
font-family: sans-serif;
```

```
body {
  color: #444444;
  font-size: 16px;
  font-family: sans-serif;

  border-top: 10px solid #1098ad;
}
```

The `border` property does **NOT** get inherited

```
h1 {
  color: #1098ad;
  font-size: 32px;
  text-transform: uppercase;
}
```

OVERRIDING INHERITED STYLES

☝️ Not all properties get inherited. It's mostly ones **related to text**: `font-family`, `font-size`, `font-weight`, `font-style`, `color`, `line-height`, `letter-spacing`, `text-align`, `text-transform`, `text-shadow`, `list-style`, etc.

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

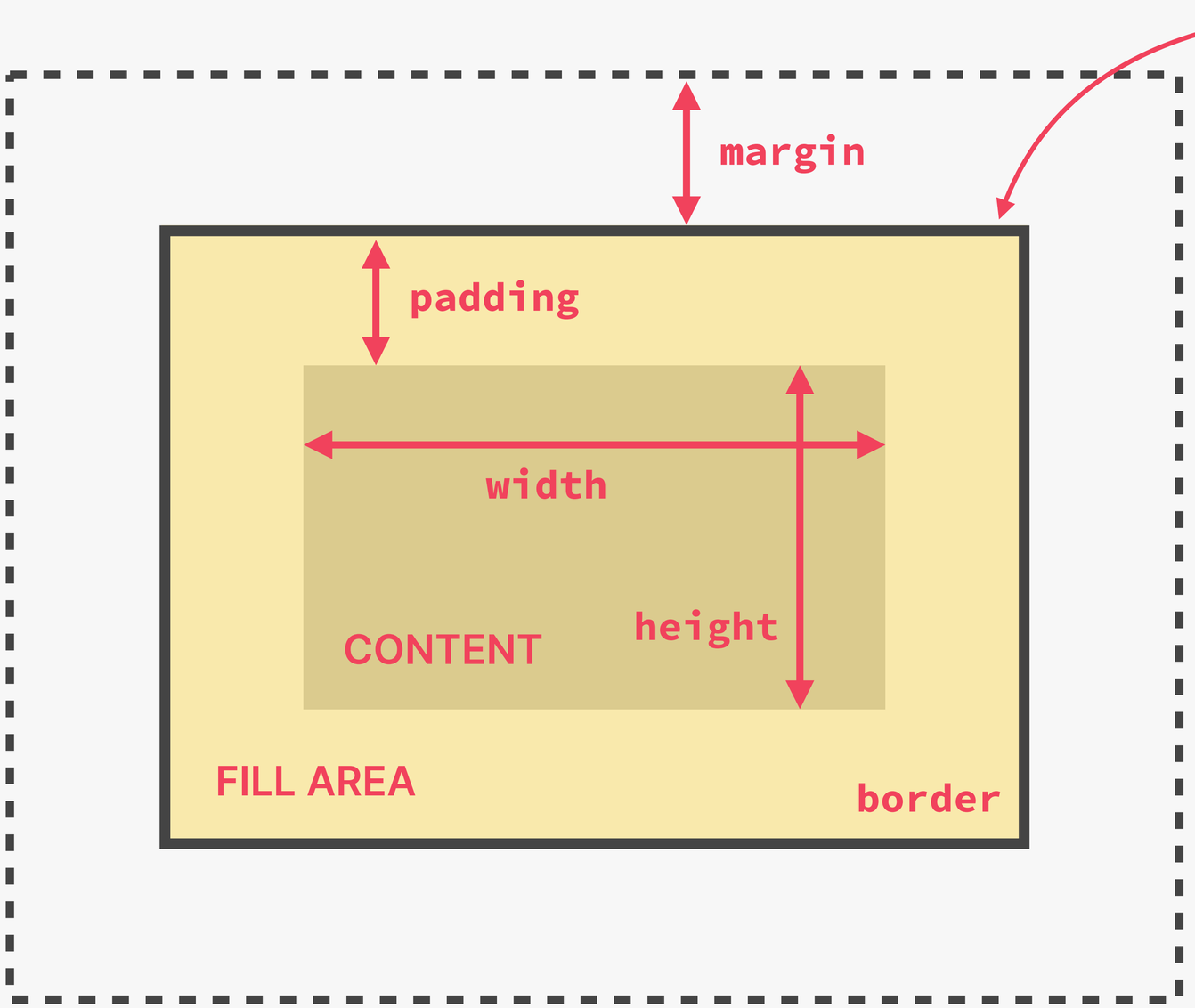
SECTION

CSS FUNDAMENTALS

LECTURE

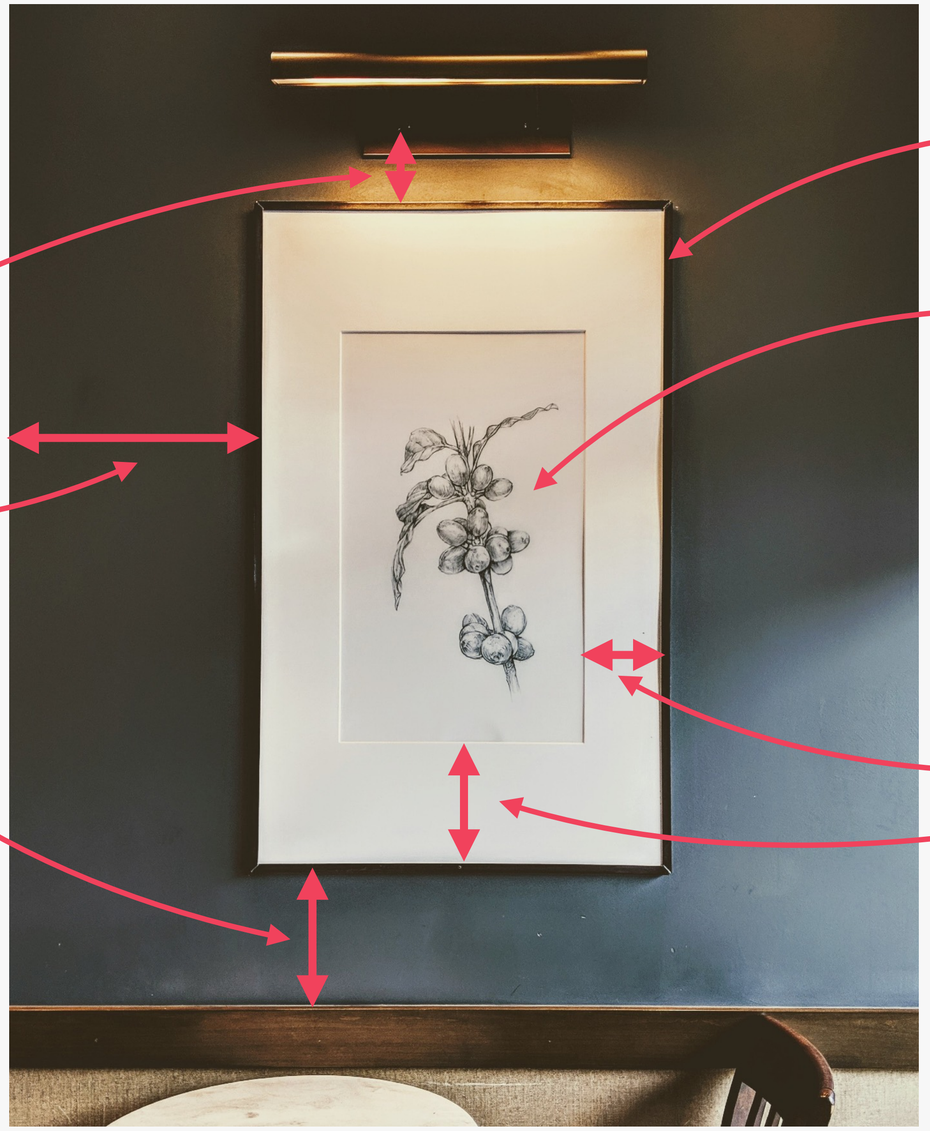
CSS THEORY #3: THE CSS BOX MODEL

THE **CSS BOX MODEL**



- 👉 **Content**: Text, images, etc.
- 👉 **Border**: A line around the element, still **inside** of the element
- 👉 **Padding**: Invisible space around the content, **inside** of the element
- 👉 **Margin**: Space **outside** of the element, between elements
- 👉 **Fill area**: Area that gets filled with **background color** or **background image**

ANALOGY FOR THE CSS BOX MODEL

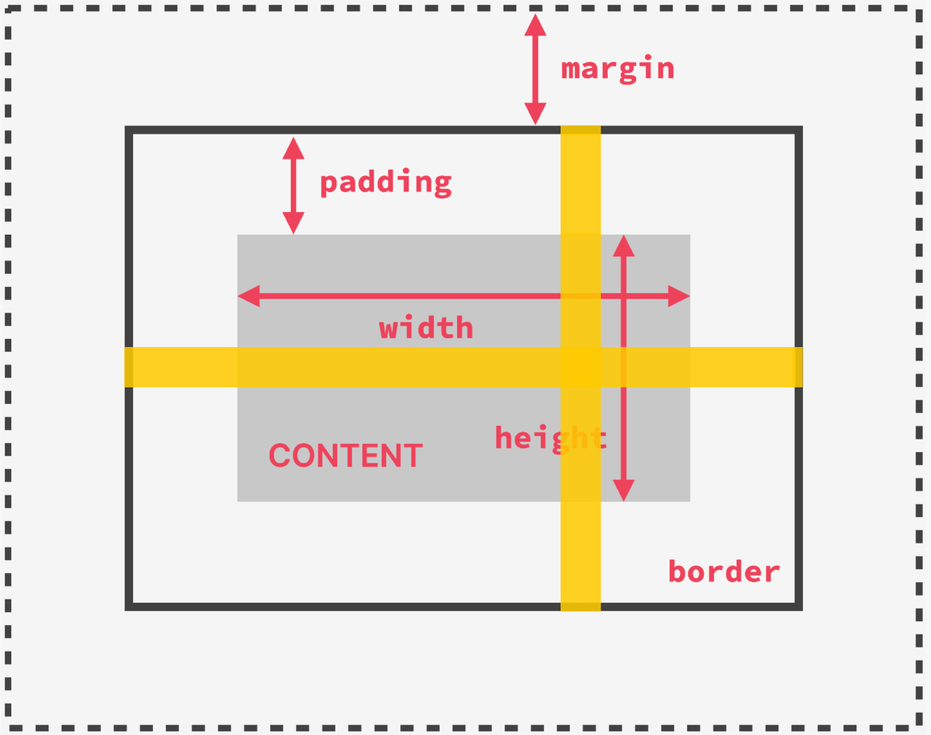


ELEMENT **HEIGHT** AND **WIDTH** CALCULATION

Final element width = left border + left padding + width + right padding + right border

Final element height = top border + top padding + height + bottom padding + bottom border

- 👉 We can specify all these values using CSS properties
- 👉 This is the **default behavior**, but we can change it



JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

CSS FUNDAMENTALS

LECTURE

CSS THEORY #4: TYPES OF BOXES

BLOCK-LEVEL ELEMENTS

- 👉 Elements are formatted visually as **blocks**
- 👉 Elements occupy **100% of parent element's width**, no matter the content
- 👉 Elements are **stacked vertically** by default, one after another
- 👉 The box-model **applies as showed** earlier

Default elements: `body, main, header, footer, section, nav, aside, div, h1-h6, p, ul, ol, li, etc.`

With CSS: `display: block`

The Basic Language of the Web: HTML

*Posted by **Laura Jones** on Monday, June 21st 2027*

All modern websites and web applications are built using three *fundamental* technologies: HTML, CSS and JavaScript. These are the languages of the web.

In this post, let's focus on HTML. We will learn what HTML is all about, and why should learn it.

What is HTML?

Lorem ipsum dolor sit amet consectetur adipisicing elit. Quam recusandae reprehenderit vitae ratione veritatis corrupti sit ut vero, dolores nulla exercitationem eos quod iusto incidunt, perferendis alias tenetur. Est, vel!

In HTML, each element is made up of 3 parts:

1. **The opening tag**
2. The closing tag
3. *The actual element*

You can learn more at the MDN Web Docs.

INLINE ELEMENTS

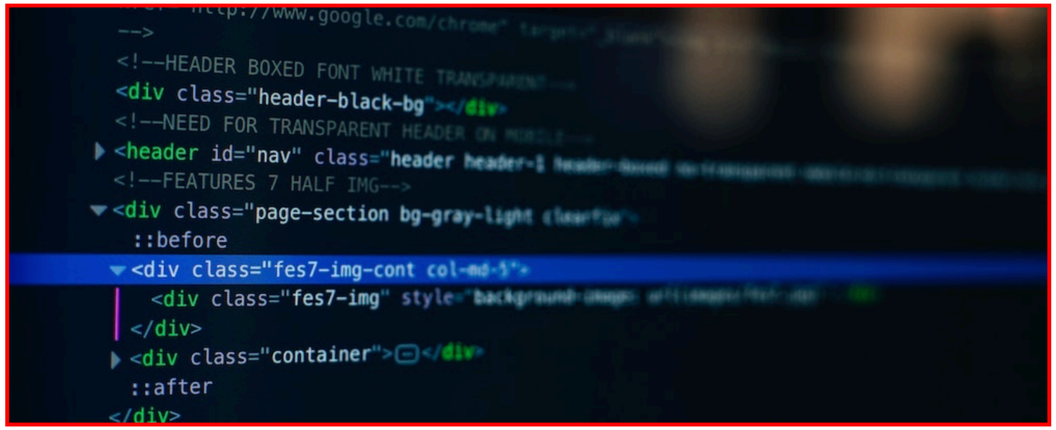
- 👉 Occupies only the space **necessary for its content**
- 👉 Causes **no line-breaks** after or before the element
- 👉 Box model applies in a different way: **heights and widths do not apply**
- 👉 **Paddings and margins** are applied **only horizontally** (left and right)

Default elements: `a, img, strong, em, button, etc.`

With CSS: `display: inline`

The Basic Language of the Web: HTML

*Posted by **Laura Jones** on Monday, June 21st 2027*



All modern websites and web applications are built using three *fundamental* technologies: HTML, CSS and JavaScript. These are the languages of the web.

In this post, let's focus on HTML. We will learn what HTML is all about, and why should learn it.

What is HTML?

Lorem ipsum dolor sit amet consectetur adipisicing elit. Quam recusandae reprehenderit vitae ratione veritatis corrupti sit ut vero, dolores nulla exercitationem eos quod iusto incidunt, perferendis alias tenetur. Est, vel!

In HTML, each element is made up of 3 parts:

1. **The opening tag**
2. The closing tag
3. *The actual element*

You can learn more at the MDN Web Docs.

SUMMARY: INLINE, BLOCK-LEVEL AND INLINE-BLOCK BOXES

BLOCK-LEVEL BOXES

- 👉 Elements formatted visually as blocks
- 👉 100% of parent's width
- 👉 Vertically, one after another
- 👉 Box-model applies as showed

INLINE-BLOCK BOXES

- 👉 Looks like inline from the **outside**, behaves like block-level on the **inside**
- 👉 Occupies only content's space
- 👉 Causes no line-breaks
- 👉 Box-model applies as showed

```
display: inline-block
```

INLINE BOXES

- 👉 Occupies only content's space
- 👉 Causes no line-breaks
- 👉 Box model is different: heights and widths do not apply
- 👉 Paddings and margins only horizontal (left and right)

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

CSS FUNDAMENTALS

LECTURE

CSS THEORY #5: ABSOLUTE POSITIONING

NORMAL FLOW **VS.** ABSOLUTE POSITIONING

NORMAL FLOW

- 👉 Default positioning
- 👉 Element is “**in** flow”
- 👉 Elements are simply laid out according to their order in the HTML code

```
Default positioning
position: relative
```

ABSOLUTE POSITIONING

- 👉 Element is removed from the normal flow: “**out of** flow”
- 👉 No impact on surrounding elements, might overlap them
- 👉 We use `top`, `bottom`, `left`, or `right` to offset the element from its **relatively positioned container**

```
position: absolute
```

UNDERSTANDING **ABSOLUTE** POSITIONING


```
.el {
  position: absolute;
  top: 100px;
  left: 200px;
  background-color: #f4b33f;
}
```

```
.container {
  position: relative;
  background-color: #f7e6c1;
}
```

SECTION 04 — LAYOUTS: FLOATS, FLEXBOX, AND CSS GRID FUNDAMENTALS

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

LAYOUTS: FLOATS, FLEXBOX, AND CSS GRID FUNDAMENTALS

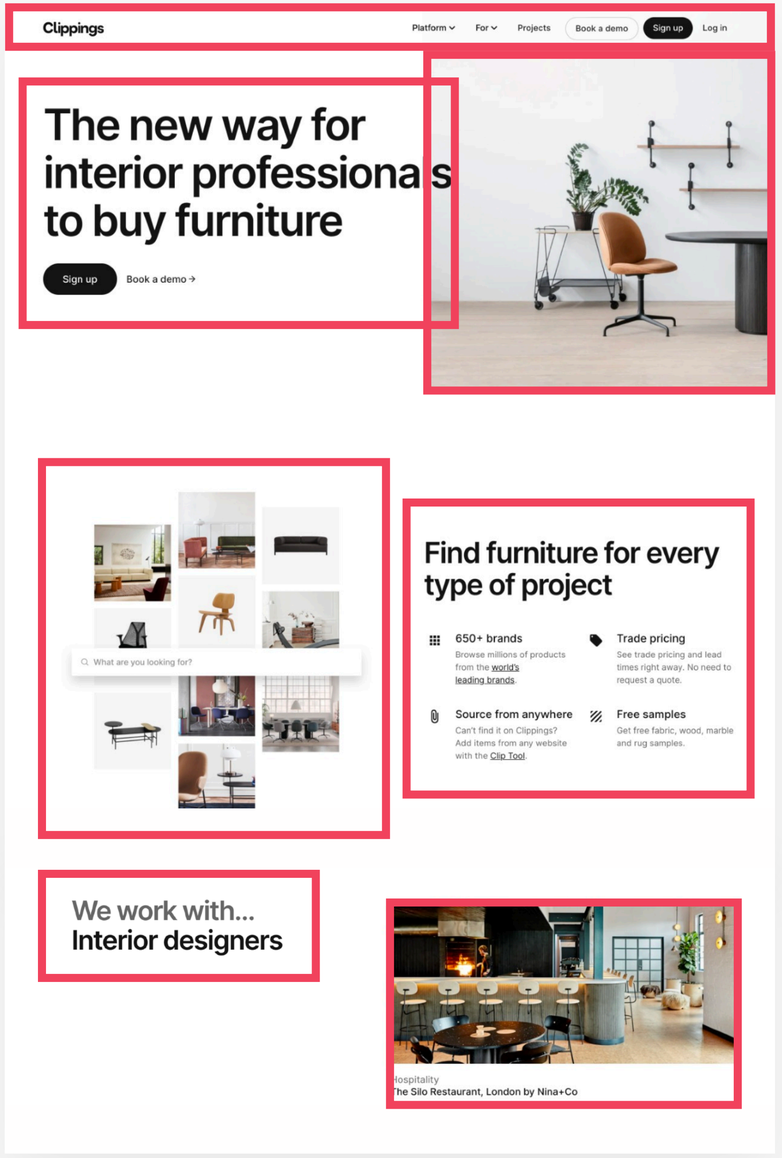
LECTURE

THE 3 WAYS OF BUILDING LAYOUTS

WHAT DOES “**LAYOUT**” MEAN?

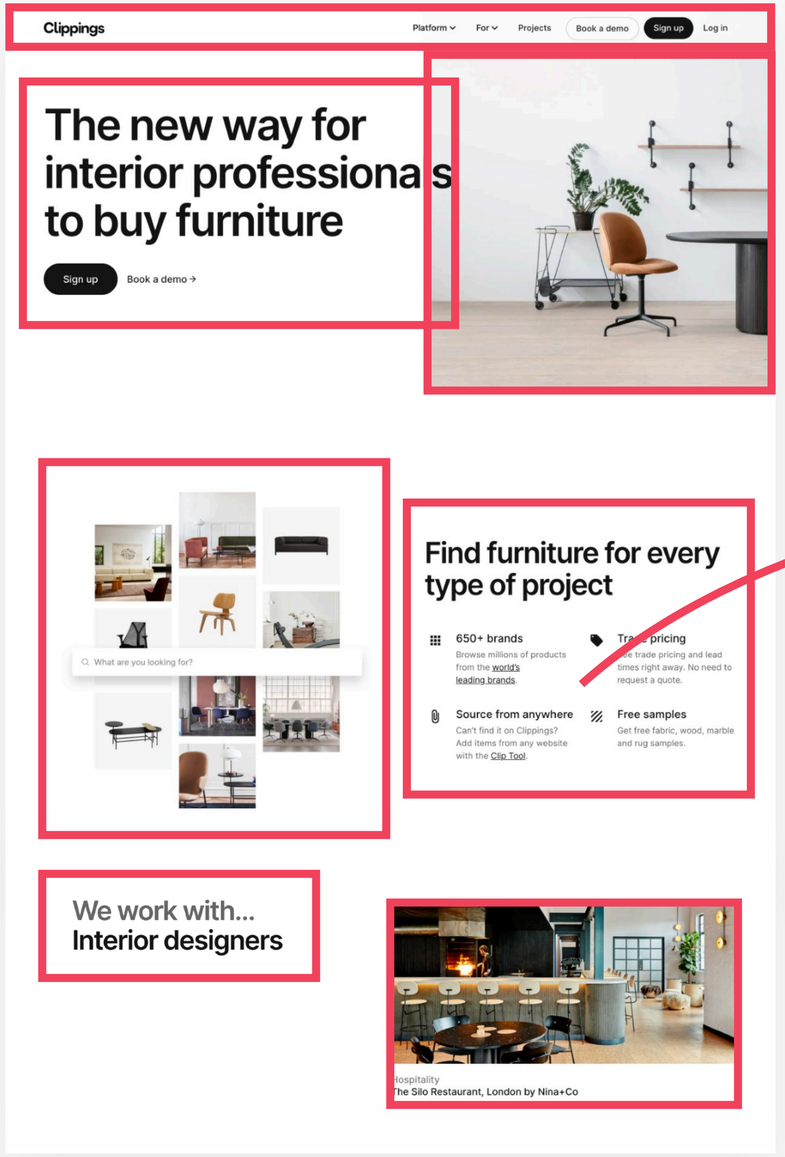
LAYOUT

- 👉 Layout is the way text, images and other content is placed and arranged on a webpage
- 👉 Layout gives the page a visual structure, into which we place our content
- 👉 **Building a layout:** arranging page elements into a visual structure, instead of simply having them placed one after another (normal flow)



PAGE LAYOUT **VS.** COMPONENT LAYOUT

PAGE LAYOUT



650+ brands

Browse millions of products from the world's leading brands.

Trade pricing

See trade pricing and lead times right away. No need to request a quote.

Source from anywhere

Can't find it on Clippings? Add items from any website with the Clip Tool.

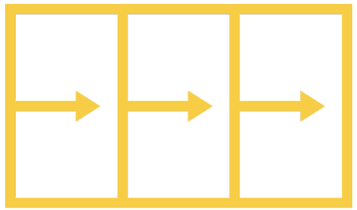
Free samples

Get free fabric, wood, marble and rug samples.

COMPONENT LAYOUT

THE **3 WAYS** OF BUILDING LAYOUTS WITH CSS

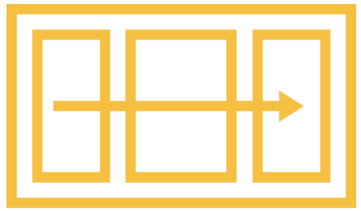
1



FLOAT LAYOUTS

The **old way of building layouts** of all sizes, using the `float` CSS property. Still used, but getting outdated fast.

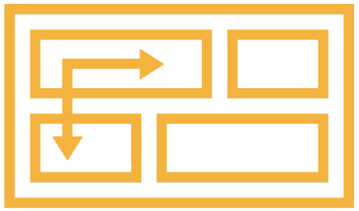
2



FLEXBOX

Modern way of laying out elements in a **1-dimensional row** without using floats. Perfect for **component layouts**.

3



CSS GRID

For laying out element in a fully-fledged **2-dimensional grid**. Perfect for **page layouts and complex components**.

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

LAYOUTS: FLOATS, FLEXBOX, AND CSS GRID FUNDAMENTALS

LECTURE

USING FLOATS

ABSOLUTE POSITIONING **VS.** FLOATS

NORMAL FLOW

- 👉 Default positioning
- 👉 Element is “**in** flow”
- 👉 Elements are simply laid out according to their order in the HTML code

```
Default positioning
position: relative
```

ABSOLUTE POSITIONING

- 👉 Element is removed from the normal flow: “**out of** flow”
- 👉 No impact on surrounding elements, might overlap them
- 👉 We use `top`, `bottom`, `left`, or `right` to offset the element from its **relatively positioned container**

```
position: absolute
```

=

≠

FLOATS

- 👉 Element is removed from the normal flow: “**out of** flow”
- 👉 Text and inline elements will wrap around the floated element
- 👉 The container will **not** adjust its height to the element

```
float: left
float: right
```

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

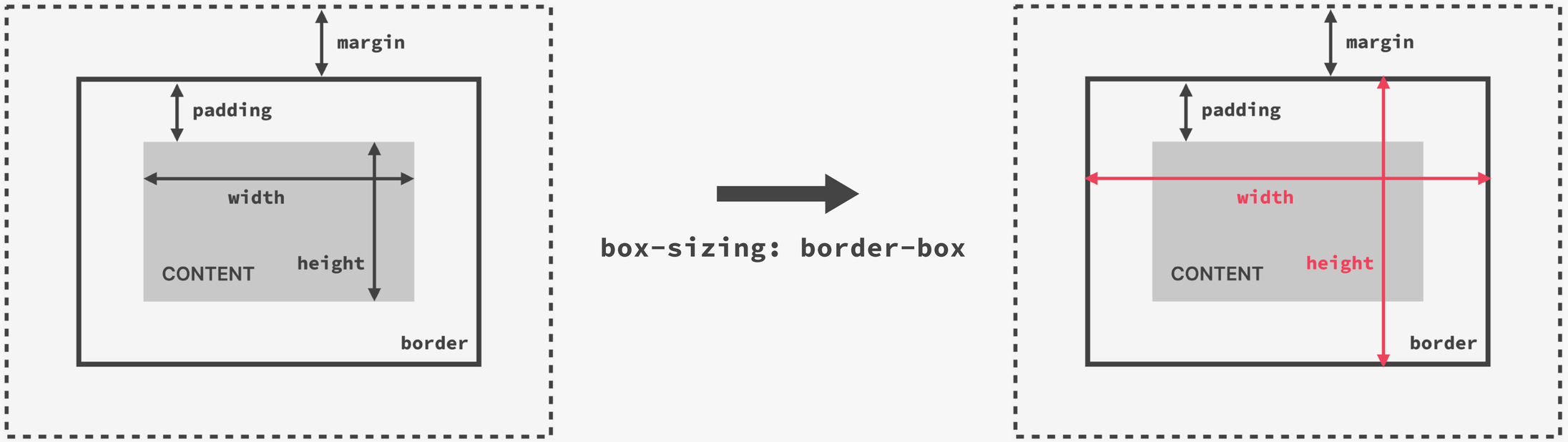
SECTION

LAYOUTS: FLOATS, FLEXBOX, AND CSS GRID FUNDAMENTALS

LECTURE

BOX-SIZING: BORDER-BOX

THE BOX MODEL WITH **BOX-SIZING: BORDER-BOX**



Final element width = ~~right border~~ + ~~right padding~~ + width + ~~left padding~~ + ~~left border~~

Final element height = ~~top border~~ + ~~top padding~~ + height + ~~bottom padding~~ + ~~bottom border~~

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

LAYOUTS: FLOATS, FLEXBOX, AND CSS GRID FUNDAMENTALS

LECTURE

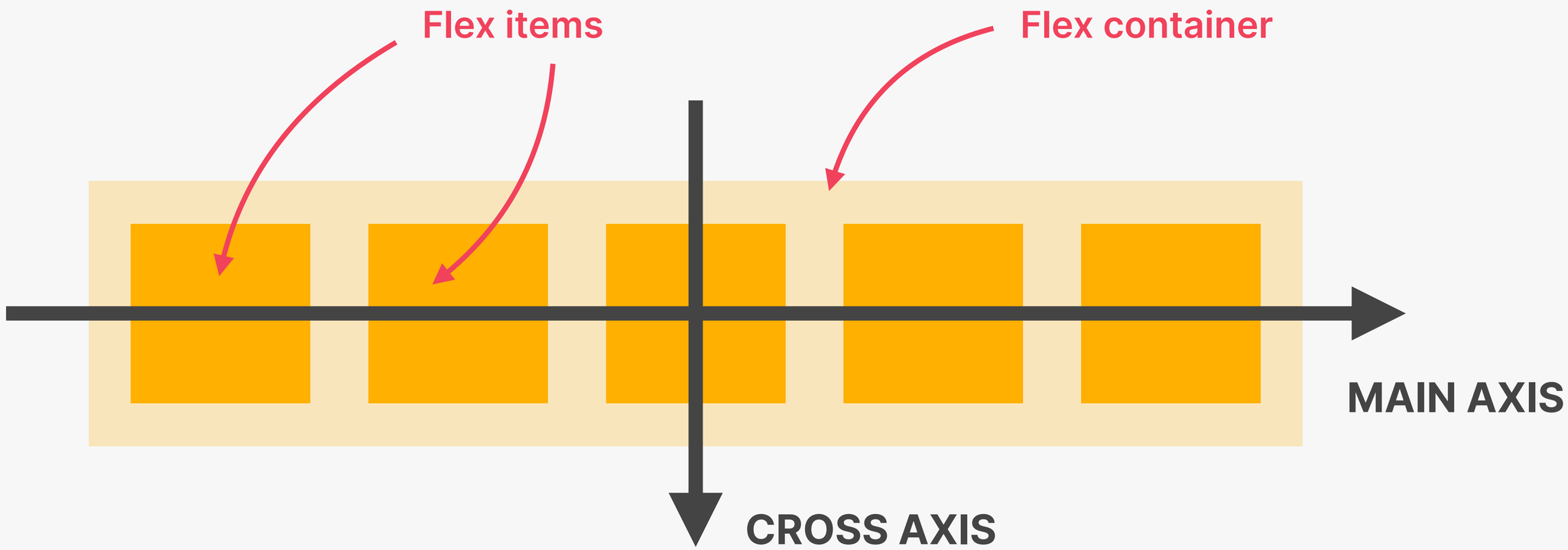
A FLEXBOX OVERVIEW

WHAT IS **FLEXBOX**?

FLEXBOX

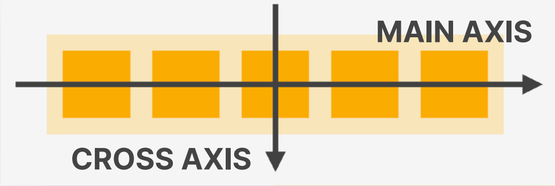
- 👉 Flexbox is a set of related **CSS properties** for **building 1-dimensional layouts**
- 👉 The main idea behind flexbox is that empty space inside a container element can be **automatically divided** by its child elements
- 👉 Flexbox makes it easy to automatically **align items to one another** inside a parent container, both horizontally and vertically
- 👉 Flexbox solves common problems such as **vertical centering** and creating **equal-height columns**
- 👉 Flexbox is perfect for **replacing floats**, allowing us to write fewer and cleaner HTML and CSS code

FLEXBOX **TERMINOLOGY**




```
display: flex
```

FLEX CONTAINER



1. **`gap: 0`** | `<length>`
   - 👉 *To create **space between items**, without using `margin`*
2. **`justify-content: flex-start`** | `flex-end` | `center` | `space-between` | `space-around` | `space-evenly`
   - 👉 *To align items along main axis (**horizontally**, by default)*
3. **`align-items: stretch`** | `flex-start` | `flex-end` | `center` | `baseline`
   - 👉 *To align items along cross axis (**vertically**, by default)*
4. **`flex-direction: row`** | `row-reverse` | `column` | `column-reverse`
   - 👉 *To define which is the **main axis***
5. **`flex-wrap: nowrap`** | `wrap` | `wrap-reverse`
   - 👉 *To allow items to **wrap into a new line** if they are too large*
6. **`align-content: stretch`** | `flex-start` | `flex-end` | `center` | `space-between` | `space-around`
   - 👉 *Only applies when there are **multiple lines** (`flex-wrap: wrap`)*

FLEX ITEMS

1. **`align-self: auto`** | `stretch` | `flex-start` | `flex-end` | `center` | `baseline`
   - 👉 *To **overwrite** `align-items` for individual flex items*
2. **`flex-grow: 0`** | `<integer>`
   - 👉 *To allow an element **to grow** (0 means no, 1+ means yes)*
3. **`flex-shrink: 1`** | `<integer>`
   - 👉 *To allow an element **to shrink** (0 means no, 1+ means yes)*
4. **`flex-basis: auto`** | `<length>`
   - 👉 *To define an item's width, **instead of the `width` property***
5. **`flex: 0 1 auto`** | `<int> <int> <len>`
   - 👉 ***Recommended** shorthand for `flex-grow`, `-shrink`, `-basis`.*
6. **`order: 0`** | `<integer>`
   - 👉 *Controls order of items. -1 makes item **first**, 1 makes it **last***

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

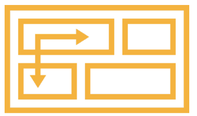
LAYOUTS: FLOATS, FLEXBOX, AND CSS GRID FUNDAMENTALS

LECTURE

A CSS GRID OVERVIEW

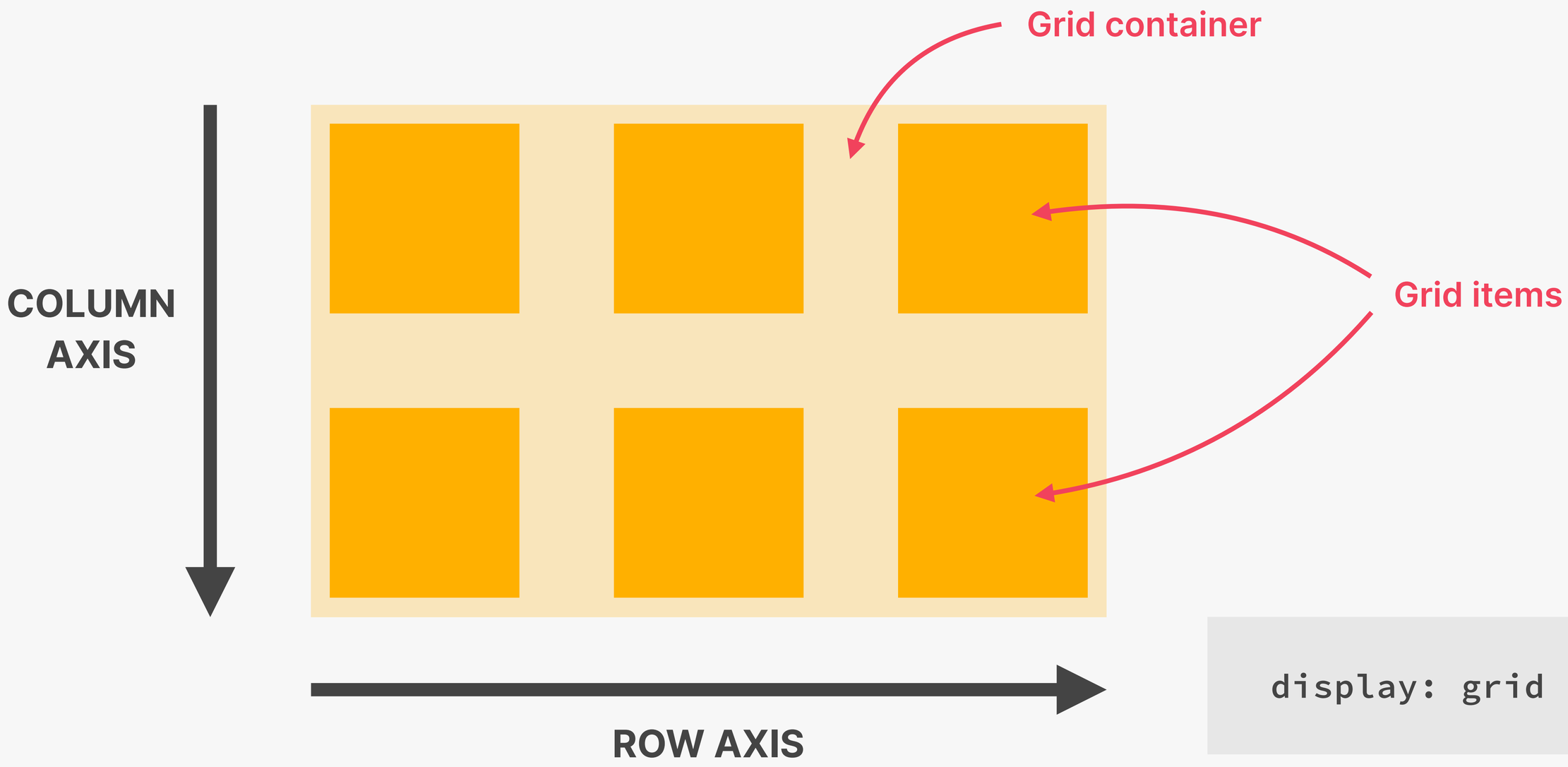
WHAT IS **CSS GRID?**

CSS GRID



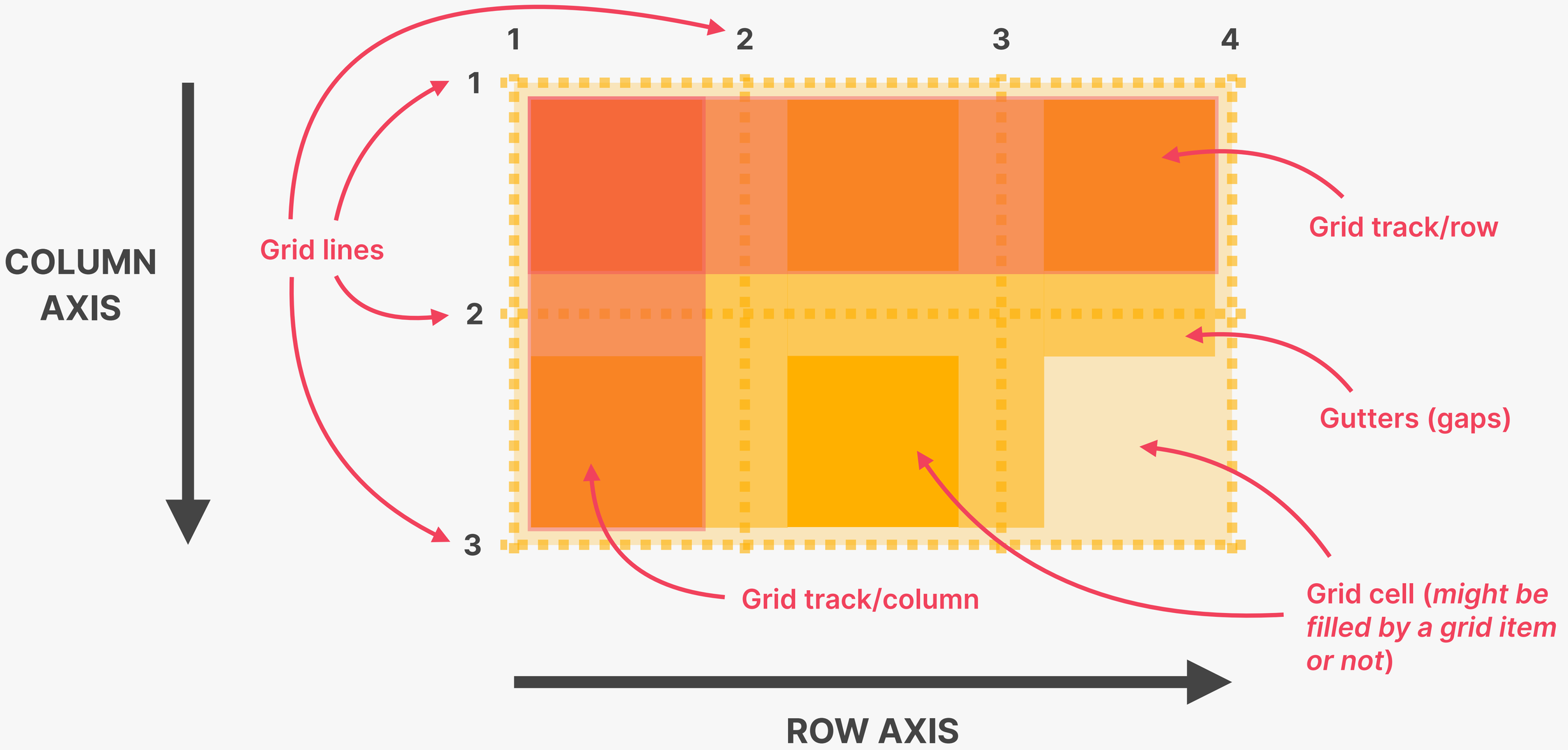
- 👉 CSS Grid is a set of **CSS properties** for **building 2-dimensional layouts**
- 👉 The main idea behind CSS Grid is that we **divide a container element into rows and columns** that can be filled with its child elements
- 👉 In two-dimensional contexts, CSS Grid allows us to write **less nested HTML** and **easier-to-read CSS**
- 👉 CSS Grid is **not meant to replace flexbox**! Instead, they work perfectly together. Need a **1D** layout? Use flexbox. Need a **2D** layout? Use CSS Grid.

BASIC CSS GRID **TERMINOLOGY**




```
display: grid
```

MORE CSS GRID **TERMINOLOGY**



GRID CONTAINER

1

```
grid-template-rows: <track size>*
grid-template-columns: <track size>*
```

👉 *To establish the grid **row and column tracks**. One length unit for each track. Any unit can be used, new **fr** fills unused space*

2

```
row-gap: 0 | <length>
column-gap: 0 | <length>
```

] `gap: 0 | <length>`

👉 *To **create empty space** between tracks*

3

```
justify-items: stretch | start | center | end
align-items: stretch | start | center | end
```

👉 *To align items inside rows / columns (**horizontally** / **vertically**)*

4

```
justify-content: start | start | center | end | ...
align-content: start | start | center | end | ...
```

👉 *To align entire **grid inside grid container**. Only applies if container is larger than the grid*

GRID ITEMS

1

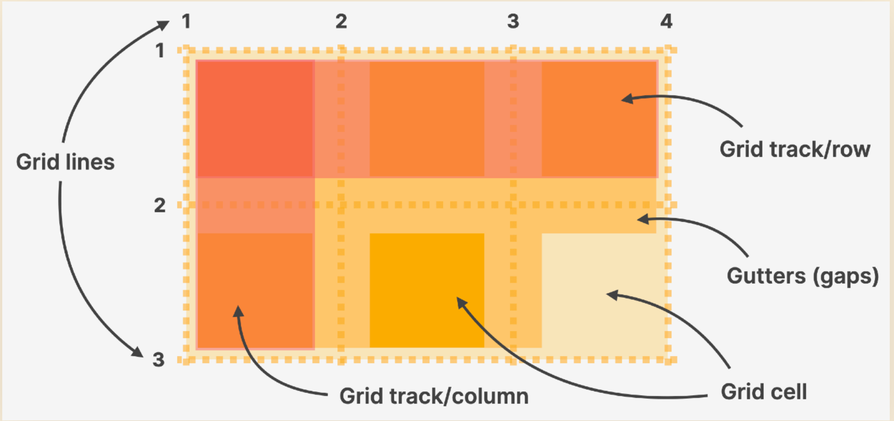
```
grid-column: <start line> / <end line> | span <number>
grid-row: <start line> / <end line> | span <number>
```

👉 *To **place a grid item** into a specific cell, based on line numbers. `span` keyword can be used to span an item across more cells*

2

```
justify-self: stretch | start | center | end
align-self: stretch | start | center | end
```

👉 *To **overwrite** `justify-items` / `align-items` for single items*



☝️ *This list of CSS Grid properties is not exhaustive, but enough to get started.*

SECTION 05 — WEB DESIGN RULES AND FRAMEWORK

JONAS.IO
SCHMEDTMANN

BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

SECTION

WEB DESIGN RULES AND FRAMEWORK

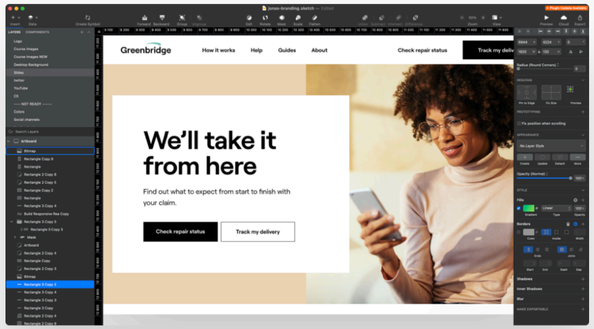
LECTURE

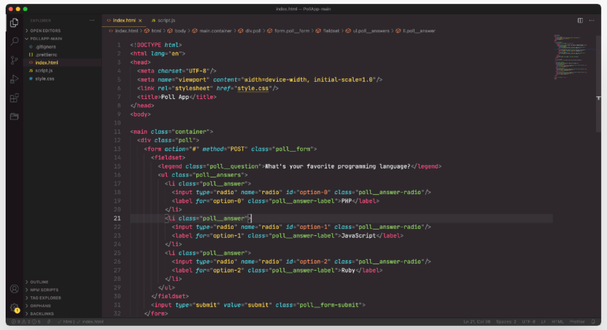
OVERVIEW OF WEB DESIGN AND WEBSITE PERSONALITIES

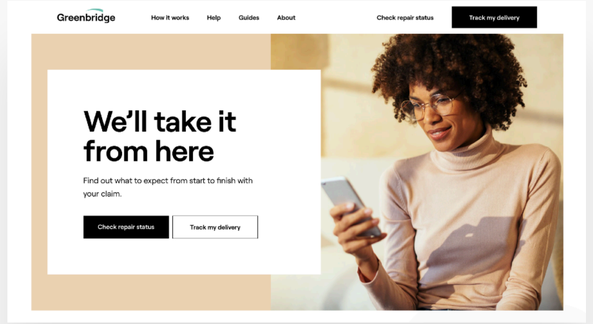
WEB DESIGN VS. DEVELOPMENT

Web **designers** create the overall **look and feel** of a website

Web **developers** implement the design using **HTML, CSS and JavaScript code**







DESIGNER

DEVELOPER

USER

DESIGNER + DEVELOPER

WHY TAKE DESIGN **SERIOUSLY?**

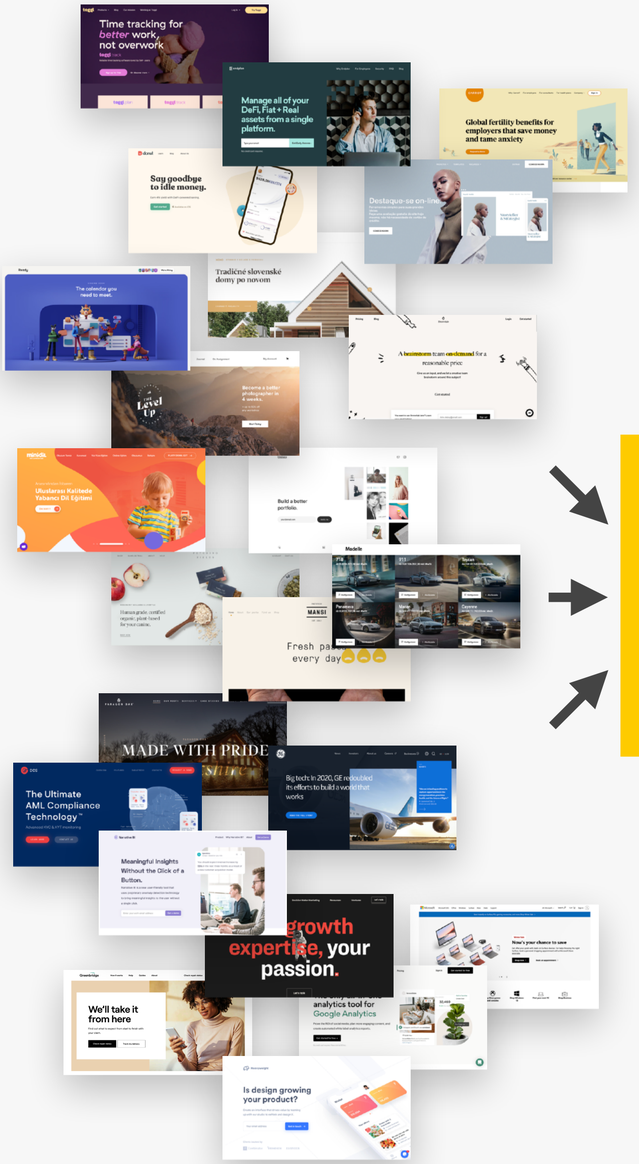
GOOD DESIGN

- ✅ Creates an immediate and lasting **good impression** of the brand or product;
- ✅ Makes the user **trust** the brand right away;
- ✅ Increases the user's **perceived value** of the brand or product;
- ✅ Gives users exactly **what they were looking for** when coming to the site, e.g. purchasing a product or finding information.

BAD DESIGN

- 🚫 Makes users believe the brand doesn't really care about their product or service;
- 🚫 Makes the user insecure about trusting the brand;
- 🚫 Makes the brand or product seem "cheap";
- 🚫 Leaves users confused, and makes it hard to for them to reach their goal.

ANYONE CAN LEARN GOOD DESIGN!

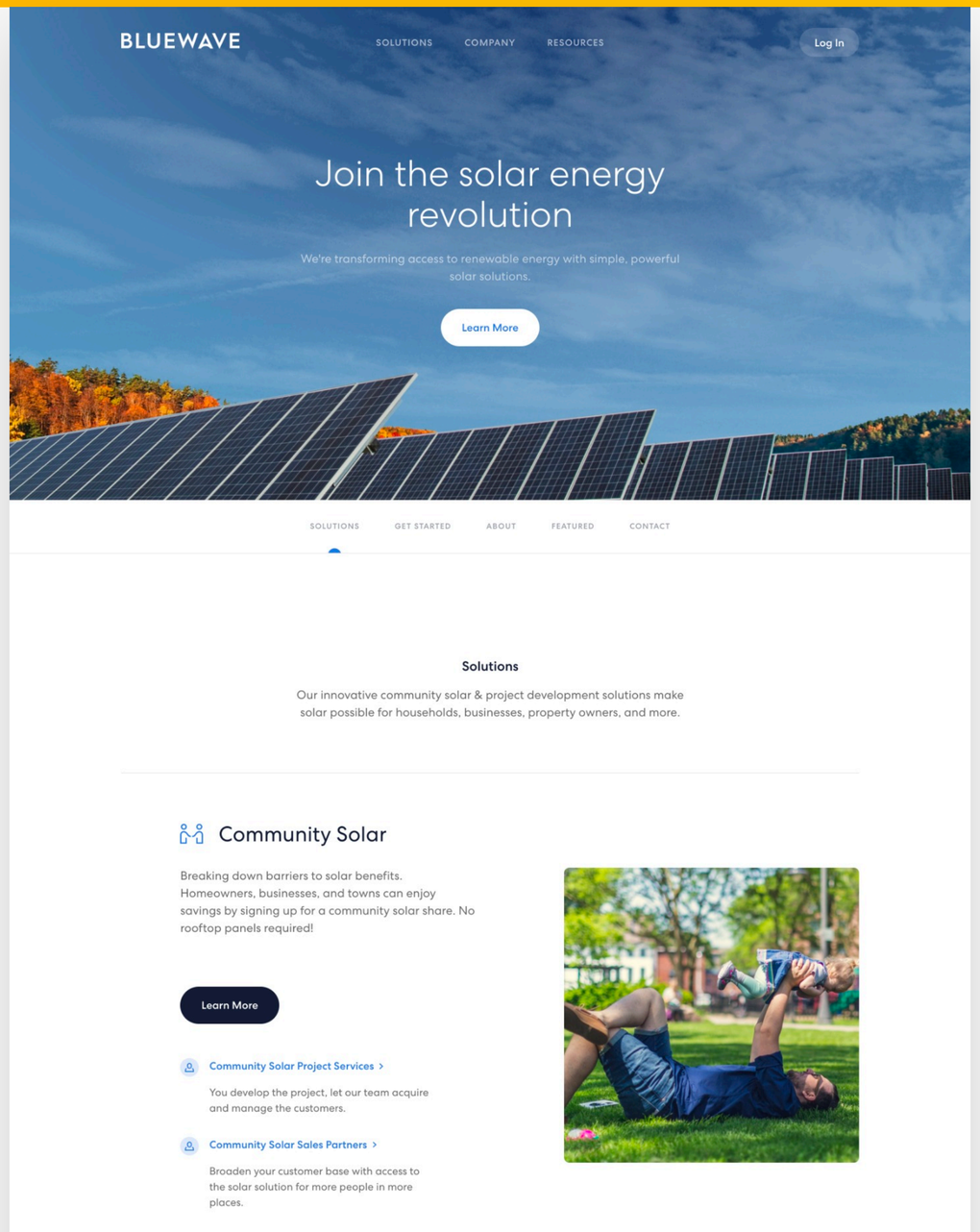


Good web design is **not subjective or creative** 👉 Everyone can learn basics by following a **framework/system**

100s of well-designed sites **deconstructed** → Distilled into easy-to-learn and easy-to-apply **rules** → Divided in 9 different areas of design: **ingredients** → Rules will be applied based on **website personality**

WEB DESIGN **INGREDIENTS** YOU WILL LEARN ABOUT

1. Typography
2. Colors
3. Images/Illustrations
4. Icons
5. Shadows



6. Border-radius
7. Whitespace
8. Visual Hierarchy
9. User Experience
10. Components/Layout

☝ Design decisions for each ingredient are based on **website personality**