Peter Wolf
<Software Developer & Advisor />

HTML5 Advanced Course

Unterrichtsinhalt

visual studio code

open in browser extension installation

HTML Session

Aufbau und Struktur

<script> attribute loading

Attribute	Loading Behavior	Execution Timing	Best For
`async`	Loads asynchronously	Executes as soon as loaded, before document parsing completes	Independent scripts, like analytics
`defer`	Loads asynchronously	Executes after full HTML parsing is done	Scripts relying on full DOM

If no **async** or **defer** is specified, scripts are loaded and executed immediately as they appear in the HTML, which **can block the page rendering**.

To improve performance, especially for non-critical scripts, place <script> tags at the end of the <body> or use async and defer in the <head>.

```html
<head>
    <!-- Character Encoding -->
    <meta charset="UTF-8">

    <!-- Viewport for Responsive Design -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- SEO Meta Tags -->
    <title>Your Page Title Here</title>
    <meta name="description" content="A brief description of your page's content."
    <meta name="keywords" content="keyword1, keyword2, keyword3">

    <!-- Author (optional) -->
    <meta name="author" content="Your Name or Company Name">

    <!-- Favicon -->
    <link rel="icon" href="favicon.ico" type="image/x-icon">

    <!-- Open Graph (for social media sharing) -->
    <meta property="og:title" content="Your Page Title Here">
    <meta property="og:description" content="A brief description for social sharin
    <meta property="og:image" content="https://example.com/image.jpg">
    <meta property="og:url" content="https://example.com/">
    <meta property="og:type" content="website">

    <!-- Twitter Card (for Twitter sharing) -->
    <meta name="twitter:card" content="summary_large_image">
    <meta name="twitter:title" content="Your Page Title Here">
    <meta name="twitter:description" content="A brief description for Twitter shar
    <meta name="twitter:image" content="https://example.com/image.jpg">

    <!-- Stylesheets -->
    <link rel="stylesheet" href="styles.css">

    <!-- Preconnect for External Resources (Performance Optimization) -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

    <!-- Fonts and Icons (example) -->
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&displ

    <!-- JavaScript -->
    <script defer src="script.js"></script> <!-- Use `defer` to load JavaScript af
</head>
```

Character Encoding (<meta charset="UTF-8">): Ensures text is displayed correctly.

- **Viewport** (<meta name="viewport" content="width=device-width, initial-scale=1.0">): Sets up responsive scaling for mobile devices.
- **SEO Meta Tags** (title, description, keywords): Improve search engine indexing and appearance in search results.
- **Open Graph and Twitter Cards**: Enable rich previews when shared on social media platforms like Facebook and Twitter.
- **Favicon**: Defines the icon that appears in browser tabs.
- **Stylesheets** (<link rel="stylesheet" href="styles.css">): Loads your CSS.
- **Preconnect**: Speeds up loading of external resources like Google Fonts by pre-establishing connections.
- **Script Loading** (<script defer src="script.js">): Using defer ensures JavaScript runs only after HTML parsing completes.

Semantik structure

Beispiele für Semantische HTML Struktur

<article> => artikel / post

<aside> => sidebar navigation

<summary> => defines a visible headline for <details>
<details> => hidden details like in accordion

<figcaption> => defines a caption for a <figure>
<figure>

<footer> => defines footer
<header> => defines header
<main> => defines main content
<mark> => defines highlighted / not blockquote
<nav> => defines a navigaiton block
<section> => defines a section block

<time> => defines a date / time

Peter Wolf
<Software Developer & Advisor />

HTML Multilanguage

General language settings

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="language" content="en">
```

alternative content sources

```
<head>
    <link rel="alternate" hreflang="es" href="https://example.com/es/" />
    <link rel="alternate" hreflang="fr" href="https://example.com/fr/" />
</head>
```

multilanguage in one document

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Multilingual Example</title>
</head>
<body>
    <h1>Welcome to the Website</h1>

    <p lang="es">¡Hola, bienvenido al sitio web!</p> <!-- Spanish -->
    <p lang="fr">Bonjour, bienvenue sur le site Web!</p> <!-- French -->
    <p lang="de">Hallo, willkommen auf der Website!</p> <!-- German -->
</body>
</html>
```

RTL settings

```
<!DOCTYPE html>
<html lang="ar" dir="rtl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>موقع ويب متعدد اللغات</title> <!-- Arabic title -->
</head>
<body>
    <h1>مرحبا بك في الموقع!</h1> <!-- Arabic greeting -->
</body>
</html>
```

Peter Wolf
<Software Developer & Advisor />

Image tag

Use an image and put some variations in it for different width values

```
<img src="image-320w.jpg"
     srcset="image-320w.jpg 320w,
             image-480w.jpg 480w,
             image-768w.jpg 768w,
             image-1024w.jpg 1024w"
     sizes="(max-width: 600px) 100vw,
            (max-width: 900px) 50vw,
            (max-width: 1200px) 33vw,
            25vw"
     alt="Responsive image example">
```

VIDEO and AUDIO

```
<video width="640" height="360" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.webm" type="video/webm">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

- The width and height attributes specify the size of the video.
- The controls attribute adds playback controls to the video.
- The <source> elements allow you to provide multiple formats of the video, ensuring compatibility across browsers. If the first format is not supported, the browser will try the next one.
- The text inside the <video> tag ("Your browser does not support the video tag.") will be displayed if the browser doesn't support the <video> element.

```
<audio controls>
  <source src="audio.mp3" type="audio/mp3">
  <source src="audio.ogg" type="audio/ogg">
  <source src="audio.wav" type="audio/wav">
  Your browser does not support the audio element.
</audio>
```

- The controls attribute adds playback controls to the audio player.
- The <source> elements provide multiple formats of the audio file for broader browser compatibility.
- The text inside the <audio> tag ("Your browser does not support the audio element.") will be shown if the browser doesn't support the <audio> element.

Key Points:

- **Formats**: Common video formats include .mp4, .webm, and .ogg. Common audio formats include .mp3, .ogg, and .wav.
- **Autoplay**: You can use the autoplay attribute to make the video or audio start playing automatically when the page loads. Example: <video autoplay controls>.
- **Looping**: You can use the loop attribute to make the video or audio loop continuously. Example: <audio loop controls>.

Peter Wolf
<Software Developer & Advisor />

Different File Types and usages

1. MP4 (MPEG-4 Part 14)

- Description: MP4 is one of the most popular video file formats, commonly used for streaming and storing video content. It supports both video and audio, and often uses H.264 video compression, making it widely compatible with most devices and browsers. It's known for a good balance between quality and file size.

- Usage: Video streaming, online media, YouTube, and video storage.

2. WebM

- Description : WebM is an open-source video format designed for the web, optimized for high-quality video with a smaller file size. It typically uses the VP8 or VP9 video codec and the Vorbis or Opus audio codec. It's supported by modern browsers like Chrome, Firefox, and Opera but has less widespread compatibility compared to MP4.

- Usage: HTML5 video, web-based video applications, open-source projects.

3. OGG (Ogg Vorbis)

- Type : Audio format

- Description : OGG is a free, open-source container format often used for audio compression. The most common audio codec used with OGG is Vorbis, which provides efficient compression without a significant loss in sound quality. OGG is supported in many media players and web applications, but it’s less popular than MP3.

- Usage : Music streaming, games, and multimedia applications.

4. MP3 (MPEG Audio Layer III)

- Type : Audio format

- Description : MP3 is one of the most widely used audio formats, known for compressing audio files while retaining good sound quality. It uses lossy compression to reduce file size, making it ideal for music storage and streaming. MP3 is supported by nearly all devices and media players.

- Usage : Music files, podcasts, audio streaming, and digital downloads.

5. WAV (Waveform Audio File Format)

- Type : Audio format

- Description : WAV is an uncompressed audio format, which means it retains the original sound quality without any loss. This results in large file sizes compared to MP3 or OGG. WAV files are widely used in professional audio recording, editing, and sound design due to their high quality.

- Usage : Professional audio production, sound editing, archival storage.

MarkUp vs structured data

structured data example with jsonLD:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Person Structured Data</title>
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "Person",
  "name": "John Doe",
  "birthDate": "1990-05-15",
  "address": {
    "@type": "PostalAddress",
    "streetAddress": "123 Main Street",
    "addressLocality": "Springfield",
    "addressRegion": "IL",
    "postalCode": "62701",
    "addressCountry": "US"
  },
  "worksFor": {
    "@type": "Organization",
    "name": "Example Corp"
  },
  "telephone": "+1-555-555-5555"
}
</script>
</head>
<body>
<h1>Person Information</h1>
<p><strong>Name:</strong> John Doe</p>
<p><strong>Birthdate:</strong> May 15, 1990</p>
<p><strong>Address:</strong> 123 Main Street, Springfield, IL, 62701, US</p>
<p><strong>Company:</strong> Example Corp</p>
<p><strong>Phone:</strong> +1-555-555-5555</p>
</body>
</html>
```

MarkUp

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Person Structured Data</title>
</head>
<body>
<div itemscope itemtype="https://schema.org/Person">
  <h1>Person Information</h1>
  <p><strong>Name:</strong> <span itemprop="name">John Doe</span></p>
  <p><strong>Birthdate:</strong> <span itemprop="birthDate">1990-05-15</span></p>

  <div itemprop="address" itemscope itemtype="https://schema.org/PostalAddress">
    <p><strong>Address:</strong>
      <span itemprop="streetAddress">123 Main Street</span>,
      <span itemprop="addressLocality">Springfield</span>,
      <span itemprop="addressRegion">IL</span>,
      <span itemprop="postalCode">62701</span>,
      <span itemprop="addressCountry">US</span>
    </p>
  </div>

  <p><strong>Company:</strong> <span itemprop="worksFor" itemscope itemtype="https
    <span itemprop="name">Example Corp</span>
  </span></p>

  <p><strong>Phone:</strong> <span itemprop="telephone">+1-555-555-5555</span></p>
</div>
</body>
</html>
```

Peter Wolf

<Software Developer & Advisor />

example blogarticle

```
<!-- JSON-LD Structured Data -->
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "Article",
  "headline": "How to Use Structured Data in HTML",
  "author": {
    "@type": "Person",
    "name": "John Doe"
  },
  "publisher": {
    "@type": "Organization",
    "name": "Web Tutorials",
    "logo": {
      "@type": "ImageObject",
      "url": "https://example.com/logo.png"
    }
  },
  "datePublished": "2024-11-15",
  "dateModified": "2024-11-15",
  "mainEntityOfPage": {
    "@type": "WebPage",
    "@id": "https://example.com/articles/structured-data"
  },
  "image": "https://example.com/articles/structured-data-image.jpg",
  "description": "An informative article about structured data and block article
  "articleBody": "Structured data helps search engines understand the content of
}
```

CSS Session

CSS CALC()

Example 1 for WIDTHS

```
<style>
    .container {
        width: 100%;
        display: flex;
    }

    .sidebar {
        width: 200px;
        background-color: #f4f4f4;
    }

    .main-content {
        width: calc(100% - 200px); /* Calculate remaining width */
        background-color: #e0e0e0;
        padding: 20px;
    }
</style>
```

Peter Wolf
<Software Developer & Advisor />

example 2 for centering element with calc

```
<style>
    .box {
        width: 300px;
        height: 200px;
        margin-left: calc(50% - 150px); /* 50% of the container width minus half o
        background-color: #8fa3bf;
        text-align: center;
        line-height: 200px;
        color: white;
    }
</style>
```

takes the 50% of the min of half of element

example 3 combining calc with padding

```
<style>
    .content-box {
        width: calc(100% - 40px); /* Takes full width minus padding */
        padding: 20px;
        border: 2px solid #333;
        box-sizing: border-box; /* Ensures width includes padding and border */
        background-color: #ffeadb;
    }
</style>
```

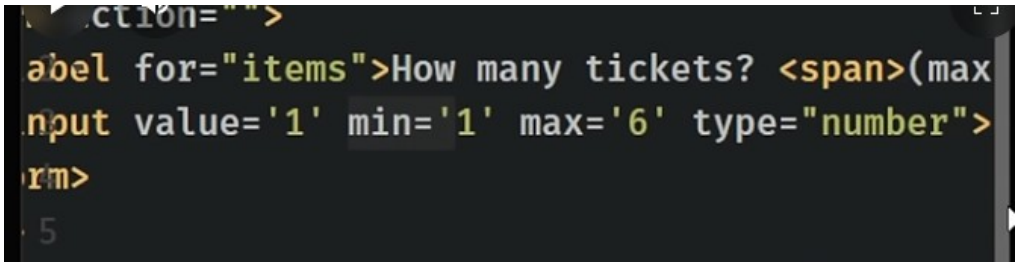
Peter Wolf
<Software Developer & Advisor />

Device Sizes Covered:

1. **Mobile (320px - 600px)**: For small mobile devices, such as smartphones in portrait mode.
2. **Tablet (601px - 900px)**: For tablets or smaller devices in landscape mode.
3. **Desktop (901px - 1200px)**: For standard desktop screens or larger tablets.
4. **Large Desktop (1200px and up)**: For large desktop monitors or wide-screen displays.

From styling pro tips

input type number




```
input[type="number"]:not(:in-range) {
  background-color: hsl(348, 55%, 49%);
  outline: 3px solid hsl(348, 55%, 49%);
}
```

Peter Wolf
<Software Developer & Advisor />

Media print

with the css attribute

@media print {}

you can change the design behavior for printing a page.#

TIPS für Cross Device usage:

1. platziere die @media print{} abfrage Immer am Ende der CSS datei. Manche Browser überschreiben sonst die regulären responsive @media screen queries.

2. Margin und padding können manchmal unterschiedlich erscheinen. Mit @media print die margin und padding neutralisieren und auf NULL setzen.

3. Flexbox kann in print wie auch in emails nicht bis falsch interpretiert werden. Für print am besten floats oder inline-blocks verwenden.

Summary:
Use @media print rules at the end of your CSS to ensure they override previous rules.

- Stick to basic layouts, avoiding complex flex or grid arrangements if precise print output is required.
- Test across browsers (especially Chrome, Firefox, Safari, and Edge) to catch any inconsistencies.
- Avoid backgrounds unless absolutely necessary, as they may not print by default.
- Explicitly define page-break-* properties with fallbacks and test how they break in Chrome and Firefox.

```css
<style>
  /* Basic Reset */
  * {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
  }

  /* Navbar container */
  .navbar {
    background-color: #333;
    padding: 1rem;
    font-family: Arial, sans-serif;
  }

  /* Navbar items */
  .navbar ul {
    list-style: none;
    display: flex;
    gap: 1rem;
  }

  .navbar li {
    position: relative;
  }

  /* Navbar links */
  .navbar a {
    color: white;
    text-decoration: none;
    padding: 0.5rem 1rem;
    display: block;
  }

  .navbar a:hover {
    background-color: #555;
  }

  /* Dropdown container */
  .dropdown-container {
    position: relative;
  }

  /* Dropdown menu */
  .dropdown {
    position: absolute;
    top: 100%;
    left: 0;
    background-color: #333;
    display: none; /* Ensure dropdowns are hidden by default */
    min-width: 150px;
    opacity: 0;
    transition: opacity 0.3s ease;
  }

  .dropdown li {
    width: 100%;
  }

  /* Show dropdown on hover of dropdown container */
  .dropdown-container:hover .dropdown {
    display: block;
    opacity: 1;
  }
</style>
```

Navigation DropDown menu only with CSS

```html
<div class="navbar">
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li class="dropdown-container">
      <a href="#">Services</a>
      <!-- Dropdown menu -->
      <ul class="dropdown">
        <li><a href="#">Web Design</a></li>
        <li><a href="#">Development</a></li>
        <li><a href="#">SEO</a></li>
      </ul>
    </li>
    <li><a href="#">Contact</a></li>
  </ul>
</div>
```

o developer & advisor

Peter Wolf
<Software Developer & Advisor />

CSS LIMITATIONS

CSS is a powerful tool for styling websites, but it does come with certain limitations that can make some designs challenging or impractical. Here are some of the major limitations in CSS:

1. Lack of Logic and Conditionals

- Issue : CSS does not support true logical conditions like `if` statements or loops, so you can't apply styles based on complex conditions directly in CSS (e.g., "if this element has class A and class B, apply style X").

- Workarounds : CSS preprocessors like Sass and Less offer limited conditionals, but they’re only evaluated at compile time, not in the browser. JavaScript can also be used to dynamically apply or remove classes.

2. No Real Variables in Pure CSS (Until CSS Custom Properties)

- Issue : Prior to CSS Custom Properties (`--variables`), CSS lacked true variables. Variables help maintain and update values across a site without redundant code.

- Current State : CSS variables (`--var`) work well but have limitations: they don’t support conditions, functions, or complex expressions. Also, browser support can vary in older environments.

3. Limited Math Functions

- Issue : While CSS has `calc()`, its math capabilities are limited to basic arithmetic (`+`, `-`, `*`, `/`). It doesn’t support more complex functions like trigonometry, powers, or condition-based calculations.

- Workarounds : Complex calculations must be done in preprocessors or JavaScript, limiting what you can achieve solely with CSS.

4. Inconsistent Browser Support

- Issue : Different browsers and their versions support CSS properties inconsistently. For example, older versions of Internet Explorer or Edge may lack support for newer properties like Grid, Flexbox, and Custom Properties.

- Workarounds : Use feature detection (e.g., `@supports`) or polyfills for better compatibility, but this adds extra steps and often some code bloat.

5. Cannot Target Parent or Previous Sibling Elements

- Issue : CSS can select child and next sibling elements but has no mechanism for targeting parent elements or previous siblings. For instance, there is no simple way to style a parent element when one of its children is focused.

- Workarounds : Use JavaScript to dynamically apply classes to parent elements or use `:has()` for parent selection, though `:has()` has limited support and can impact performance.

6. Limited Dynamic Adaptability (No True State Management)

- Issue : CSS has limited tools for dynamically adapting to real-time changes on a page. For example, you can't directly use CSS to change styles based on state or data, like calculating width based on dynamic content.

- Workarounds : CSS variables can be modified with JavaScript, and frameworks like React or Vue handle state more robustly, but this often requires JavaScript.

7. Complex Layouts Can Be Difficult

- Issue : While CSS has improved with tools like Grid and Flexbox, complex or intricate layouts (like asymmetrical grids or intricate layered designs) can still be cumbersome. Aligning and positioning elements across dynamic screen sizes requires careful planning and can lead to complex, hard-to-maintain CSS.

- Workarounds : Carefully combine Flexbox, Grid, and absolute positioning or use a framework. However, designing and maintaining these complex layouts can still be challenging.

8. Limited Animation Capabilities

- Issue : CSS animations work well for simple transitions but lack fine-grained control, like timelines or complex sequences, that are common in animation libraries (e.g., GSAP).

- Workarounds : JavaScript libraries provide more complex animation control, though this adds more dependencies and can slow down page load times.

9. No Direct Interaction with User Input

- Issue : CSS can't directly capture or respond to user input, such as form values, which limits its utility for interactive or dynamic interfaces.

- Workarounds : JavaScript is required for any interactive behavior, such as conditionally showing or hiding elements based on input.

10. Limited Control Over Document Flow

- Issue : CSS generally applies a top-to-bottom flow, and it's difficult to reorder elements visually without affecting their DOM order.

- Workarounds : Use `flex` or `grid` layouts to change the visual order of elements, though this can complicate accessibility and isn't ideal for all layouts.

11. No Native Support for CSS Functions (Like Loops or Mixins)

- Issue : CSS doesn't natively support loops, mixins, or reusable code blocks, which are common in Sass or Less. This means there's no simple way to repeat styles or define reusable chunks.

- Workarounds : Use preprocessors like Sass or Less, which add functions like mixins, loops, and conditionals at compile time, though these are not evaluated dynamically by the browser.

12. Print-Specific Limitations

- Issue : While CSS offers `@media print`, print styles can be inconsistent across browsers, and some CSS properties don't work well for printed layouts (e.g., background images or animations).

- Workarounds : For consistent print styling, stick to basic properties and test across multiple browsers.

13. Specificity and Inheritance Challenges

- Issue : CSS's specificity model can lead to "cascading" issues, where one rule unintentionally overrides another, making debugging and maintenance difficult in large projects.

- Workarounds : Careful CSS architecture practices, like using BEM (Block Element Modifier) or utility classes, can help reduce these issues, though they require careful planning

Design Material

Frameworks

Bootstrap

Tailwind

Projects

Homepage design and programming

jeder TN soll ein MVP design für eine Website erstellen.

Dazu gehört:

MockUp => skizze der Seite auf Papier oder mit https://wireframe.cc/

Umsetzung => visual studio code

Mobile und Desktop

Barrierefreiheit beachten

Form interaction

Ein formular nach Wahl erstellen.

Erfasste Daten sollen dann im Browser ausgegeben werden.

Nutzerrückmeldung

toDoList

Erstelle ein TODO list mit dummy daten.

Wenn gruppe in JS erfahrung hat