HTML-Grundlagen

HTML ist eine Auszeichnungssprache (Markup Language).

Für die **Auszeichnung (das Markup) der Inhalte** werden Tags verwendet:
`<p>...</p>`
Leere Elemente: `<br />` oder `<br>`

Zeichencodierung

Unicode: 1'112'064 valid code points (charactes)

ISO-Codierungen

Bei ASCII ist das achte Bit (Bit 7) frei. Wenn Bit 7 auf 0 steht, handelt es sich um ein ASCII-Zeichen, andernfalls um ein Zeichen aus einer erweiterten Zeichentabelle (ISO). **Nachteil:** In einem reinen Textdokument können verschiedene Zeichen einer ISO-Codierungen verwendet werden.
Aus diesem Grund werden ISO-Codierungen heute im Web **nicht mehr eingesetzt.** Vorherrschend ist heute die **Unicode**-Variante **UTF-8.**

UTF-8

```
<meta charset="UTF-8">
```

– 1 up to 4 bytes per character
– encodes all valid Unicode points
– **more memory efficient** for latin (ascii) text **than UTF-16**
– less efficient for non-Latin scripts (Chinese, Japanese, Arabic, etc.)
– **fully backward compatible** with ASCII (encoding is identical)
– Vorherrschend im Web

Jedes Zeichen wird eine Bytekette variabler Länge zwischen einem und vier Bytes zugeordnet. Es ist abwärtskompatibel zu ASCII (1 Byter pro Zeichen).

Zeichen-Enkodierung:

Codes mit bis zu 7 Bits:	`0xxxxxxx`	(1 Byte)
Codes mit 8 bis 11 Bits:	`110xxxxx 10xxxxxx`	(2 Bytes)
Codes mit 12 bis 16 Bits:	`1110xxxx 10xxxxxx 10xxxxxx`	(3 Bytes)
Codes mit 17 bis 21 Bits:	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`	(4 Bytes)

UTF-16

– 2 or 4 bytes per character
– encodes all valid Unicode points
– **Not backward compatible** with ASCII
– **JavaScript repräsentiert Strings intern mit UTF-16**

Convert UTF-16 to UTF-8:

```
"👹".codePointAt() // returns index in UTF-16 code units
127801 = 00000000 00000001 11110011 00111001 (17 bits)

17 Bits benötigen eine 4-Byte-Enkodierung:
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx (Encoding template)
11110000 10011111 10001100 10111001 = F0 9F 8C B9 (UTF-8 encoding)
```

<!DOCTYPE html>

Deklariert, welche **DTD** (Document Type Declaration) verwendet werden soll.

Eine DTD beschreibt die erlaubten Elemente, Attribute und Kombinationsmöglichkeiten.

DTD:

```
<!ELEMENT peoplelist (person)*>
<!ELEMENT person (name, birthdate?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ATTLIST person
  member (true|false) #REQUIRED
  active CDATA #IMPLIED
>
```

XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE peoplelist SYSTEM "peoplelist.dtd">
<peoplelist>
  <person member="false">
    <name>Fred Bloggs</name>
    <birthdate>2008-11-27</birthdate>
  </person>
</peoplelist>
```

Container-Elemente und Überschriften

`section, article, aside` und `nav` definieren *Sectioning Content,* einen **eigenen** Geltungsbereich für Überschriften (`h1, h2, ...`).
Die Ebene wird dann aus der Dokumentenhierarchie abgeleitet.

Inline-Elemente

Elementen, die innerhalb von Fliesstext auftauchen können.

`strong, em, b, i, span, mark, time, meter, progress.`
`mark:` Textabschnitte hervorzuheben, z.B. markieren von Suchergebnissen.
`time: <time datetime="2012-07-17T17:00">5pm on July 17th</time>`

Block-Elemente

Elemente, die den verfügbaren horizontaler Platz einnehmen.
Vergleichbar mit Absatzstilen und Textstilen.

Support

canisuse.com, html-now.github.io

Content-Types

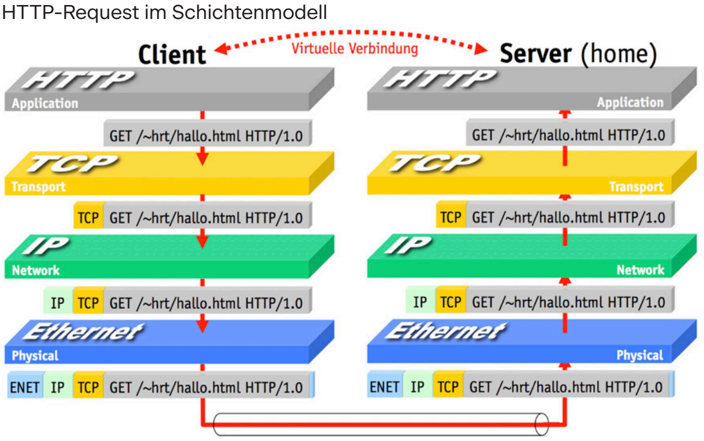
Entstammen historisch eigentlich dem E-Mail-Dienst. Werden auch als **Mime-Types** (Multipurpose Internet Mail Extensions) bezeichnet.

`text/html, text/css, application/javascript, application/pdf, image/jpeg, image/svg+xml, video/mp4, audio/mpeg`

Internet-Protokoll-Stack

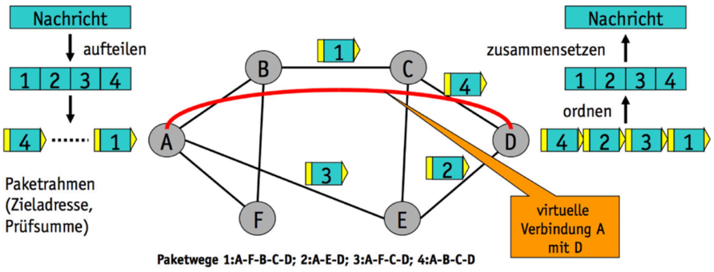
ISO/OSI	Internet	Protokolle	Typische Angaben
Application (Anwendung)	Application	HTTP FTP SMTP Telnet	URL: http://www.zhwin.ch Mailadresse: mustepet@zhwin.ch
Presentation (Darstellung)			
Session (Sitzung)			
Transport	Transport	TCP UDP	Portnummer 80 = HTTP 25 = SMTP
Network (Netzwerk)	Internet	IP	IP-Adresse 192.168.0.1
Data Link (Sicherung)	Physical / Access	Ethernet Wireless LAN Token Ring PPP/(Modem, ISDN, xDSL)	MAC-Adresse 00:0F:7F:23:45:67 Telefonnummer: 0878/123456
Physical (Bitübertragung)			

HTTP-Request im Schichtenmodell



Internet-Dienste: E-Mail, S/FTP, www (Web, seit 1992), etc.

Prinzip der Paketvermittlung



Web-Plattform

Browser = User Agent (Client)

Drei Grundpfeiler:

Sprachen: HTML, CSS, JavaScript
Adressierung: URI/URL, IP-Adresse/Domain-Name
Protokoll: HTTP(S)

Adressierung

Uniform Resource Locator, URL (heutzutage das Selbe wie URI)

```
"http://" [user[:password]@] host [:port] path [ "?" query ] [ "#" fragment ]

    scheme-specific-part →                              →               → |
    |
http://hans:geheim@example.org:80/demo/example.cgi?land=de&stadt=aa#geschichte
|       |    |          |       | |                |                  |
|       |    |        host      | url-path        query            fragment
|       |  password            port
|      user
scheme (hier gleich Netzwerkprotokoll)
```

IPv4: 32 Bit, 173.194.40.95

IPv6: 128 Bit, 243f:6a88:85a3:08d3:1319:8a2e:0370:7344

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                  href                                   │
├──────────┬┬───────────┬─────────────────┬──────────────────────────┬───┤
│ protocol ││   auth    │      host       │           path           │ hash │
│          ││           ├──────────┬──────┼──────────┬───────────────┤     │
│          ││           │ hostname │ port │ pathname │    search     │     │
│          ││           │          │      │          ├─┬─────────────┤     │
│          ││           │          │      │          │ │    query    │     │
"  http:   // user:pass @ host.com : 8080   /p/a/t/h  ?  query=string  #hash "
│          ││           │          │      │          │ │             │     │
└──────────┴┴───────────┴──────────┴──────┴──────────┴─┴─────────────┴─────┘
```

JavaScript

Just-in-time Compiler (JIT)

JS-Plattformen: Web-Browser, Node.js, Deno, Bun

Plattformspezifische APIs im Browser: DOM, Cookies, Storage, etc.

Ein paar Design-Mängel aus den Anfangstagen und grundlegende Änderungen nicht möglich.

JavaScript Engines

V8: Google Chrome / Edge
JavaScriptCore (Nitro): Apple Safari
Spidermonkey: Firefox

Standards

ECMAScript: ES6 (2015), ES7 (2016), ES2020

Zahlen

– **64 Bit Floating Point** entsprechend IEEE 754 (wie double in Java)
  Enthält alle **32 Bit Ganzzahlen**
  → oft Rechenungenauigkeit bei Zahlen mit Nachkommastellen:
  `0.1 // hexadezimal 3FB999999999999A ist nicht exakt 0.1`

Spezielle Zahlen: `Infinity, -Infinity, NaN, BigInt (seit ES2020)`

Hoisting

In JavaScript, a variable can be used before it has been declared.
This only works for variables declared with `var` and `function` declarations.

```
x = 5;            // assign
f(x, 5);          // use
var x;            // declare
function f(x) {}  // declare
assert.strictEqual(typeof f, 'function');
```

Value and Reference Types

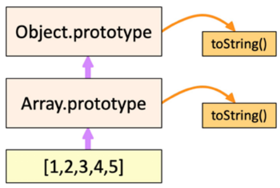
Value (Primitives) Types

`String, Number, Boolean, undefined, null, Symbol and BigInt`
Copy by value on assignment.
Stored directly in the variable's memory location.
== or === compares the values.

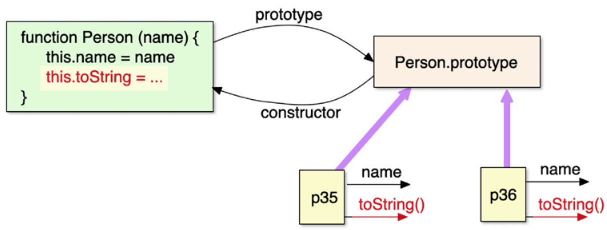
Reference (Object) Types

`new String, new Number, Object, Array and Function`
Copy by reference on assignment. Stores as a reference (or pointer).
== or === will only **compare the references** values (memory locations) and not the values.

Vererbung und Prototypkette

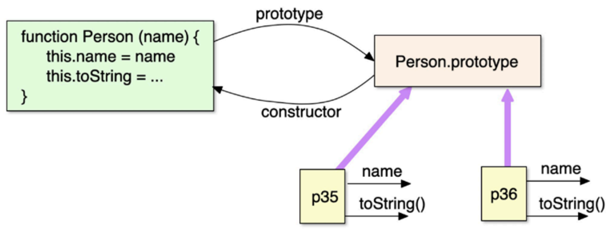


Bad




```
let p35 = new Person(...);
let p36 = new Person(...);
```

Good



Asynchronous JavaScript

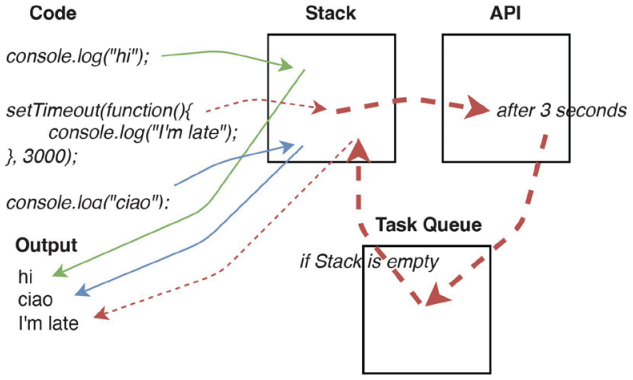
JavaScript is **single-threaded.**

Web Workers are a way to run JavaScript **parallel** in the background. However, Web Workers do not allow access to the DOM.

Event Loop

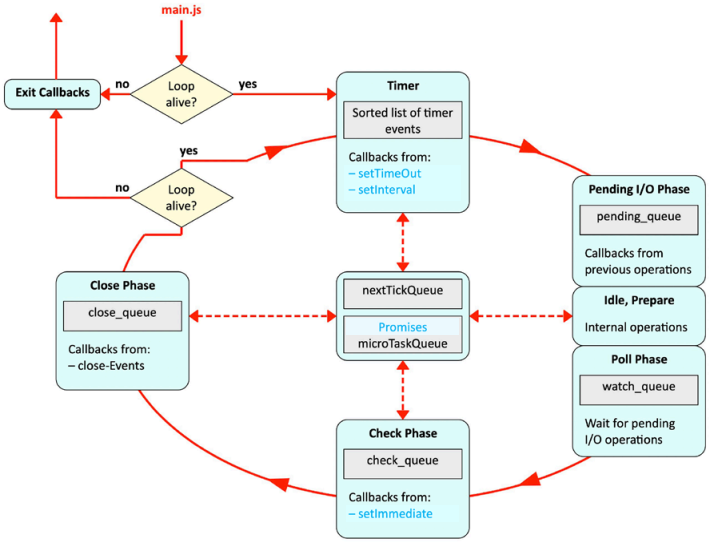
1. Funktionsaufrufe werden zuerst in den **Call Stack** verschoben. Der Stack wird vorzu abgearbeitet.
2. Synchrone Funktionen werden sogleich abgearbeitet. Asynchrone (`setTimeout, async, etc.`) werden in die **Event Queue** verschoben.
3. Sobald der Call Stack leer ist, werden die «paraten» asynchornen Funktionen im **Event Loop** abgearbeitet.
4. Das Programm ist beendet, sobald die Event Queue leer ist.

Vereinfachte Darstellung



Event Loop (node.js)

Hinweis: Die Event Loop im Browser unterscheidet sich.



Prioritäten: nextTickQueue > Promises > immediate > timeout

nextTickQueue / Promises: Höchste Priorität, werden so früh wie möglich abgearbeitet.

Timer: Sortierte Liste nach Fälligkeit. Eintrag in die Timer-Liste auch bei einem Timeout bei 0. `setTimeout(f, 0)` wird also nie sofort ausgeführt.

Pending I/O: Aufgeschobene Callbacks, e.g. Systemoperationen wie Fehlermeldungen bestimmter TCP-Aufrufe.

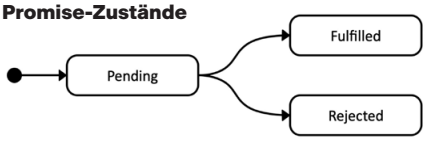
```
// Add timeout to the timer-list in the event loop.
setTimeout(() => {
  // This should be called after 100 ms, but... */
}, 100);

runCallbackAfter95ms(() => {
    // timeout is not ready yet, call the next function in line
    thisTakes10ms(); // blocks the event loop
}); // blocks the event loop

// timeout is ready now and finally called after 105 ms.
```

Promises (async/await)

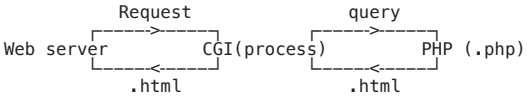
Promise-Zustände



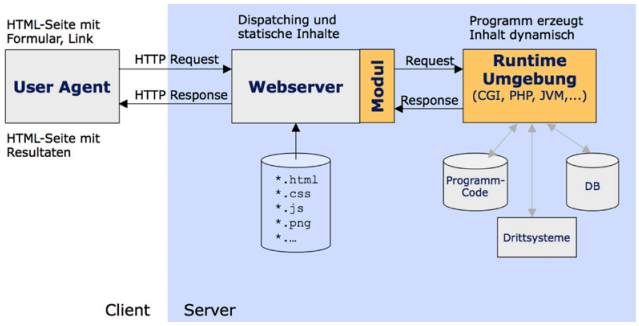
Web Server

CGI (Common Gateway Interface)

Enables web servers to execute an external program (e.g. PHP) to process HTTP or HTTPS user requests. **Problem:** Creates (and destroys) one (CGI) process for each request.



FastCGI can handle more than one request separately from the Web server as a seperate process. Each process listens on a socket. The Web server sends requests to that socket only for dynamic content, while static content is usually handled directly by the Web server.



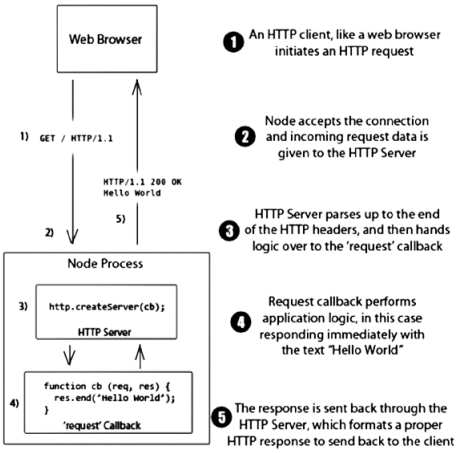
HTTP Requests Methods

GET: Ressource anfordern. Daten werden in der URL gesendet.
POST: Ressource senden. Daten werden im Body gesendet.
PUT: Ressource anlegen/aktualisieren. Daten werden im Body gesendet.
DELETE: Ressource löschen.
PATCH: Ressource teilweise aktualisieren

HTTP Status Codes

1xx: Informational, z.B: 101 Switching Protocols
2xx: Success, z.B: 200 OK, 204 No Content
3xx: Redirection, z.B: 301 Moved Permanently, 304 Not Modified
4xx: Client Error, z.B: 400 Bad Request, 401 Unauthorized, 404 Not Found
5xx: Server Error, z.B: 500 Internal Server Error, 502 Bad Gateway

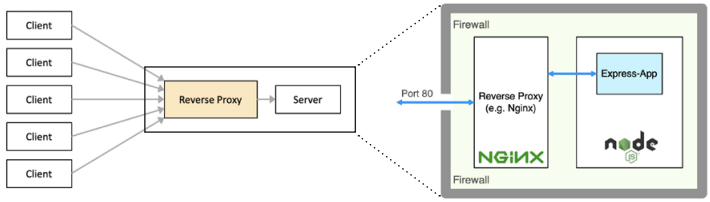
Node Web Server



A **forward proxy** acts on behalf of clients to manage and secure their requests to external servers, while a **reverse proxy** acts on behalf of servers to handle incoming requests from clients, optimizing performance and enhancing security. Essentially, forward proxies protect client identities, whereas reverse proxies protect server identities.

Express.js and Reverse Proxy

Express-App wird über Reverse Proxy, z.B. ein nginx-Server, angesprochen. Dieser leitet Anfragen an die Express-App weiter.



JavaScript in the Browser

HTML-Elemente haben den `nodeType` 1.

NodeType	Konstante	Bedeutung
1	Node.ELEMENT_NODE	Elementknoten
3	Node.TEXT_NODE	Textknoten
8	Node.COMMENT_NODE	Kommentarknoten

Attribut `node.childNodes` ist eine NodeList (Array-like, non-iterable),
e.g. `[ #text, h1, p ]`

Attribut `node.children` ist ein HTMLCollection (Array-like object)
und enthält nur untergeordnete Elementknoten,
e.g. `{ 0: h1, 1: p, length: 3 }`

Client-Server

Cookies

Werden hauptsächlich dafür verwendet, wenn der **Server** Informationen über den Client speichern möchte. Zugriff auf Cookies ist mit JavaScript nur möglich, wenn `HttpOnly` nicht gesetzt ist.

Möchte der **Server** ein Cookie setzen, so sendet er folgenden **HTTP-Response** Header:

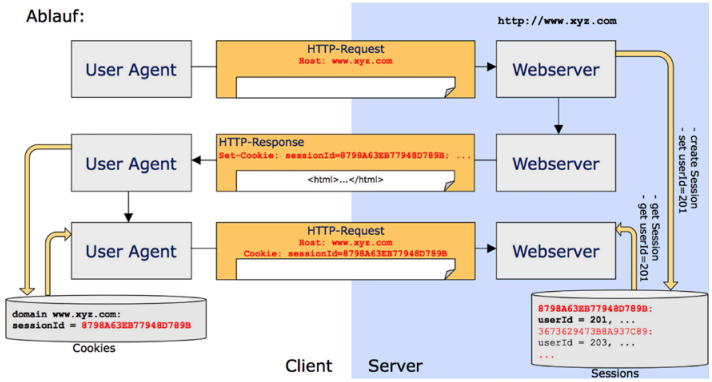
```
Set-Cookie: name=value; expires=Fri, 31 Dec 9999 23:59:59 GMT;
```

Der **Client** speichert das Cookie und sendet es im Header bei jedem **HTTP-Request:**

```
Host: example.com
Cookie: name=value
```

Cookie und SessionId

Problem: Cookies sind leicht manupulierbar (vom Client aus). Stattdessen werden keine Daten im Cookie gespeichert sonden eine eindeutige **SessionId** verwendet.



Frameworks

Kontrolle liegt beim Framework.
Inversion of Control (auch «Hollywood-Prinzip»): «Don't call us, we'll call you.»

MVC (Model-View-Controller)

```
Model <—> Controller <—> View
```

Model: Räpresentiert die Daten. UI-unabhängig. Informieren Observers (e.g. Controller) über Zustandsänderungen. Model wird vom Controller aktualisert.

View: User-Oberfläche, reflektiert Model visuel. Gibt User-Eingaben (z.B. Clicks) an Observers (e.g. Controller) weiter.

Controller: Verarbeitet Eingaben und aktualisieren Models. Verbindet View und Model miteinander. View und Model wissen aber grundsätzlich nichts voneinander.

SPA (Single Page Application)

Ziel: Neuladen von Seiten vermeiden.

Teil-Inhalte werden dynamisch geladen, wenn sie benötigt werden. Kommunikation mit Server im Hintergrund.

CSS-Grundlagen (Cascading Style Sheet)

Zeichencodierung

```
@charset "utf-8";
```

Selektoren

```
p.info.error { ... }
<p class="info error">...</p>
```

Child-Combinator

`p em:` alle `em` irgendwo im Baum unter einem `p`
`p > em:` alle `em` direkt unter einem `p`

Adjacent Sibling Combinator

`h2 + p:` alle `p` unmittelbar folgend auf ein `h2` (= Geschwisterelement)
`li + li + li:` Alle `li` ab dem dritten `li` in der Liste

Attributselektoren

`E[foo]` Alle `E` mit Attribut `foo`
`E[foo=value]` ... das den Wert `value` hat
`E[foo~=value]` ... in dessen Wert das Wort `value` vorkommt
`E[lang|=en]` ... das mit `en` beginnt, gefolgt von einem Bindestrich und dem Rest des Attributwerts.

Pseudoklassen

Elemente, die in einem bestimmten Zustand sind.
`a:link, a:visited, a:hover, a:focus, a:active`

Pseudoelemente

Sie beziehen sich auf Bereiche im HTML-Dokument, die nicht explizit durch Elemente ausgezeichnet sind.

`::first-line, ::first-letter`
`::before, ::after`: Eine Position vor oder nach dem Inhalt eines Elements.
Bps: `a::after { content: " [" attr(href) "]"; }`

Box-Modell

box-sizing

`content-box:` **`width`** und **`height`** beziehen sich nur auf den **Inhalt** (Standard)
`border-box:` **`width`** und **`height`** beziehen sich auf **Inhalt + Padding + Border**

In JavaScript:

`clientWidth` und **`clientHeight`** beziehen sich auf **Inhalt + Padding**
`offsetWidth` und **`offsetHeight`** beziehen sich auf **Inhalt + Padding + Border**

`clientTop` und **`clientLeft`** beziehen sich auf die **Breite des Rahmens**
`offsetTop` und **`offsetLeft`** beziehen sich auf die **Position des Elements**

Wenn wir in CSS den Standard-Abstand des body-Elements zurücksetzen (zum Beispiel mit einem Reset-Stylesheet, s. letzte Lektion) und die section einfärben, um die Grösse sehen zu können, ist das Resultat wie in der Abbildung 62 gezeigt. Die Box nimmt die gesamte verfügbare Breite und die benötigte Höhe ein. Nun ergänzen wir Breiten- und Höhenangaben.

```
/* css */
width: 300px;
padding: 10px;

/* js */
clientWidth = 320 // = 300 + 2·10
offsetWidth = 330 // = 300 + 2(10 + 5)
```

Wenn wir in CSS den Standard-Abstand des body-Elements zurücksetzen (zum Beispiel mit einem Reset-Stylesheet, s. letzte Lektion) und die section einfärben, um die Grösse sehen zu können, ist das Resultat wie in der Abbildung 62 gezeigt. Die Box nimmt die gesamte verfügbare Breite und die benötigte Höhe ein. Nun ergänzen wir Breiten- und Höhenangaben.

```
/* css */
box-sizing: border-box;
width: 300px;
padding: 10px;

/* js */
clientWidth = 290 // = 300 – 2·10
offsetWidth = 300
```

Variablen

Variablen beginnen mit `--` und können mit `var()` wieder referenziert werden.
Die Pseudo-klasse `:root` steht für die Wurzel des HTML-Dokuments, die definierten Variablen sind also im ganzen Dokument nutzbar.

```
:root {
  --main-fg-color: darkblue;
  --main-bg-color: #fffacd;
}

footer {
  color: var(--main-fg-color);
  background-color: var(--main-bg-color, white);
}
```

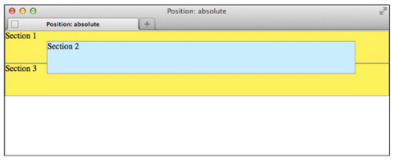
Positionierung

static
Standardeinstellung. Block-Elemente werden untereinander angezeigt. Inline-Elemente nebeneinander.

relative
Ein relativ positioniertes Element kann von seiner Normalposition verschoben werden. Sein ursprünglicher Platz im Elementfluss bleibt dabei frei.


```css
#section2 {
    position: relative;
    top: 20px; left: 80px;
    width: 80%;
    background-color: #cef;
}
```

absolut
Elemente werden **aus dem Elementfluss herausgenommen** und entsprechend der Angaben top, left, bottom or right (innerhalb des Parents) positioniert.




```css
#section2 {
    position: absolute;
    top: 20px; left: 80px;
    width: 80%;
    background-color: #cef;
}
```

fixed
Position bezieht sich auf den Rand des Viewports.

Fliessende Boxen (float)

Das Element wird **aus dem Elementfluss herausgehoben,** nachfolgende Elemente rücken dadurch nach oben.

clear (none, both, left, right) bricht das Umfliessen ab.

Display

inline: Breite und Höhe wird ignoriert. `span`, `a`, `img`, `strong`, `cite`, etc.
block: Breite und Höhe wird berücksichtigt. `div`, `p`, `h1`,`blockquote`, etc.