JONAS.IO
SCHMEDTMANN

# BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

## SECTION

HTML FUNDAMENTALS

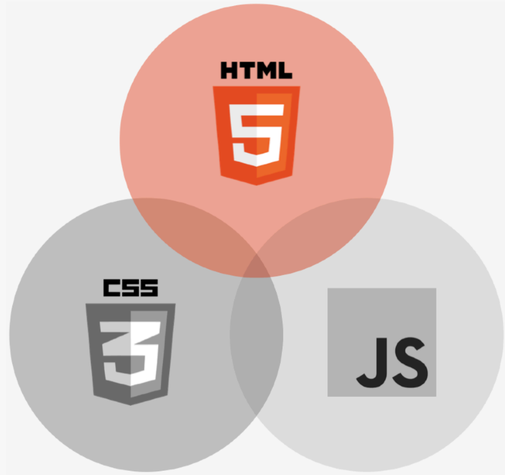
## LECTURE

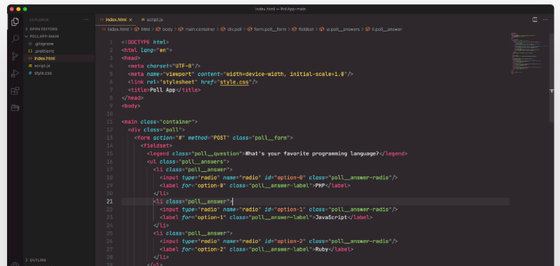
INTRODUCTION TO HTML

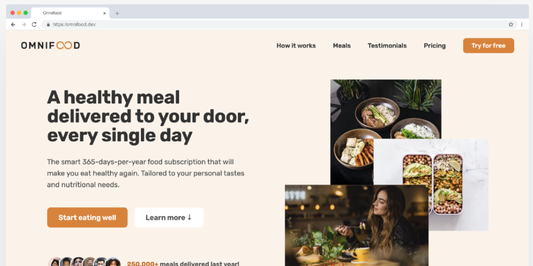
# WHAT IS **HTML**?

## HTML

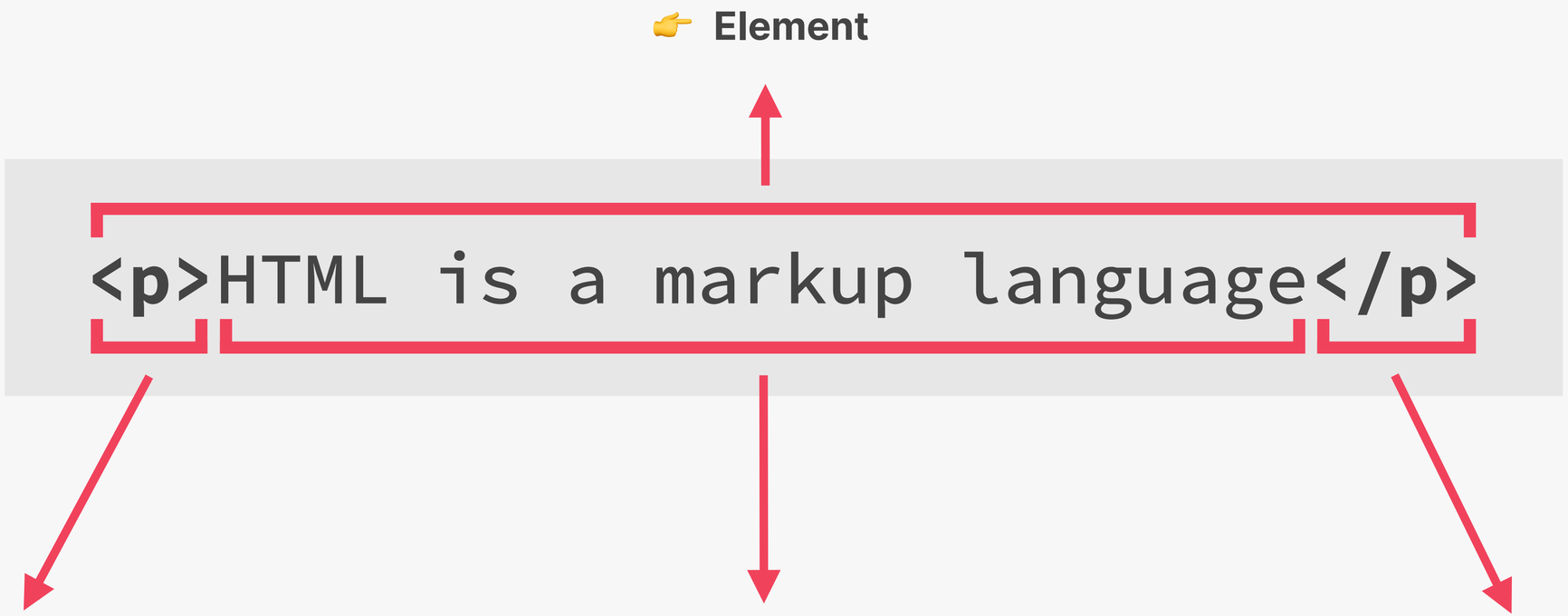
- 👉 **H**yper**T**ext **M**arkup **L**anguage
- 👉 HTML is a markup language that web developers use to **structure and describe the content** of a webpage (*not a programming language*)
- 👉 HTML consists of **elements** that describe different types of content: paragraphs, links, headings, images, video,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

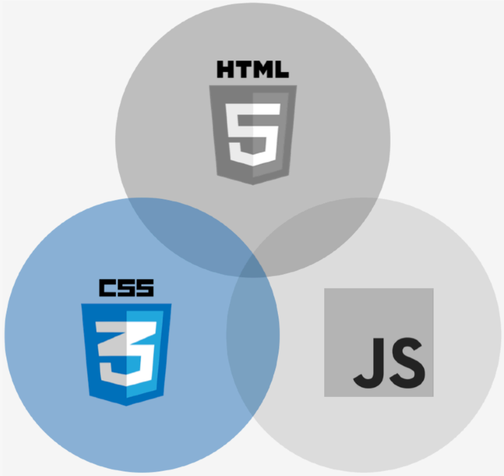
## LECTURE

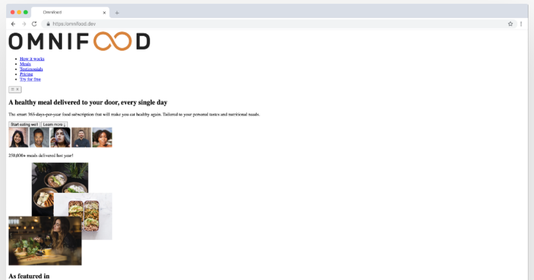
INTRODUCTION TO CSS

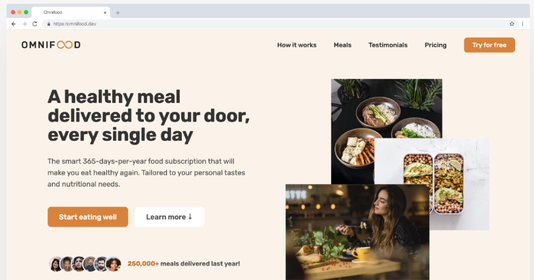
# WHAT IS **CSS**?

## 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.

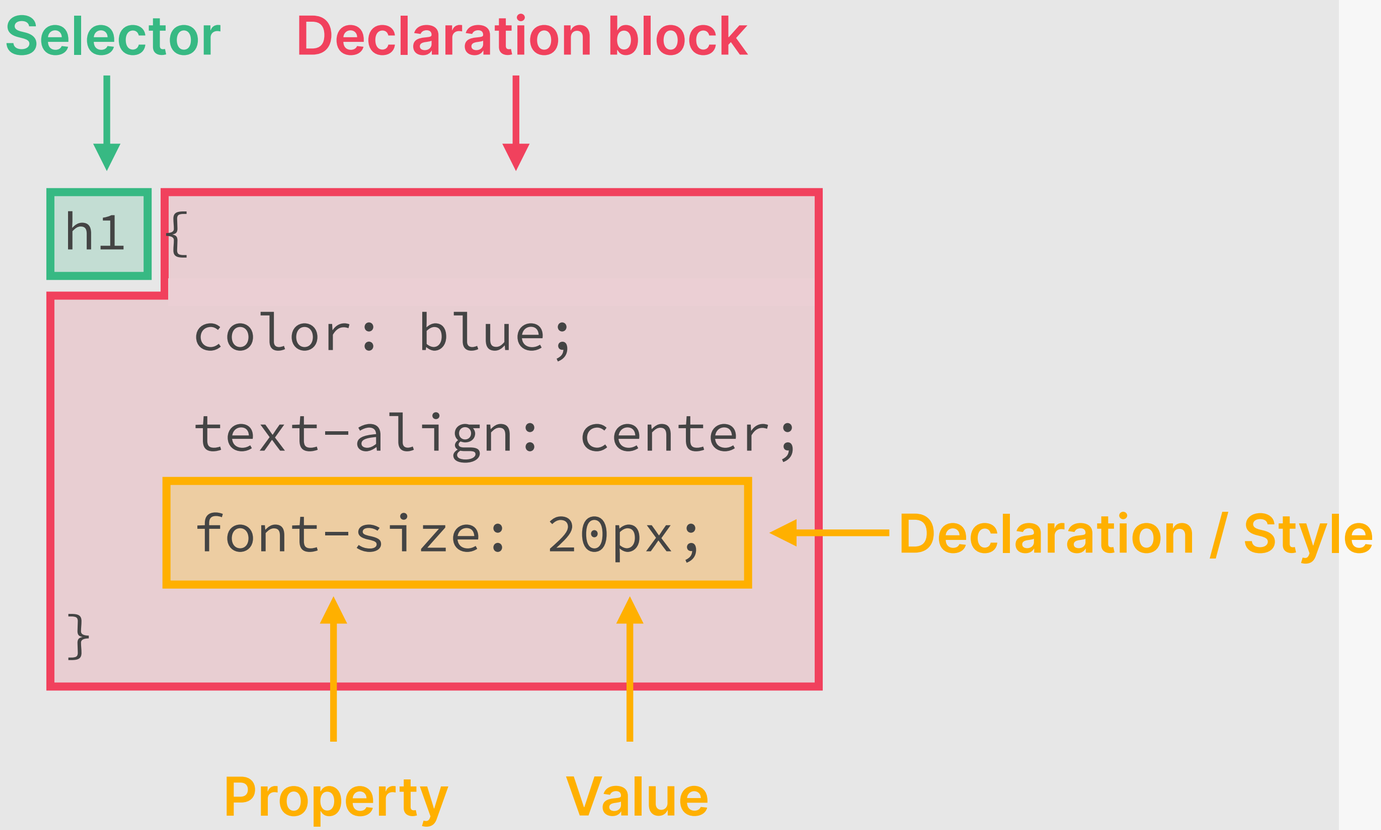


# HOW WE **SELECT** AND **STYLE** ELEMENTS

## A CSS RULE



JONAS.IO
SCHMEDTMANN

# BUILD RESPONSIVE REAL-WORLD WEBSITES WITH HTML AND CSS

@JONASSCHMEDTMAN

## SECTION

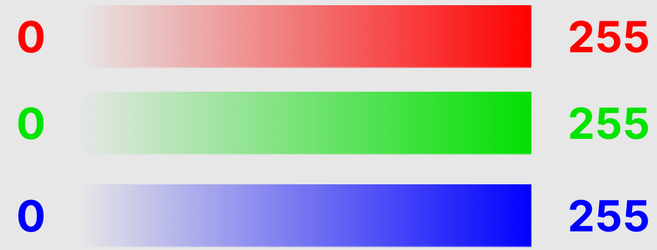
CSS FUNDAMENTALS

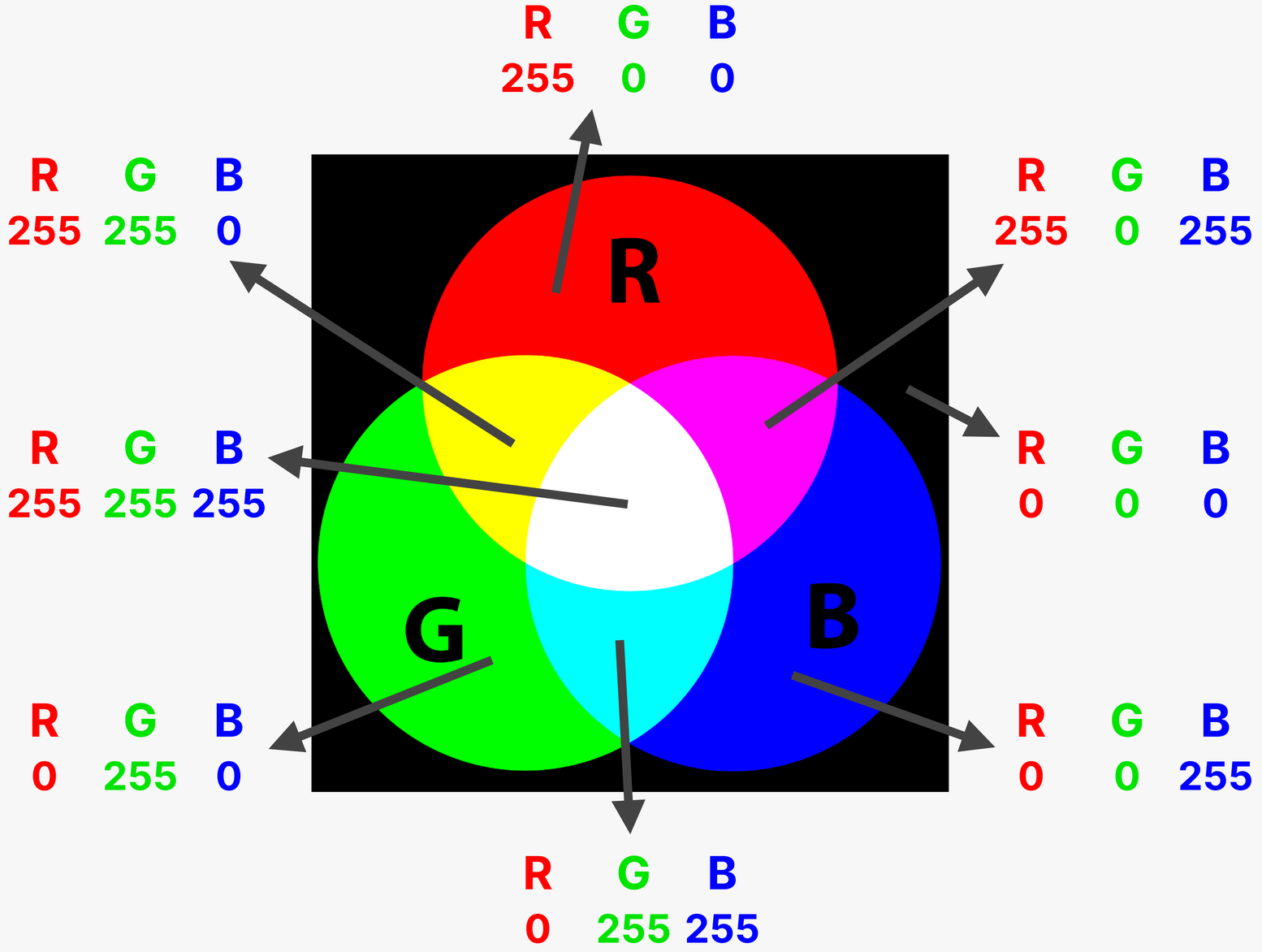
## LECTURE

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

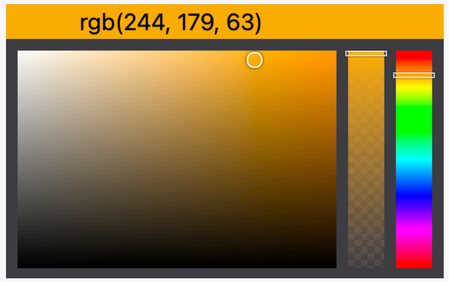
👉 Instead of using a scale from 0 to 255, we go from **0 to ff** (255 in hexadecimal numbers)

```
#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;
}
```

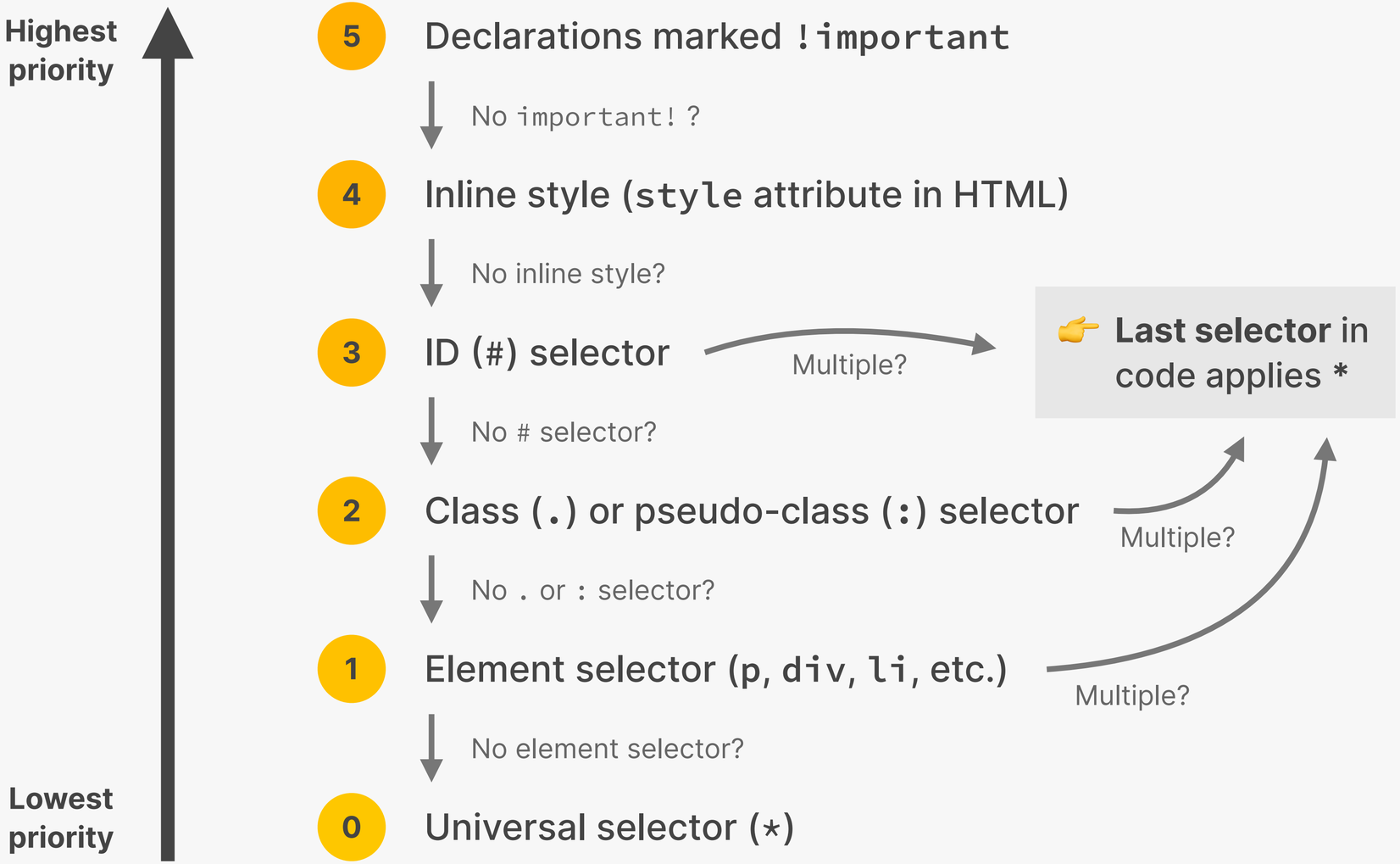
🤔 There are **multiple selectors** selecting the same element. **Which one of them applies?**

🤓 ***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

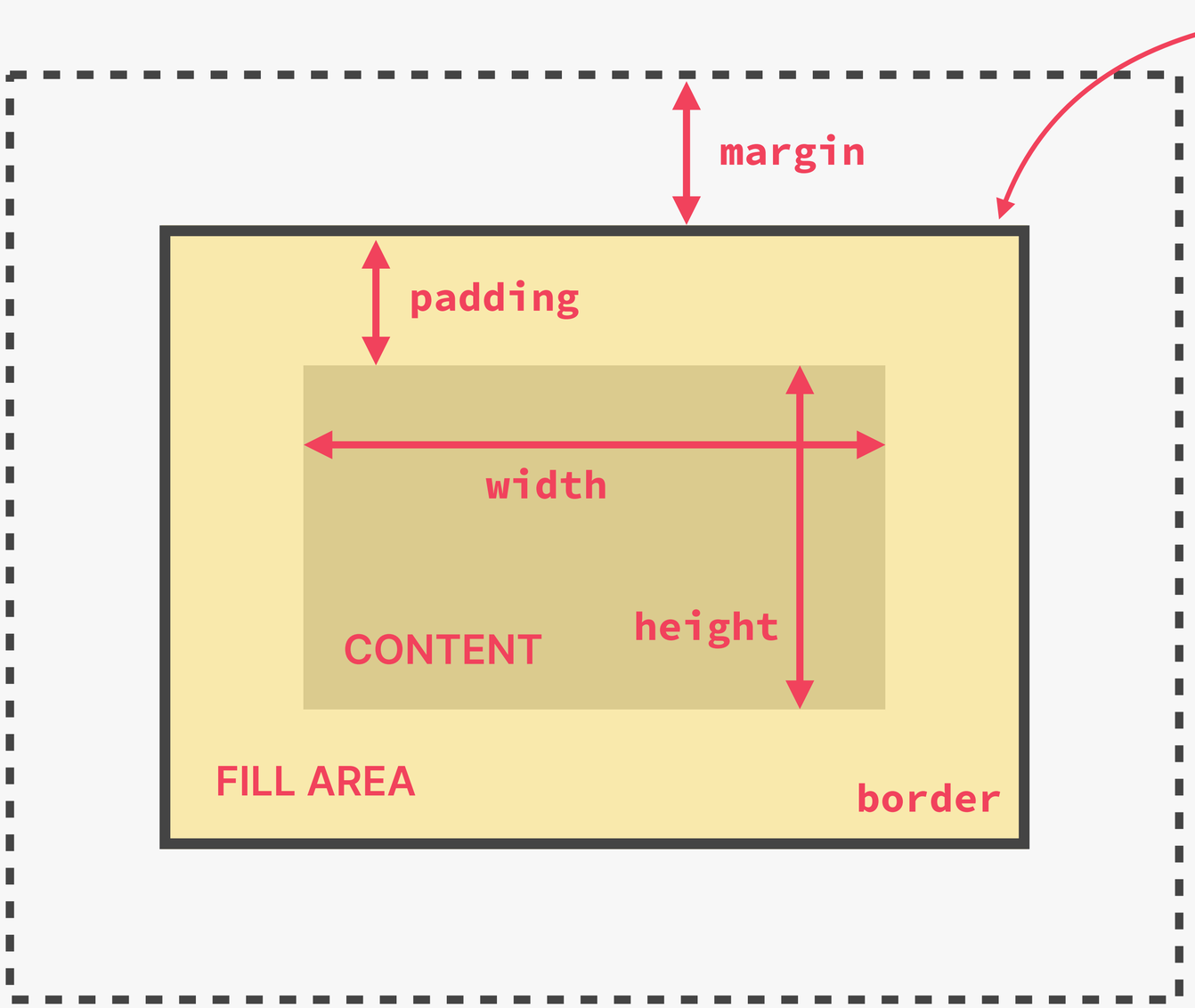
## SECTION

CSS FUNDAMENTALS

## LECTURE

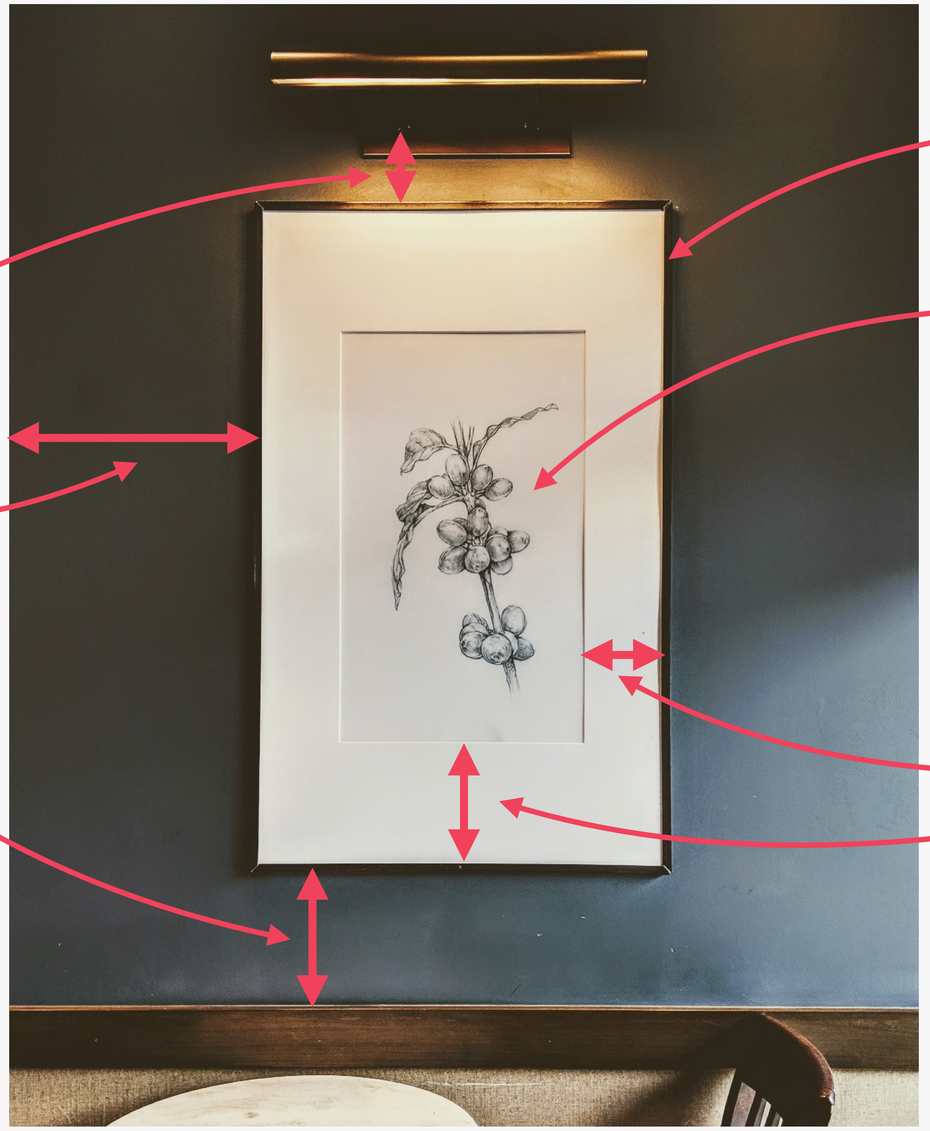
CSS THEORY #3: THE CSS BOX MODEL

# 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

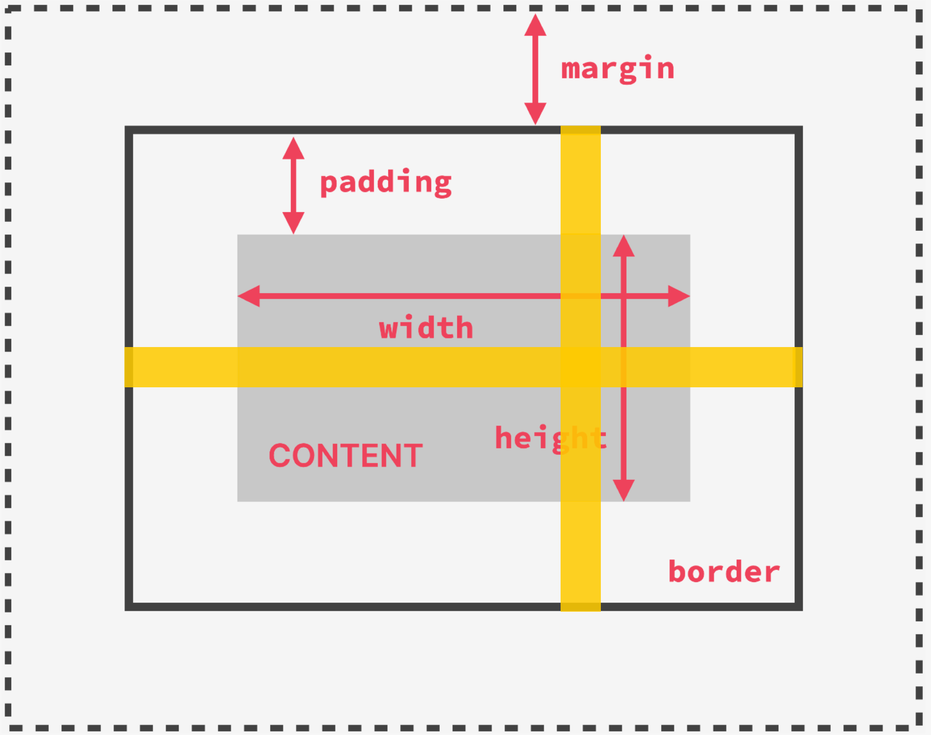


# ELEMENT **HEIGHT** AND **WIDTH** CALCULATION

**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

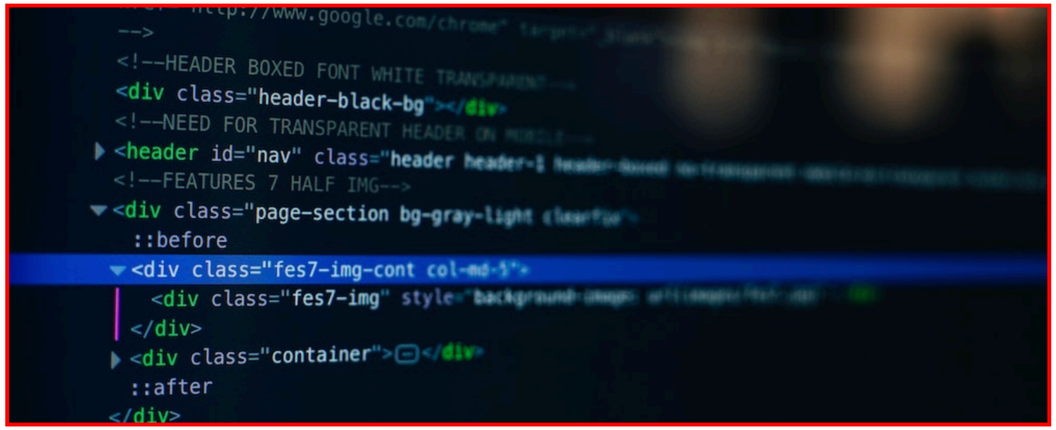
- 👉 Occupies only the space **necessary for its content**
- 👉 Causes **no line-breaks** after or before the element
- 👉 Box model applies in a different way: **heights and widths do not apply**
- 👉 **Paddings and margins** are applied **only horizontally** (left and right)

**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

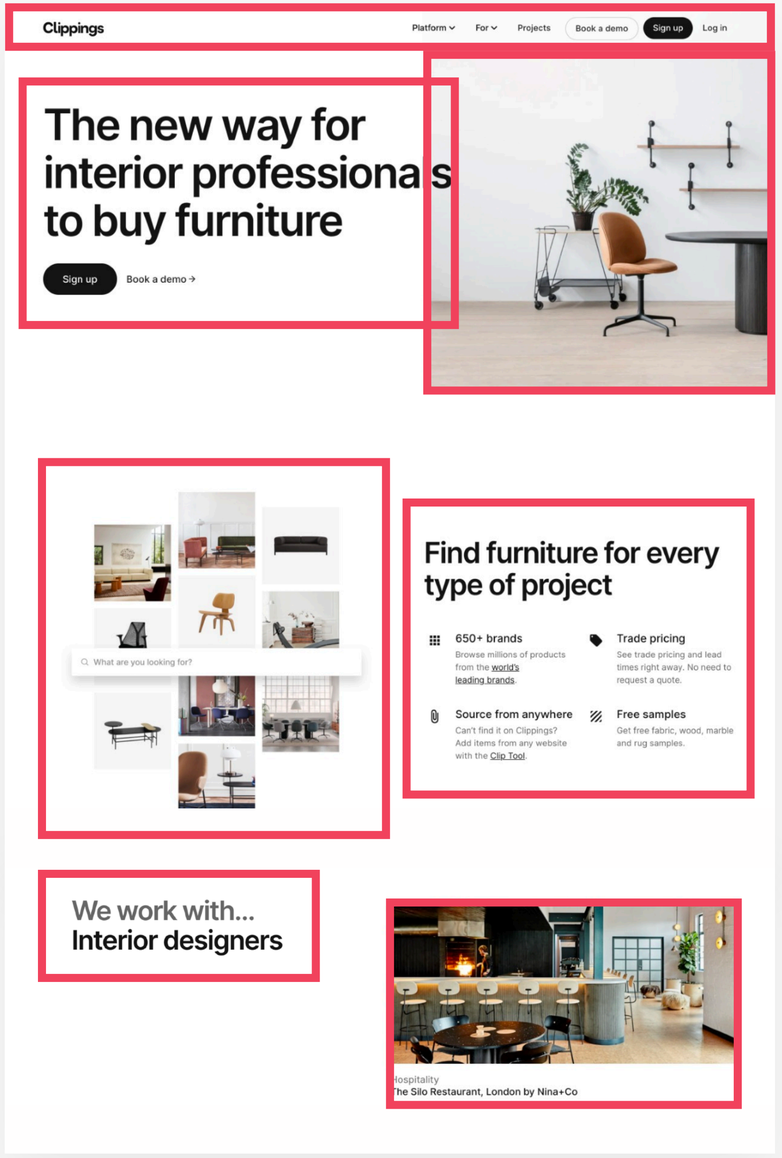
## LECTURE

THE 3 WAYS OF BUILDING LAYOUTS

# WHAT DOES “**LAYOUT**” MEAN?

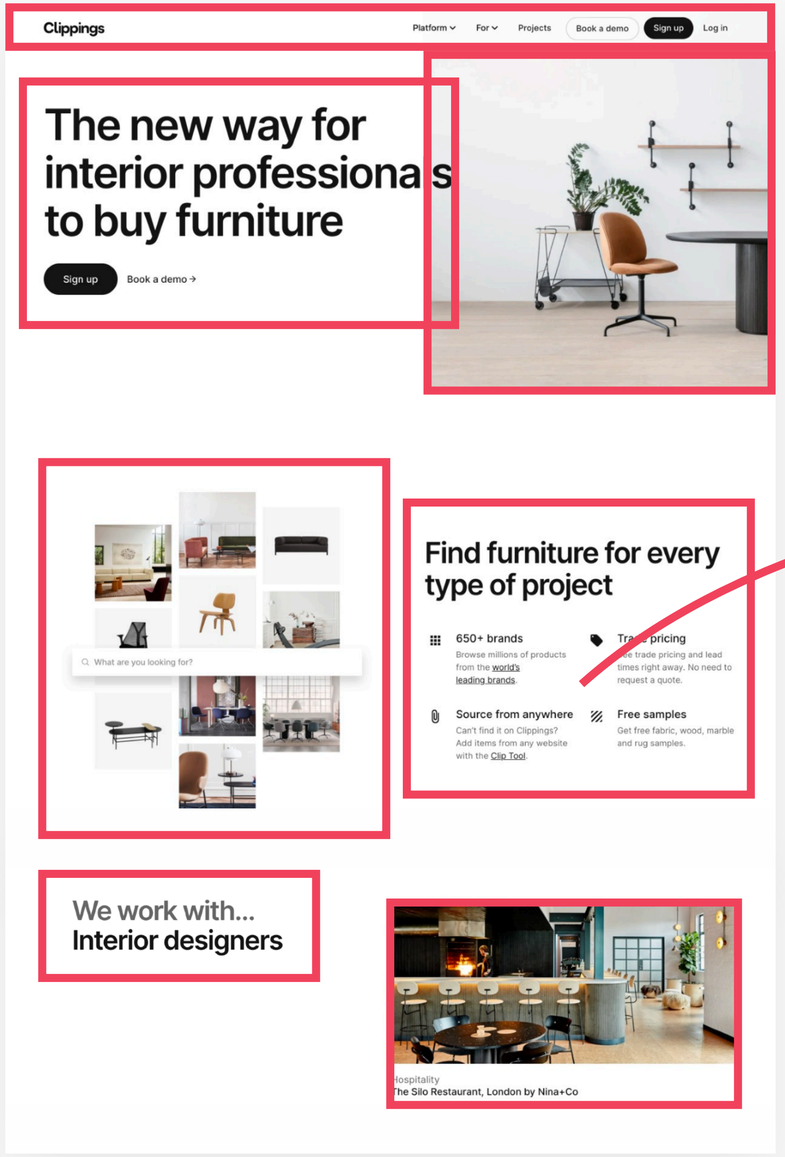
## LAYOUT

- 👉 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.

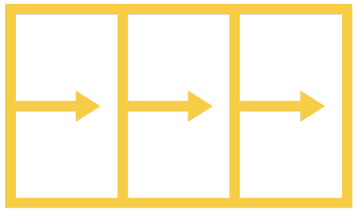
Free samples

Get free fabric, wood, marble and rug samples.

**COMPONENT LAYOUT**

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

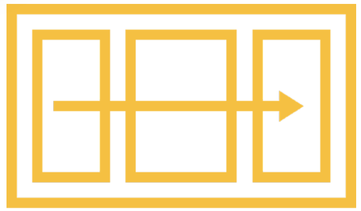
1



## FLOAT LAYOUTS

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

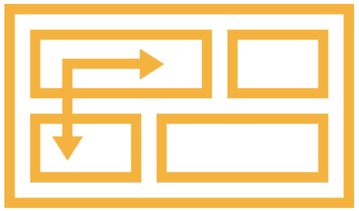
2



## 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

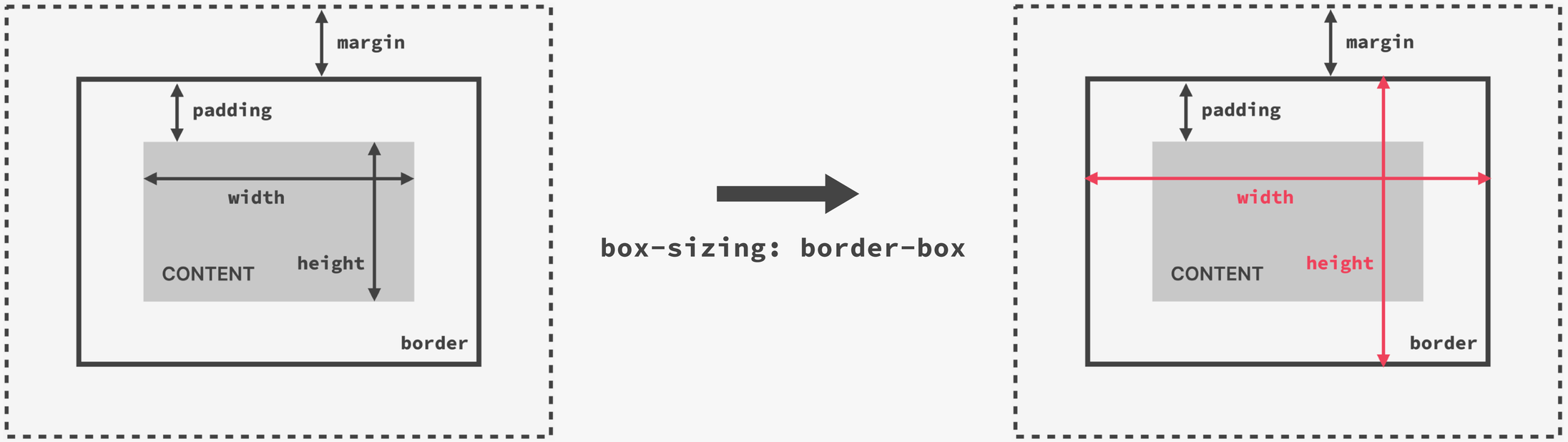
## SECTION

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~~