# Winter School 2022

## Bergische Universität Wuppertal

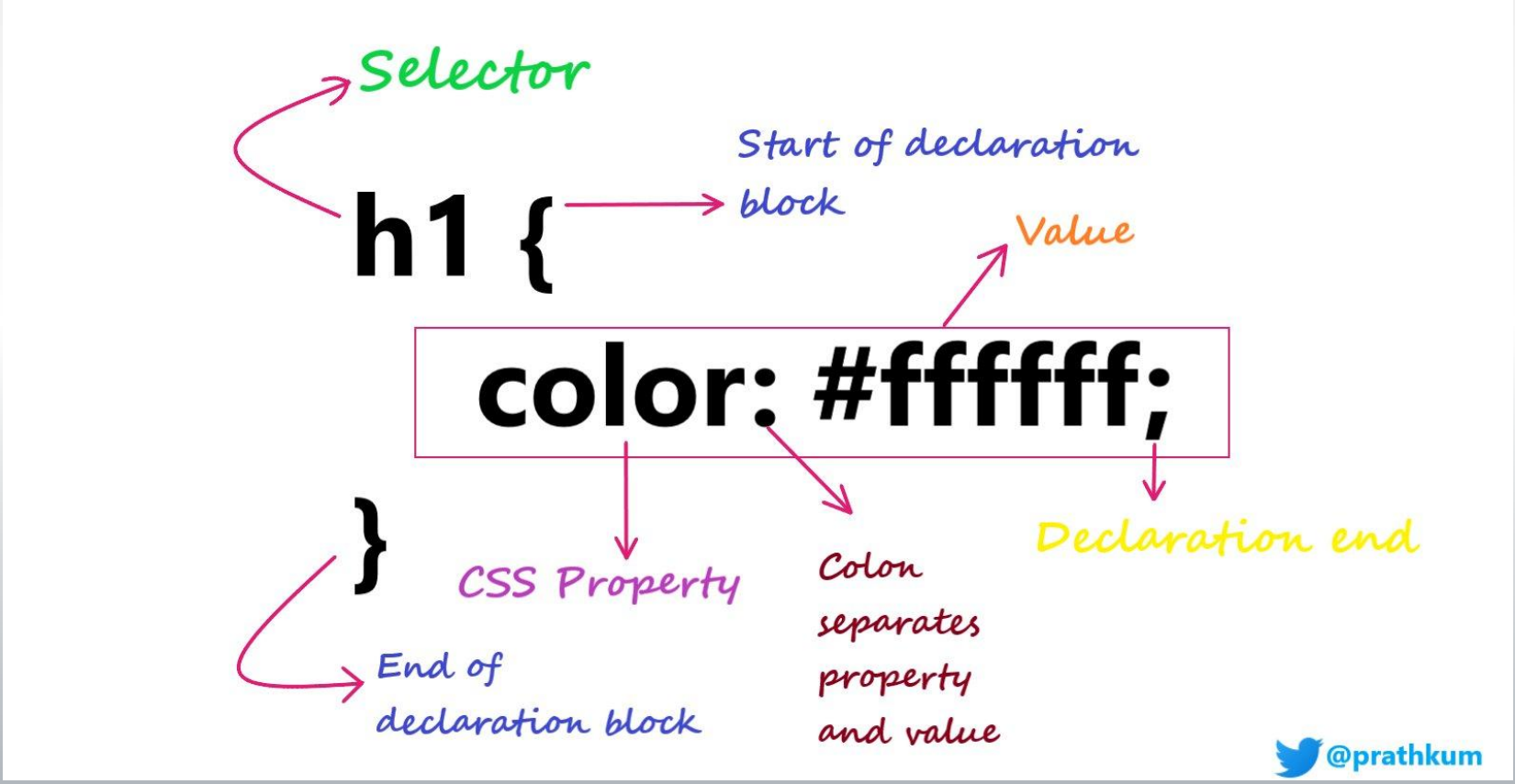
21.03 – 25.03.2022

Einführung in
**HTML & CSS**

Nadine Sutor & Andreas Mertgens

#wsde22

**Cascading Stylesheets:** Zur Erstellung von Gestaltungsanweisungen



# HTML und CSS - Einführung

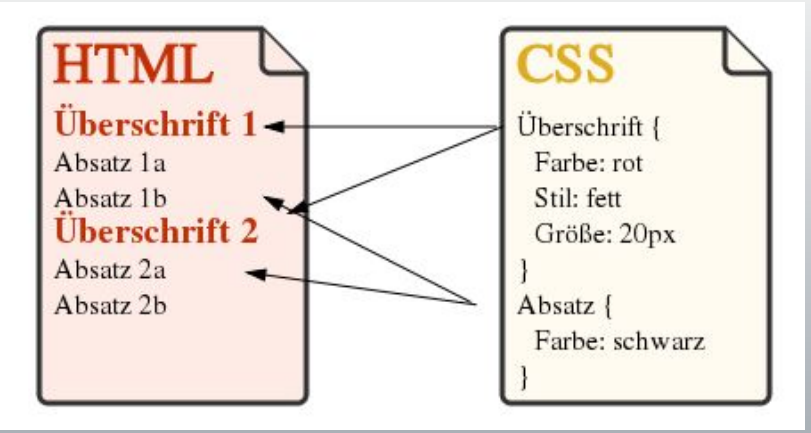

- **diese Gestaltungsanweisungen** können auf unterschiedlichen Wegen mit den Inhalten eines HTML-Dokuments **verknüpft** werden
- Weiterentwickelt und gepflegt vom *W3C*
- WICHTIG: Trennung!
  - der *äußeren Darstellung* (Layout, Farben und Typografie) von den
  - *Inhalten* (Gliederung des Dokuments und die Bedeutung seiner Teile)

- Auszeichnungssprache HTML dient dazu einen Text zu ***strukturieren***, nicht aber zu ***formatieren***
- Die visuelle Darstellung ist nicht Teil der HTML-Spezifikation

**Konzepte:** Selektoren, IDs, Klassen

- CSS verfügt über eine eigene Syntax, die man aber schnell lernen kann (z.B. mit dem Tutorial auf w3schools)
- Um Inhalte aus HTML-Dokumenten zu “stylen” muss eine Referenz hergestellt werden
- Elemente mittels bestimmter Anweisungen “ansprechen”

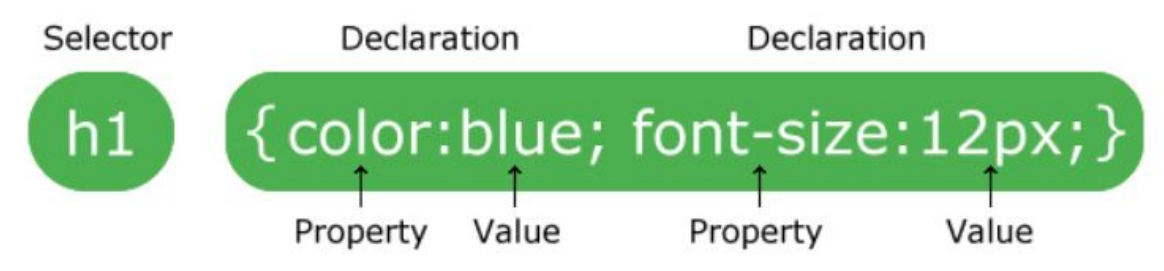
  → **selector** (HTML-Element)**, #id, .class**



**Konzepte:** Selektoren, IDs, Klassen

**Syntax**

- Der ***selector*** referenziert das HTML-Element das formatiert werden soll
- Es folgt die ***declaration*****.** Sie kann eine oder mehrere Gestaltungsanweisungen enthalten
- Jede Deklaration beinhaltet eine CSS-Eigenschaft (*property*) und einen Wert (*value*), verbunden durch einen Doppelpunkt
- Mehrere CSS-Deklarationen werden durch Semikolon voneinander getrennt
- Die vollständige Deklaration wird umschlossen von **{curly braces}**



## **Konzepte:** Selektoren, IDs, Klassen

- Selektoren werden genutzt um HTML-Elemente zu *finden* und im Stylesheet *referenzieren* um sie dann zu *formatieren*
- Unterschiedliche Selektor-Kategorien:
  - CSS element selector (*selects HTML elements based on the **element name***)
  - CSS id selector (*uses the **id** attribute of an HTML element to select a specific element*)
  - CSS class selector (*selects HTML elements with a specific **class** attribute*)
  - CSS universal selector (*selects **all HTML elements** on the page*)
  - CSS grouping selector (*selects all HTML elements with the **same style definitions)***

1. CSS element selector (*selects HTML elements based on the **element name***)

```
p {
  text-align: center;
  color: red;
}
```

Mit dieser CSS-Anweisung werden alle <p>-Elemente zentriert und der Textinhalt in einer roten Farbe dargestellt.

2. CSS id selector (*uses the **id** attribute of an HTML element to select a specific element*)

```
#para1 {
  text-align: center;
  color: red;
}
```

```
<div id="para1">
    ...
</div>
```

Die CSS-Regel wird nur auf diejenigen HTML-Elemente angewendet, die eine ID mit dem Wert “para1” haben.  D.h. eine CSS-Anweisung soll nur für ein bestimmtes Element gelten. Um das entsprechende Element mit dieser ID zu “stylen”, schreibt man in CSS ein **“#”**

**Wichtig:** eine ID darf nicht mit einer Zahl beginnen!

3. CSS class selector (*selects HTML elements with a specific **class** attribute*)

```
p.center {
  text-align: center;
  color: red;
}
```

```
<p class="center">
    ...
</p>
```

Der class-selector referenziert alle HTML-Elemente mit einem class-Attribut. Ausschlaggebend ist der Wert des class-Attributs. Um diese Elemente mittels CSS zu stylen, muss ein Punkt (.) geschrieben werden, gefolgt vom Namen, bzw. Wert (center) der Klasse.

4. CSS universal selector (*selects **all HTML elements** on the page*)

```
* {
  text-align: center;
  color: blue;
}
```

Diese CSS-Anweisung greift für alle Elemente in einem HTML-Dokument

5. CSS grouping selector (*selects all HTML elements with the **same style** definitions)*

```
h1 {
  text-align: center;
  color: red;
}

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

**vs.**

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

Durch Kommata getrennt
→ weniger Code

## HTML & CSS: Workflow und Priorisierung

- CSS-Anweisungen *inline* innerhalb der Textstruktur des HTML-Dokuments schreiben

  → Element/Attribut-Kombination **`<span style="..."/>`**

- CSS-Anweisung im HTML-Dokument schreiben
  (in das `style`-Element im `head` )

  → **`selektor {eigenschaft:wert;}`**

- CSS-Anweisungen in einem externen Stylesheet schreiben und via Link in ein HTML-Dokument einbinden

  → **`<link rel="stylesheet" href="my-first.css"/>`**

- CSS-Anweisungen (als default) im Browser

# F R A G E N ?

**Weitere CSS-Eigenschaften** die oft und gerne genutzt werden (Auswahl)

- color
- background-color
- background-image
- border
- text-align
- text-decoration
- font-family
- font-size
- font-style
- font-weight
- padding/margin ("box-model")
  - top
  - right
  - bottom
  - left
- float
  - left
  - right
- position
  - static
  - relative
  - fixed
  - absolute
  - sticky
- display
  - none
  - inline
  - block
- overflow
  - visible
  - hidden
  - scroll
- z-index
- opacity
- linear-gradient

https://www.w3schools.com/css/

# Exkurs:
# CSS Grid

**Seitenstrukturen und Layouts mit CSS**

CSS property `"display"`.

Jedes HTML Element hat einen Default Wert:

`"block"` oder `"inline"` (bekannt aus HTML Folien)

Default Werte können überschrieben werden.

`inline`: kein Whitespace, ignoriert `width` und `height`

`block`: vertikal ausgerichtet, default: `width = 100%`

**Seitenstrukturen und Layouts mit CSS**

Klassische Wege: Layout basteln aus Block und Inline Elementen, float und absoluten Positionswerten

oder als Tabelle

- fehleranfällig
- unübersichtlicher CSS Code
- wenig responsiv

**Seitenstrukturen und Layouts mit CSS**

Layout entweder über externe Frameworks

oder

**Native Lösungen in CSS 3:**

Flexbox (`display: flex`)

W3C's candidate recommendation (CR) (2018)

https://www.w3.org/TR/css-flexbox-1/

Grid (`display: grid`)

W3C's candidate recommendation (CR) (2020)

https://www.w3.org/TR/css-grid-1/

Beide Ansätze werden von alle modernen Browsern (Versionen seit 2017) unterstützt.

## Seitenstrukturen und Layouts mit CSS

Flexbox (`display: flex`)

Anordnung entweder horizontal **oder** vertikal

- `flex-flow: row`

| 1 | 2 | 3 | 4 |
|---|---|---|---|

- `flex-flow: column`

| 1 |
|---|
| 2 |
| 3 |
| 4 |

zusätzliche Optionen wie wrap oder reverse

- `flex-flow: row wrap`

| 1 | 2 | 3 |
|---|---|---|
| 4 | | |

- `flex-flow: column reverse`

| 4 |
|---|
| 3 |
| 2 |
| 1 |

## Seitenstrukturen und Layouts mit CSS

Grid (`display: grid`)

Anordnung immer als Raster von konfigurierbaren *rows* und *columns*

z.B.

- `grid-template-columns: 1fr 2fr 1fr;`
- `grid-template-rows: 50% 30% 20%;`

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 8 | 8 | 9 |

## CSS Grid

```
<div class="container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    …
</div>
```

```
.container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    grid-template-rows: 50% 30% 20%;
    grid-gap: 1em;
```

-> Alle **child Elemente** werden als *CSS Grid* dargestellt

-> Definiert das Muster für Spalten
Hier: 3 Spalten. Verteilung nach Fractions (fr)

-> Definiert das Muster für Zeilen
Hier: 3 Zeilen mit Prozentwerten 50% 30% 20%

-> Abstand zwischen den einzelnen Grid Elementen

Konfiguration des Grids im **Container/Parent** Element -> bestimmt Darstellung der **child Elemente**

# -> Demo: Grid01.html

## CSS Grid

```
<div class="container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    …
</div>
```

```
.container {
    display: grid;

    grid-template-columns: 1fr 2fr 1fr;
```

Neben Angaben wie Pixel, fr, oder % kann man auch komplexere Ausdrücke verwenden:

`grid-template-columns: repeat(5, 200px)` -> 5 Spalten mit jeweils 200px Breite

`grid-template-columns: minmax(200px, 1fr)` -> Min 200px, Max 30%.

Auch möglich:

`grid-auto-rows: 200px;` Erzeugt automatisch (je nach Platz) neue Zeilen

## CSS Grid

```
<div class="container">
    <div id="header">1</div>
    <div>2</div>
    <div>3</div>
    …
</div>
```

Einzelne Elemente in einem Grid können auch individuell formatiert werden.
z.B. Header der alle 3 Spalten eines Layouts abdeckt.

```
.container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    }
#header {
    grid-column: 1/4
```

-> 1/4 bedeutet:
Element beginnt in column 1,
erstreckt sich über 3 columns,
bis zum Beginn von column **4**

```
    }
```

| Header | | |
|---|---|---|
| 2 | 3 | 4 |
| 2 | 3 | 4 |

# -> Demo: Grid02.html

## CSS Grid

## Ausblick

- Grid Elemente können gleichzeitig auch wieder Grid Container sein. -> Nested Grids
- Elemente können innerhalb von Grids ausgerichtet werden -> alignment
- Elemente können Namen/IDs haben und dann frei im Grid positioniert werden
- Responsive Layouts für verschiedene Endgeräte

## CSS Grid

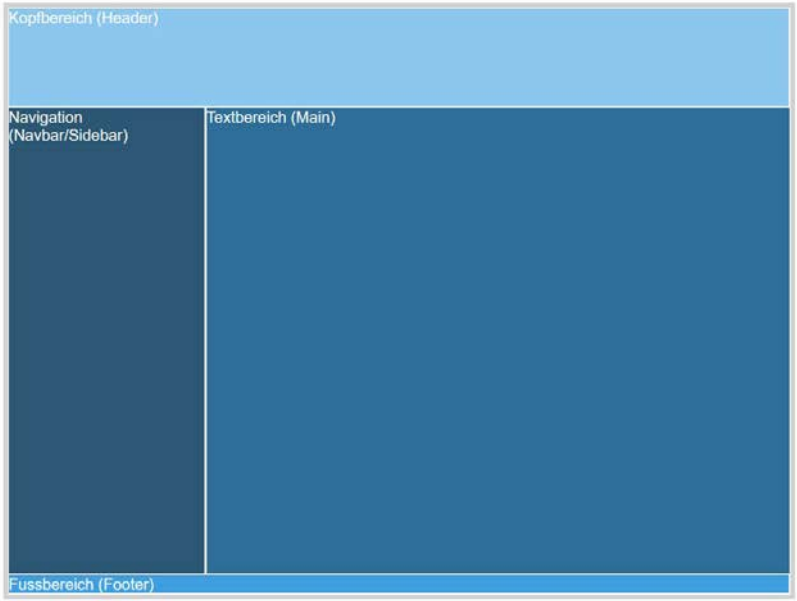
Weiterführende Tutorials/Referenzen zu **CSS Grid:**

https://www.w3schools.com/css/css_grid.asp
https://developer.mozilla.org/de/docs/Web/CSS/CSS_Grid_Layout

# H A N D S - O N

# 1. Erstellung eines einfachen Layouts

a. Kopfbereich
b. Navigation
c. Textbereich
d. Footer



# 2. Erstellung einer *Landing Page*

a. Einbau von Bildern
b. Einbau einer Navigationsleiste (sidebar und/oder navbar)
c. Einbau von Menüs (drop-downs)
d. Integration von Text
e. Einbau von Hyperlinks

# Vielen Dank für Ihre Aufmerksamkeit!

BERGISCHE
UNIVERSITÄT
WUPPERTAL

BERGISCHE
UNIVERSITÄT
WUPPERTAL