

WEB Engineering 2023/24

Prof. Dr. Peter Davids

FB03 Elektrotechnik & Informatik

Hochschule Niederrhein

Aufbau eines HTML-Dokuments

```
<html>
  <head>
    <title>
      Titel des Dokuments
    </title>
  </head>
  <body>
    ....
  </body>
</html>
```

World Wide Web

Tim Berners-Lee prägte den Begriff WorldWideWeb **1990** und legte das Fundament für das, was heute als „Web1.0“ angesehen wird (und immer noch in den meisten Technologien, die wir heute im Internet nutzen, spürbar ist). Das Web 1.0 war eine Nischen-technologie mit sehr begrenzten visuellen Ausdrucksmöglichkeiten und Design-Optionen.

<http://info.cern.ch/hypertext/WWW/TheProject.html>

WEB 2.0

1999 wurde der Begriff „Web 2.0“ geprägt. Dabei handelte es sich nicht um ein klar abgegrenztes Update wie bei einer Software, sondern die Essenz von vielen verschiedenen sozialen und technischen Entwicklungen, die unter diesem Überbegriff zusammengefasst wurden.

WEB 3.0

Mehr oder weniger direkt als Reaktion auf den Begriff „Web 2.0“ entwickelte sich (vorwiegend in akademischen Kreisen) die Idee eines „Web 3.0“, eines sogenannten „semantischen Webs“, das die Daten im Web für Maschinen und Software les- und daher nutzbar machen sollte. Aber diese Variante des Web3.0 fand nie so richtig Anklang.

Das Dritte WEB

```
<iframe width="100%" height="500" src="https://tante.cc/2022/02/04/das-dritte-web/" frameborder="0"> </iframe>
```

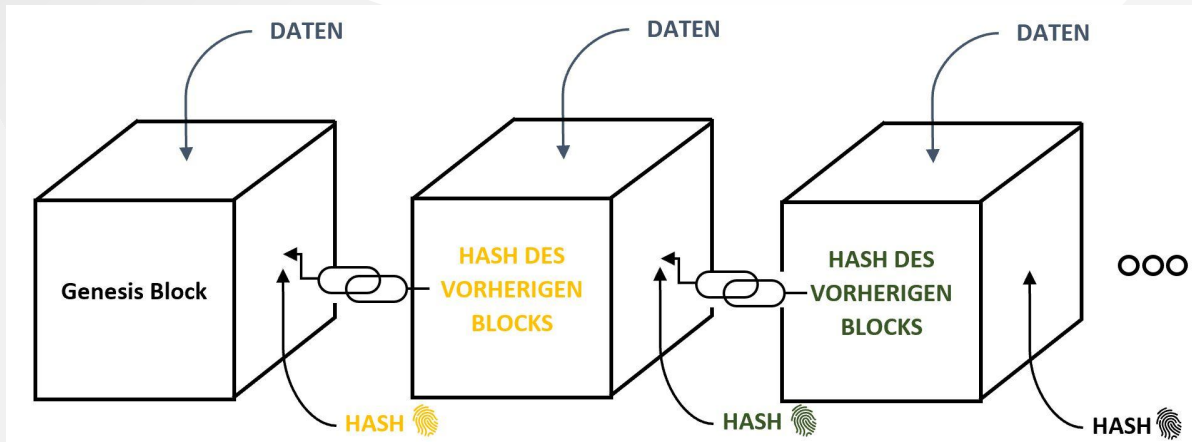
Quelle: <https://tante.cc/2022/02/04/das-dritte-web/>

Das dritte WEB

- Motivationen das bestehende WEB (2.0) zu verändern
- Technologien:
 - Blockchains
 - NFT (non-fungible tokens)
 - DAO (decentralized autonomous organizations)

Blockchains

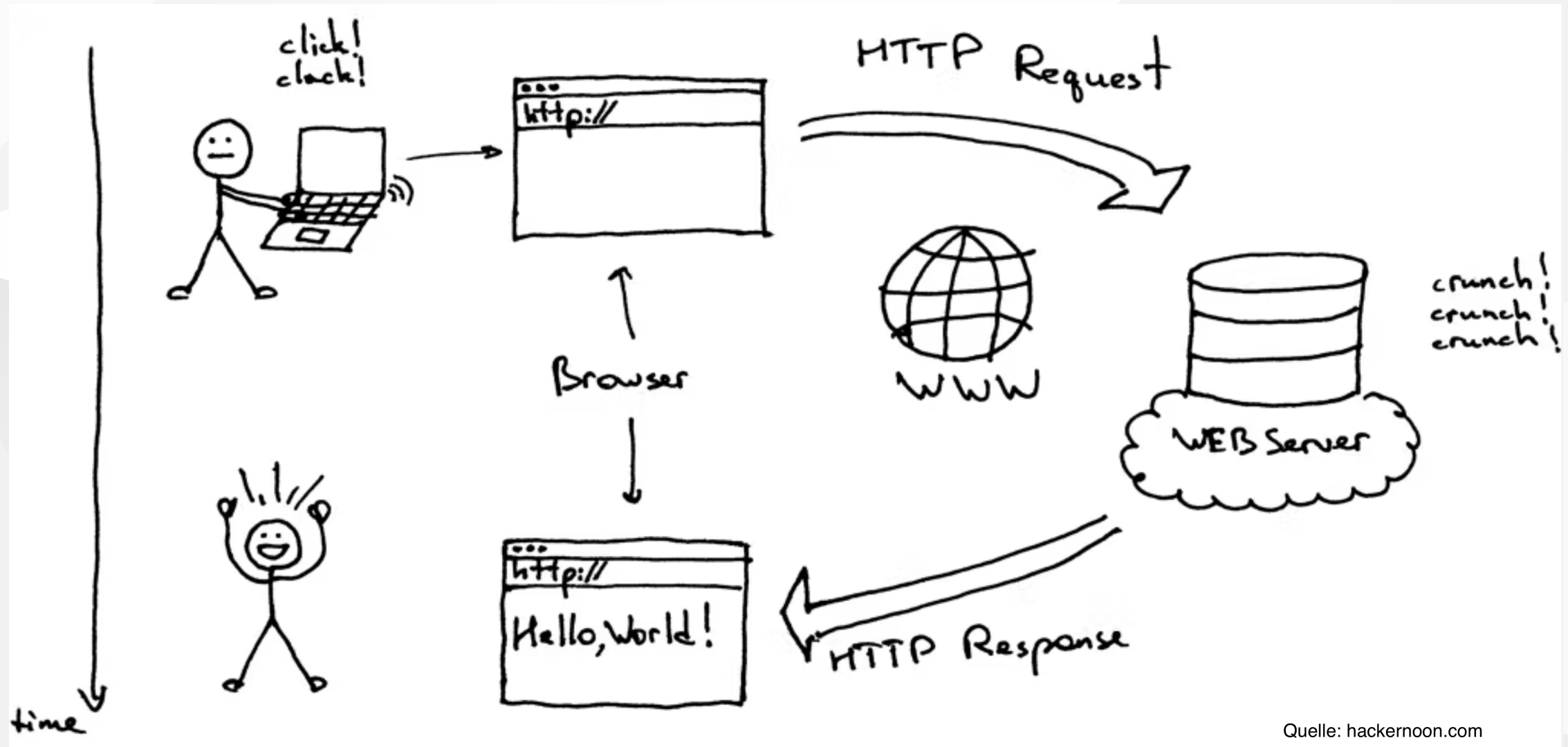
- spezielle Art von Datenbank
- dezentral (statt zentral)
- jeder Knoten hat alle Daten (lokal) bzw alle Blöcke
- jeder Block hat einen Hashwert, der im Nachfolgeblock hinterlegt wird



HTML

- HTML = Hypertext Markup Language
- Textbasierte Auszeichnungssprache für die Beschreibung von Inhalten eines Dokumentes
- HTML-Dokumente werden von einem **Browser** bzw. Webbrowser interpretiert
- W3C Standard ([World Wide Web Consortium](#))

Blickwinkel 1 - Technik



Quelle: hackernoon.com

Blickwinkel 2 - Software / Sprachen

- HTML
- CSS
- Javascript
- statisch / dynamisch
- Front-End / Back-End

Blickwinkel 3 - Tools / Frameworks

- Diverses von JetBrains (<https://jetbrains.com>)
- gitlab Pages (<https://git.ide3.de/web2023/beispiele/pages>)
- Hugo (<https://git.ide3.de/web2023/beispiele/hugo>)
- Content Management Systeme ([wikipedia/en](https://en.wikipedia.org))
- Server Side Frameworks ([wikipedia/en](https://en.wikipedia.org))
- Javascript Frameworks ([wikipedia/en](https://en.wikipedia.org))
- CSS Frameworks ([github](https://github.com))
- Static Site Generators ([jamstack](https://jamstack.org))

HTML Versionen

- ~~HTML: Urversion Anfang der 90er Jahre~~
- ~~HTML 2.0 (1995): Formulare~~
- ~~HTML 3.2 (1997): Tabellen, Bilder~~
- ~~HTML 4.0 (1998): Stylesheets, Frames~~
- ~~HTML 4.01 (1999): Standard bis HTML5~~
- HTML5 (2014): Letzte gültige W3C Empfehlung

Fehlertoleranz in C

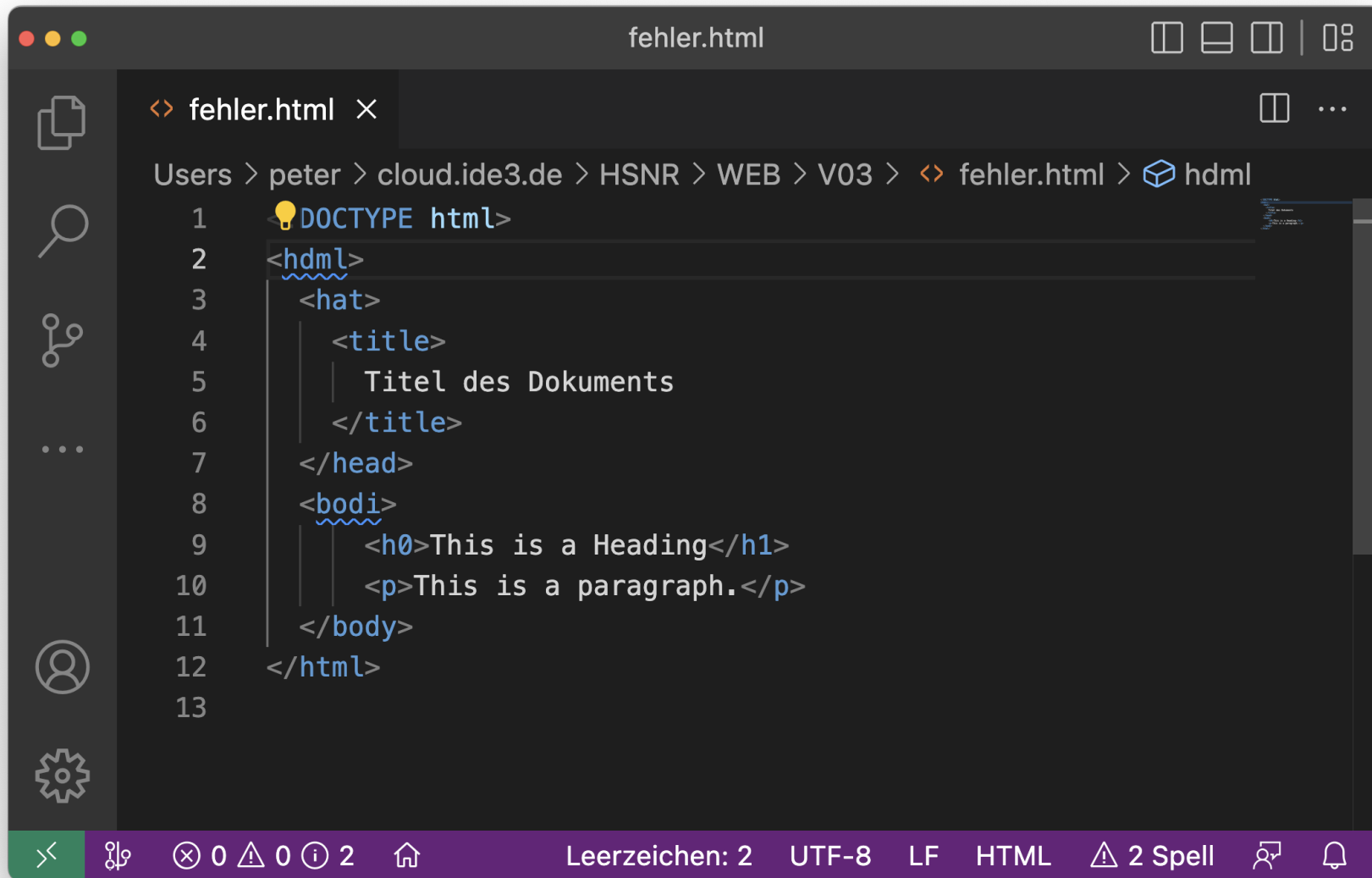


```
peter@M1-2:~/tmp
peter@M1-2 ~/tmp
$ cat hello.c
#include <stdio.h>

int main (void) {
    printf ("hello")
}
peter@M1-2 ~/tmp
$
peter@M1-2 ~/tmp
$ make hello
cc      hello.c  -o hello
hello.c:4:19: error: expected ';' after expression
    printf ("hello")
                    ^
                    ;
1 error generated.
make: *** [hello] Error 1
peter@M1-2 ~/tmp
$
```

The terminal window shows a user named 'peter' at host 'M1-2' in the directory '~/tmp'. The user creates a file 'hello.c' containing a C program. The program has a syntax error: a missing semicolon at the end of the 'printf' statement in the 'main' function. When the user runs 'make hello', the compiler 'cc' reports the error: 'hello.c:4:19: error: expected ';' after expression'. The error message also shows the caret pointing to the end of the 'printf' line and the semicolon on the next line. The user then runs '\$' to start a new shell prompt.

Fehlertoleranz in HTML



```
fehler.html
Users > peter > cloud.ide3.de > HSNR > WEB > V03 > fehler.html > hhtml
1  <!DOCTYPE html>
2  <hml>
3    <hat>
4      <title>
5        Titel des Dokuments
6      </title>
7    </head>
8    <bodi>
9      <h0>This is a Heading</h1>
10     <p>This is a paragraph.</p>
11   </body>
12 </html>
13
```

Leerzeichen: 2 UTF-8 LF HTML 2 Spell

Funktionsweise eines Browsers

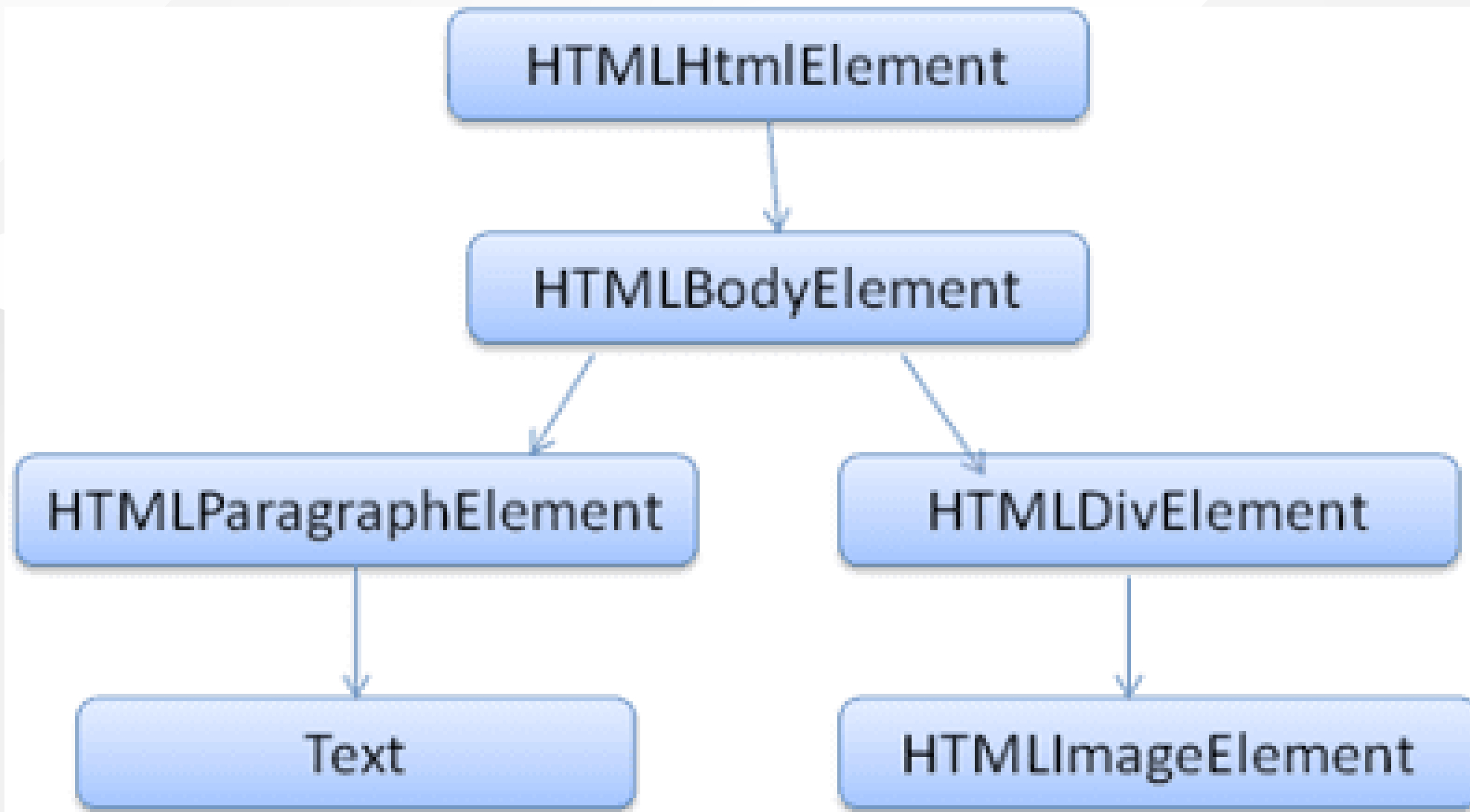
sehr interessant:

- <http://taligarsiel.com/Projects/howbrowserswork1.htm>
- <https://web.dev/articles/howbrowserswork?hl=en>

Quellcode, z.B.: Waterfox:

- <https://github.com/WaterfoxCo/Waterfox>

DOM Tree



HTML Elemente

- Element: alles vom Start-Tag bis zum Ende-Tag

```
<h1>Überschrift</h1>
```

- Es gibt *leere* Elemente, die kein Ende-Tag haben

```
<br>
```

- HTML Elemente können geschachtelt werden

```
<body> <h1>...</h1> </body>
```

- HTML ist nicht *case sensitive*

```
<p> ist das Gleiche wie <P>
```

- Alle HTML Elemente gibt es hier: <https://www.w3schools.com/tags/default.asp>

HTML Attribute

- Alle HTML Elemente können Attribute haben
- Attribute liefern weitere Informationen zu einem Element
- Attribute befinden sich im Start-Tag
- meist als Name/Value Paare `href="/123.png"`
`Visit Git`
- Üblich: Attributnamen in *lower case*
- Üblich: Attributwerte in `"` `'` einfassen
- Alle HTML Attribute gibt es hier:
https://www.w3schools.com/tags/ref_attributes.asp

Aufbau eines HTML-Dokuments

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p>Hello World</p>
    <div> </div>
  </body>
</html>
```

HTML Header und Meta Daten

```
<head>
  <title> ... </title>
  <style> ... </style>
  <script> ... </script>
  <meta charset="UTF-8">
  <meta name="description" content="Free Web tutorials">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

Üblich: document title, character set, styles, scripts, and other meta information

Wie gelangt HTML Code in den Browser

1. Statisch - Editor, lokale Datei: statisch.html
2. Statisch - über das Netzwerk, von einem Web Server:
<https://s.ide3.de/web/statisch.html>
3. Dynamisch - Lokale Datei mit Javascript: dyn.html
4. Dynamisch per Programm/Skript von einem Server: <https://s.ide3.de/py/web.py>

HTML Block Elemente

- beginnen in neuer Zeile
- enden mit einem Zeilenumbruch
- nutzen meist vollständige Breite
- Browser ergänzt diverse Ränder
- Beispiel: `<p>` `<div>`

<code><address></code>	<code><article></code>	<code><aside></code>	<code><blockquote></code>	<code><canvas></code>	<code><dd></code>	<code><div></code>
<code><dl></code>	<code><dt></code>	<code><fieldset></code>	<code><figcaption></code>	<code><figure></code>	<code><footer></code>	<code><form></code>
<code><h1>–<h6></code>	<code><header></code>	<code><hr></code>	<code></code>	<code><main></code>	<code><nav></code>	<code><noscript></code>
<code></code>	<code><p></code>	<code><pre></code>	<code><section></code>	<code><table></code>	<code><tfoot></code>	<code></code>
<code><video></code>						

HTML Inline Elemente

- beginnen **nicht** in neuer Zeile
- benutzen soviel Raum wie nötig
- Beispiel: ``

<code><a></code>	<code><abbr></code>	<code><acronym></code>	<code></code>	<code><bdo></code>	<code><big></code>	<code>
</code>
<code><button></code>	<code><cite></code>	<code><code></code>	<code><dfn></code>	<code></code>	<code><i></code>	<code></code>
<code><input></code>	<code><kbd></code>	<code><label></code>	<code><map></code>	<code><object></code>	<code><output></code>	<code><q></code>
<code><samp></code>	<code><script></code>	<code><select></code>	<code><small></code>	<code></code>	<code></code>	<code><sub></code>
<code><sup></code>	<code><textarea></code>	<code><time></code>	<code><tt></code>	<code><var></code>		

HTML & Javascript Formular

```
<form>
  X= <input onChange="calc()" type="text" id="x"><br>
  Y= <input onChange="calc()" type="text" id="y"><br><br>
  X+Y= <span id="sum" ></span>
</form>
```

Attribut `onChange=` ruft Javascript Funktion nach Wertänderung auf

`..` ist leeres Element

`id= "..."` gibt jedem Element einen Namen

HTML & Javascript Formular

```
<script>
  function calc() {
    var x = document.getElementById("x").value;
    var y = document.getElementById("y").value;
    console.log(x); console.log(y);
    var sum = x + y; // klappt das ???
    document.getElementById("sum").innerHTML= sum;
  }
</script>
```

`function` definiert Javascript Funktion

`document.getElementById("x")` gibt Objekt mit ID `x`

aber in `var x` ist eine Zeichenkette und

`var sum = x + y` konkateniert Zeichenketten!

Einbinden von Javascript in HTML Code:

- inline (innerhalb des HTML Codes; an beliebigen Stellen)
- extern (Einbinden von separaten .js Dateien; i.d.R. im `<head>`)

```
<html>
  <head>
    <script src="myScript.js"></script>
    <script src="https://www.w3schools.com/js/myScript.js"></script>
  </head>
  <body>
    <script>
      console.log ("hoho");
    </script>
  </body>
</html>
```

DOM - finden eines Elementes:

```
...  
<div id="tester">...</div>  
<script>  
const element = document.getElementById("tester");
```

Wenn Element nicht gefunden wurde, hat `element` den Wert `null`

```
element.innerHTML = "neuer Inhalt";
```

oder

```
document.getElementById("tester").innerHTML = "neuer Inhalt";
```

Javascript

Operatoren:

- Arithmetisch/Logisch: `+` `-` `*` `/` `%` `||` `&&` `!`
- Verkettung: `+`
- Zuweisung: `=` `+=` `*=` etc.
- Vergleich: `==` `<` etc.. aber auch `===` (Wert und Typ gleich)

Kontrollstrukturen (vergleichbar zu C):

- `if` / `switch`
- `for` / `while` / `do while`

Funktionen

```
function prod(x,y) {           // prod ist Name der Funktion, x y sind Parameter
    return x * y;              // keine () notwendig
}

let preis = prod (1.5,1.19);
```

- `()` ist der Operator, der die Funktion aufruft
- Der Funktionsname (ohne `()`) gibt das Funktions-Objekt zurück

[W3schools](#)

Event Listener

- Event Listener sind Mechanismen, die Ereignisse im Browser erkennen (Mouce Click etc.)
- Als Reaktion führen sie Javascript aus (Code / Funktionen)
 - `window.onload` / `window.onunload` (Seite geladen/wird verlassen)
 - `onclick` / `ondblclick` - Element wird (doppelt) angeklickt
 - `onmouseover` - Maus befindet sich über einem Element (Tablet ??)
 - `onmouseout` - Maus bewegt sich weg
 - `onmousedown` / `onmouseup` - Maus gedrückt / losgelassen

Events I

- HTML Attribut

```
<button onclick="displayDate()">Try it</button>
```

- HTML DOM

```
<script>  
document.getElementById("myBtn").onclick = displayDate;    // keine ()  
</script>
```


- `addEventListener()`

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

- oder Funktion mit Namen

```
element.addEventListener("click", myFunction);
```

```
function myFunction() {  
    alert ("Hello World!");  
}
```

DOM Navigation I

- alle Knoten in einem DOM können adressiert / erreicht werden
- Knoten können erzeugt / verändert / gelöscht werden
- jeder Knoten hat genau einen Vorgänger (parent) ausser
- der oberste Knoten, der Knoten heisst Wurzel (root)
- ein Knoten kann Nachfolger haben (child)

DOM Navigation II

- Navigation über Eigenschaften eines Knotens

```
parentNode          // Vorgänger  
childNodes[nodenum] // Array von Kindsknoten  
firstChild           // erstes Kind  
lastChild            // letztes Kind  
nextSibling          // nächster Bruder/Schwester  
previousSibling      // vorheriger Bruder/Schwester
```

DOM - Erzeugen neuer Elemente / Kinder

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
  // hier wird das neue Kind eingehängt
</div>
<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
element.appendChild(para);
</script>
```

New last child

DOM - Erzeugen neuer Elemente / Geschwister

```
<div id="div1">
// hier wird das neue Kind eingehängt
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para,child);           // Methode!
</script>
```

New first child

DOM - Löschen von Elementen

```
<div>
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
const element = document.getElementById("p1");
element.remove();
</script>
```

geht nicht in alten Browsern

DOM - Löschen von Kind-Elementen

- Vorgänger suchen, Kind suchen, löschen:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

AJAX I

Asynchronous JavaScript and XML

- Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server
- Rendering / Seitenaufbau soll nicht durch (synchrones) Javascript blockiert werden
- Seite soll sich ohne Neuladen verändern lassen (durch nachgeladene Inhalte)
- i.d.R. wird TEXT oder JSON übertragen (kein XML)

https://www.w3schools.com/js/js_ajax_intro.asp

AJAX II

- Lesen von Daten von einem Server - nachdem die Seite geladen ist
- Modifizieren einer Seite ohne vollständiges Neuladen
- Senden von Daten an einen Server - im Hintergrund / ohne Warten

```
function loadData() {  
    const xhttp = new XMLHttpRequest();  
    xhttp.onload = function() {  
        document.getElementById("demodata").innerHTML = this.responseText;  
    }  
    xhttp.open("GET", "data.txt", true);  
    xhttp.send();  
}
```


HTTP

- zustandsloses Protokoll
- Zustand muss mit übertragen werden (z.B. Cookies)
- zur Übertragung von Daten auf der Anwendungsebene (Layer 7)
- i.d.R. zwischen Web Browser und Web Server
- auf Basis von TCP (bzw. QUIC bei http/3.)
- TCP Port 80 ohne Verschlüsselung
- TCP Port 443 bei Verwendung von HTTPS (TLS, früher SSL)
- zwei Nachrichten-Typen: **Request & Response <==> Relevanter Teil**

HTTP Anfrage Methoden

- GET : fordert Dokument (URI*) vom Server an (incl. Parameter ?=)
- POST : sendet Daten (z.B. key/value Paare aus Formularen)
- HEAD : wie GET aber ohne Inhalt (nur Header)
- PUT/PATCH/DELETE: Hochladen, Ändern, Löschen von Dokumenten (webdav)
- CONNECT : Proxy-Server generieren verschlüsselte Tunnel

URI : Uniform Resource Identifier (manchmal auch *universal*)

HTTP Status-Codes

- 1XX : Informationen (z.B. Protokollwechsel)
- 2XX : Erfolgreiche Operation (200 == **OK**)
- 3XX : Umleitung (Page Moved / Umleitung, 301)
- 4XX : Client Fehler (404 == **not found**, 403 == **forbidden**)
- 5XX : Server Fehler (502 == **bad Gateway**; Proxy-Error)

HTTP URI

[illegible]

- scheme : Protokoll (http, https, ftp, ...)
- authority : User, DNS Name und ggfls. Portnummer
- path : (hierarchischer) Dokumentenpfad
- query : weiter spezifizierende Abfrage (i.d.R. Skript-Parameter)
- fragment : Sprungmarke auf Anker ``

```
body {  
  background:#fefefe;  
  color:#222;  
  font:100%/1.5 Calibri,Verdana,Helvetica,sans-serif;  
  max-width:750px;  
  margin:0 auto 30px;  
  padding:0 20px;  
}  
  
h3 {  
  background-color:#88b1a6;  
  border-bottom:1px solid #415752;  
  border-top:1px solid #415752;  
  color:#415752;  
  font-size:150%; margin-bottom:20px;  
  padding:10px 18px; text-align:center;  
  text-shadow:0 1px 0 rgba(219,231,228,.5);  
}
```

CSS Properties I

- background (9)
 - background-color (Javascript: `backgroundColor`)
 - background-image
- border (32)
 - border-width (Javascript: `borderWidth`)
- color
- display / visibility

CSS display Properties

https://www.w3schools.com/cssref/pr_class_display.php

- inline
- block
- ~~flex~~ <- Nur warum es nützlich ist
- ~~grid~~ <- Nur warum es nützlich ist
- none
- inherit

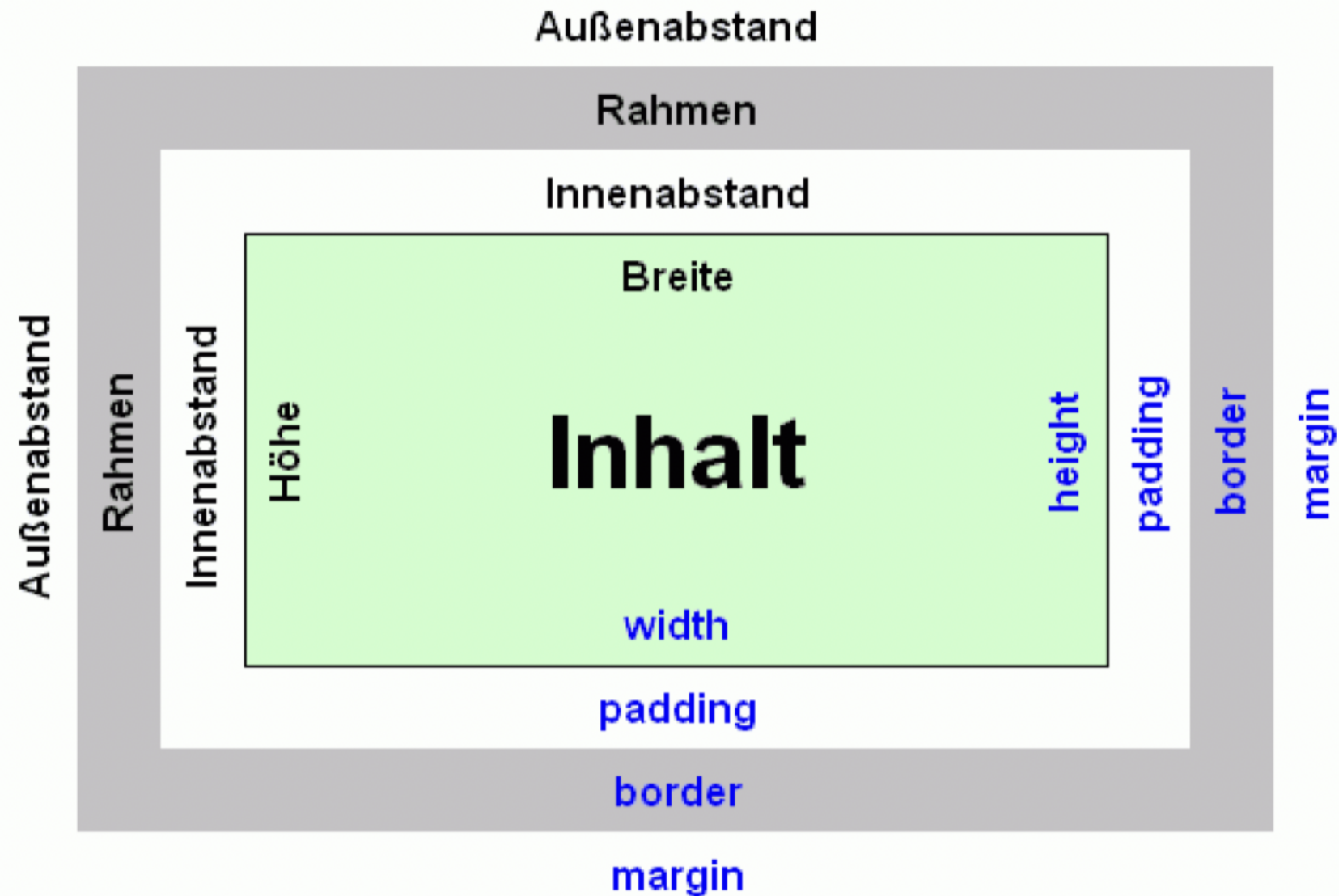
Operatoren

	Beispiel	Bedeutung
<code>_</code>	<code>div p</code>	alle <code><p></code> Elemente innerhalb eines <code><div></code> Elementes
<code>.</code>	<code>div.bunt</code>	alle <code><div></code> Elemente mit <code>class="bunt"</code>
<code>></code>	<code>div>p</code>	alle <code><p></code> Elemente wenn <code><div></code> Eltern Element ist
<code>+</code>	<code>div+p</code>	erstes <code><p></code> Element hinter <code><div></code> Element
<code>~</code>	<code>p~ul</code>	alle <code></code> Elemente die auf ein <code><p></code> Element folgen
<code>,</code>	<code>div,p</code>	alle <code><div></code> und <code><p></code> Elemente
<code>[]</code>	<code>[src]</code>	alle Elemente mit Attribut <code>src</code>

Selektor Referenz

- Liste aller Selektoren und Typen:
https://www.w3schools.com/cssref/css_selectors.php
- Zum Üben (für die Klausur):
<https://www.w3schools.com/cssref/tryssel.php>

Box - Model



Box Sizing

```
#border {  
  box-sizing: border-box;      // incl. border  
}  
  
#content {  
  box-sizing: content-box;     // content only  
}
```

https://www.w3schools.com/cssref/tryit.php?filename=trycss3_box-sizing

Frage: was ist mit `margin:` ?

Text

- Color / Farbe
- Alignment / Ausrichtung
- Decoration
- Transformation
- Spacing
- Shadow

Fonts

- Zeichensatzfamilien (family)
- Fallback-Fonts

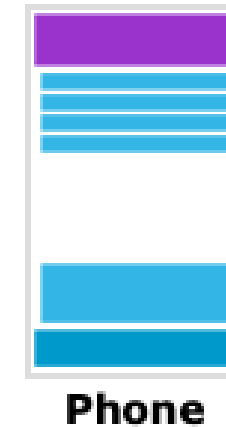
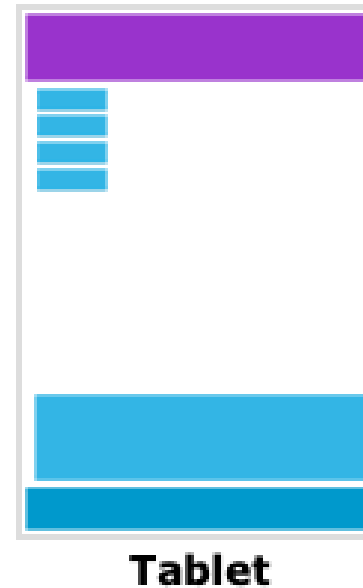
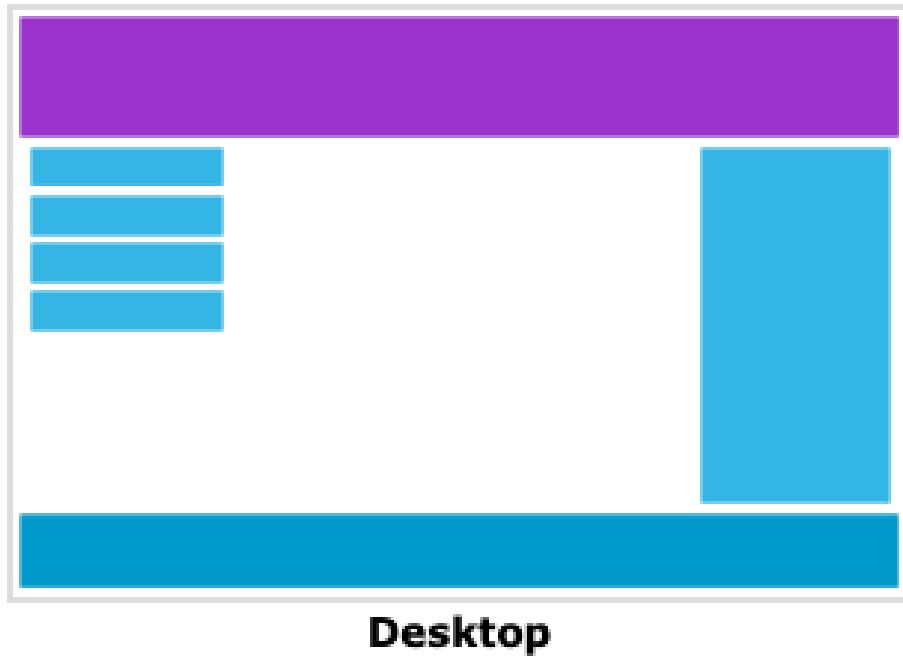
```
p {  
  font-family: Arial, Helvetica, sans-serif;  
}
```

- Web Safe Fonts:
- Arial, Verdana, Tahoma, Trebuchet MS (sans-serif)
- Times New Roman, Georgia, Garamond (serif)
- Courier New (monospace)
- Brush Script MT (cursive)

https://www.w3schools.com/css/tryit.asp?filename=trycss_font-family

RWD - Responsive Web Design

- Gute (übersichtliche) Seitendarstellung auf allen möglichen Endgeräten
- Verwendung von **HTML** und **CSS**
- Kein Javascript notwendig



RWD Regeln

- keine **breiten** Elemente mit fester Breite (größer Fensterbreite) verwenden
- keine **beliebigen** Bildschirmbreiten annehmen
- verwende **Media Queries** um das Layout an Bildschirmgröße und Layout (Portrait / Landscape) anzupassen

Folgen:

- horizontales Scrollen wird vermieden!
- manuelles Zoomen wird vermieden

Media Queries (CSS3)

@media

```
@media only screen and (max-width: 600px) {  
  body {  
    background-color: lightblue;  
  }  
  .toolarge {  
    display:none;    // Ausblenden  
  }  
}
```

Wenn das Browser Fenster 600px oder kleiner ist : himmelblau! und bestimmte Elemente ausblenden

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_mediaquery

Absicherung von Clients und Servern

- Client (Browser)
 - Netzwerk schützen (LAN/Firewall/kein offenes WLAN/VPN/Firewall/PiHole...)
 - Aktuelles Betriebssystem und aktuelle Browser verwenden
 - Passwort-Caching im Browser nur auf *vertrauenswürdigen* Endgeräten nutzen
 - Browser-Plugins *mit Bedacht* auswählen
 - Trennung von *kritischen Webanwendungen* (Banking etc.) z.B durch virtuelle Maschinen

Absicherung von Clients und Servern

- Server (apache/nginx/Datenbanken/sonstige *Backends*)
 - Aktuelles Betriebssystem und aktuelle Software verwenden
 - *Security by Isolation* (siehe ITS Vorlesung) (iptables, Netzstruktur, etc..)
 - nicht genutzte Komponenten (z.B. php oder andere Module des Webservers) entfernen
 - (Transport-)Verschlüsselung aktivieren

TLS Handshake (lt. ChatGPT)

1. Client-Server Handshake:

- Der Client sendet eine Anfrage (Client Hello) an den Server.
- Der Server antwortet (ServerHello) mit einem digital signierten Zertifikat.

2. Zertifikat-Prüfung durch den Client:

- Der Client überprüft das Zertifikat auf Gültigkeit und Authentizität.
- Der Client verwendet öffentliche Schlüssel des Zertifikats, um eine sichere Verbindung herzustellen.

3. Schlüsselaustausch:

- Der Client und der Server verwenden den öffentlichen Schlüssel des Zertifikats, um einen gemeinsamen geheimen Sitzungsschlüssel auszuhandeln.
- Dieser Sitzungsschlüssel wird für die Verschlüsselung und Entschlüsselung von Daten während der aktuellen Verbindung verwendet.

TLS Handshake

4. Verschlüsselte Datenübertragung:

- Die gesamte Kommunikation zwischen Client und Server erfolgt verschlüsselt unter Verwendung des gemeinsamen Sitzungsschlüssels.
- Die Daten können sicher über das Netzwerk übertragen werden.

Gültigkeit:

- Start- und Ablaufdatum erreicht bzw. nicht überschritten
- Name (zB git.ide3.de)
- Signatur / Issuer Zertifikat bekannt und gültig

TLS Konfiguration - nginx

```
server {  
    listen 172.30.0.43:443 ssl;  
    server_name git.ide3.de;  
    ssl_certificate      /root/.acme.sh/git.ide3.de/fullchain.cer;  
    ssl_certificate_key  /root/.acme.sh/git.ide3.de/git.ide3.de.key;  
  
    access_log /var/log/nginx/git.ide3.de.log;  
  
    location / {  
        ...  
    }  
}
```

TLS Konfiguration - apache

```
<VirtualHost 172.30.0.43:443>
    ServerName git.ide3.de

    SSLEngine on
    SSLCertificateFile /root/.acme.sh/git.ide3.de/fullchain.pem;
    SSLCertificateKeyFile /root/.acme.sh/git.ide3.de/git.ide3.de.pem

    <Directory "/var/www/html">
        ...
    </Directory>
</VirtualHost>
```

TLS Zertifikat - Früher

Schritte 1 bis 3 wurden **gekauft**

- Laufzeit 1-3 Jahre
- 100,-- bis 300,-- € pro Jahr
- Nachweis des Domain Besitzes z.B. über Handelsregister-Auszug oder sonstige *Beglaubigungen* ([Organization Validation](#))

In der Regel: manuelle Prozesse mit vielen Beteiligten

- Einkauf
- Rechts-/Steuerabteilung
- IT Administration

Problem: [Abgelaufene Zertifikate](#)

TLS Zertifikat - seit 18. November 2014 [Let's Encrypt](#)

- [Domain Validation](#)
- Kostenlos
- 90 Tage gültig
- Gründer: Mozilla, EFF (Electronic Frontier Foundation), University of Michigan, Cisco, Akamai
- Wildcard Zertifikate (`*.ide3.de`) möglich
- Automatisierbar: [ACME Protokoll](#) (siehe auch <https://de.wikipedia.org/wiki/ACME>)
Clients: <https://letsencrypt.org/docs/client-options/>
mein Favorit: <https://github.com/acmesh-official/acme.sh>
- 2020: > 1 Millarde ausgestellte Zertifikate
- Dezember 2023: > 120 Mio. Domains, > 300 Mio. **aktive** Zertifikate, 40 Zertifikate / Sekunde (<https://letsencrypt.org/stats/>)

Domain Validation 1 - DNS API Integration

<https://github.com/acmesh-official/acme.sh/wiki/dnsapi>

z.B. netcup:

51. Use netcup DNS API to automatically issue cert

First you need to login in your CCP account to get your API Key and API Password.

```
export NC_Apikey="<Apikey>"  
export NC_Apипw="<Apipassword>"  
export NC_CID="<Customernumber>"
```

```
./acme.sh --issue --dns dns_netcup -d git.davids.de
```

Domain Validation 1 - DNS API Integration

<https://github.com/acmesh-official/acme.sh/wiki/dnsapi>

z.B. Hetzner:

74. Use Hetzner API

Get the API Token: Use dnsConsole to create your hetzner api token.

```
export HETZNER_Token="<token>"
```

```
./acme.sh --issue --dns dns_hetzner -d git.davids.de
```