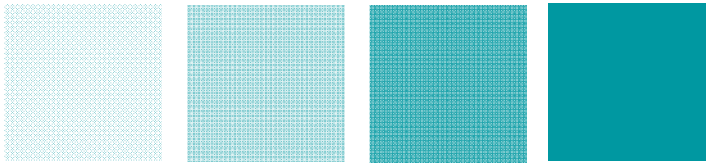
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences



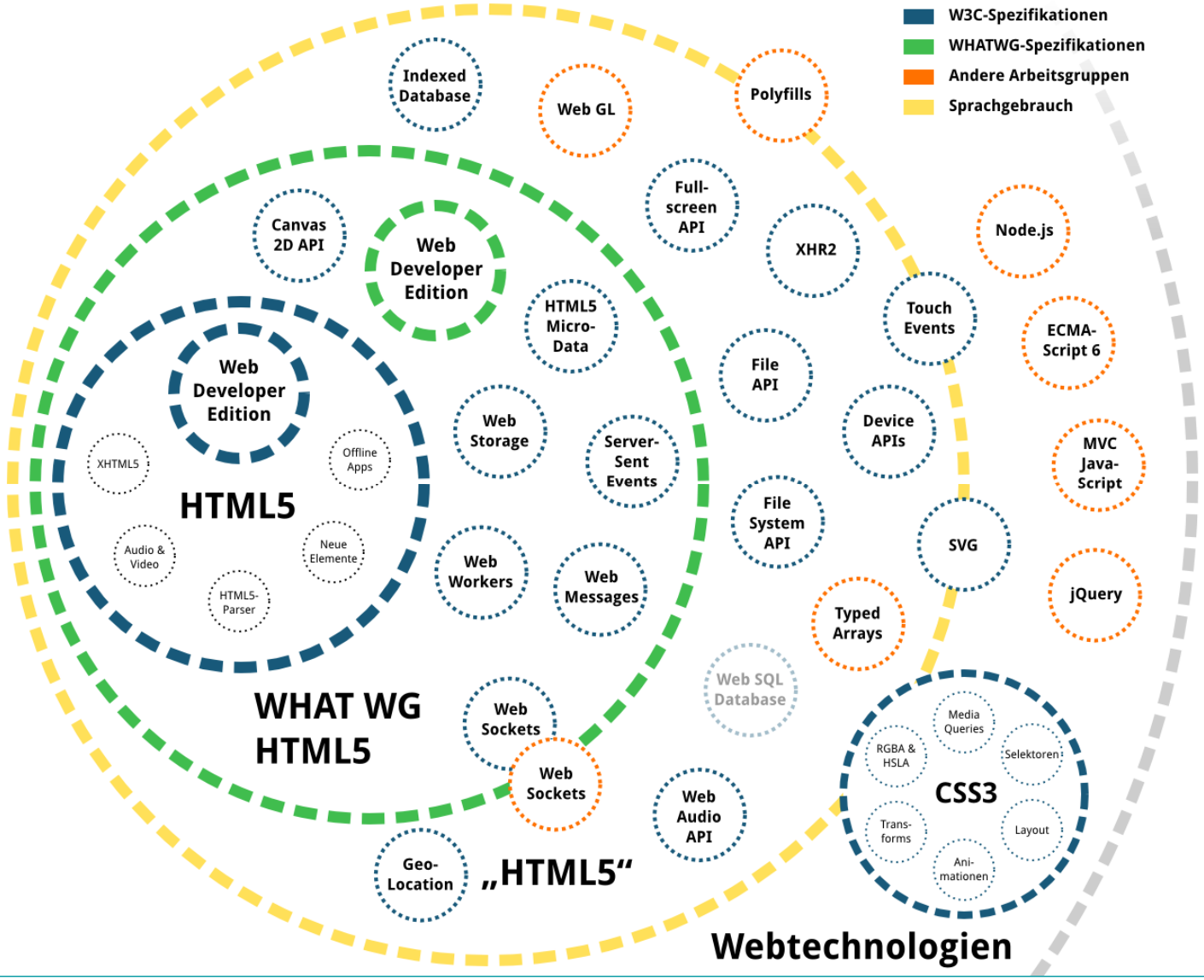
HTML 5 + JS + jQuery

Multimedia Engineering II
WS 13/14

Sven Spielvogel

HTML5

Quelle: http://www.joocom.de/blog/html5-ein-uberblick/



Die Basis

```
<!DOCTYPE html>
```
➡ definiert HTML5 als doctype

```
<html>
```
➡ beschreibt die eigentliche Website

```
<body>
```
➡ sichtbarer Inhalt der Seite

```
<h1>Hello World!</h1>
```
➡ Definition einer Überschrift

```
<p>I'm a paragraph without any
sense.</p>
```
➡ Definition eines Absatzes

```
</body>
</html>
```

Integration

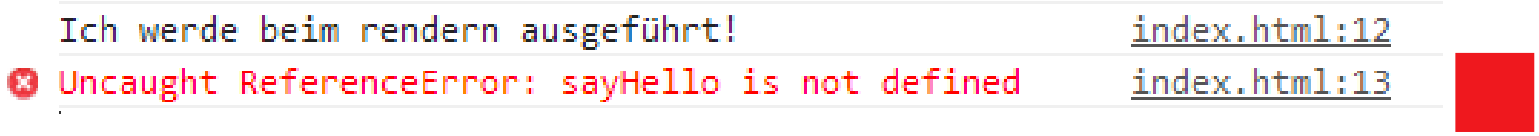
HTML und JS/CSS

JS

```
<!DOCTYPE html>
    <head>
   <meta charset="utf-8">
   <title>Demo</title>
   <meta name="description" content="demo,mme2">

   <script src="js/vendor/modernizr-2.6.2.min.js"></script>
    </head>
<html>
    <body>

   <p onclick="console.log('Hello World!')">Hello world! This a MME2 demo!</p>

   <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
   <script src="js/main.js"></script>
   <script>
  console.log("Ich werde beim rendern ausgeführt!")
   </script>
    </body>
</html>
```

- **<head>:** wird vor dem Rendern ausgeführt → kein DOM
- **<body>:** wird während des Renderns ausgeführt
  - Reihenfolge beachten! Es kann auf nichts zugegriffen werden, was noch nicht geladen wurde!
  - Bsp:

```
<body>
  <script>
 console.log("Ich werde beim rendern ausgeführt!");
 console.log(sayHello());
  </script>
  <script>
 function sayHello(){return "Hello";}
  </script>
</body>
```



JS im Head

```
<head>
  <meta charset="utf-8">
  <title>Demo</title>
  <meta name="description" content="demo,mme2">

  <script src="js/vendor/modernizr-2.6.2.min.js"></script>
</head>
```

➡ extern

JS im Body

inline

```
<body>
   <p onclick="console.log('Hello World!')">Hello world! This a MME2
demo!</p>

   <script src="http://url/jquery.min.js"></script>
   <script src="js/main.js"></script>

   <script>
 console.log("Ich werde beim rendern ausgeführt!");
   </script>
</body>
```

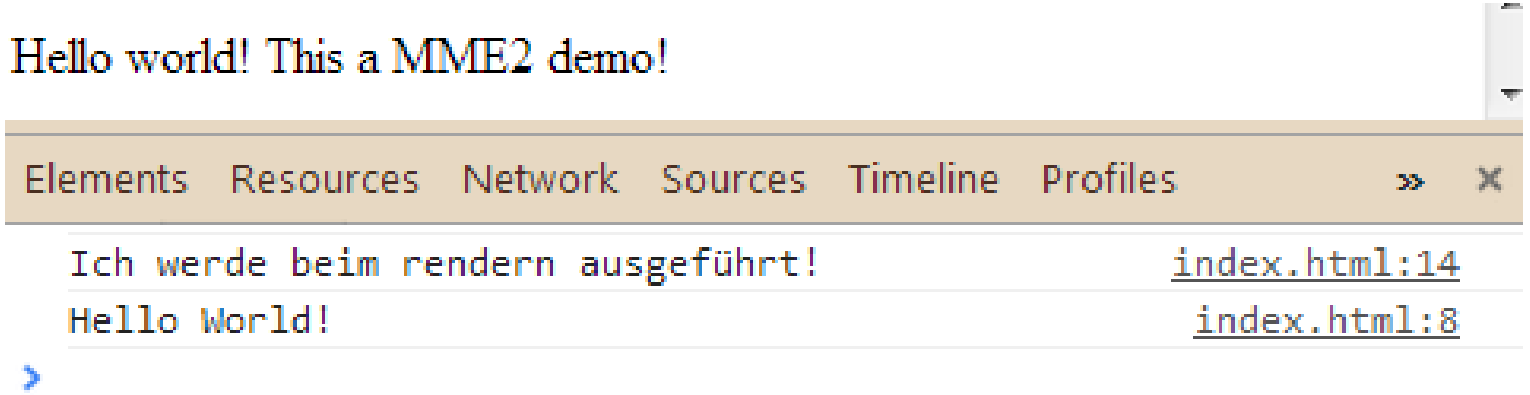
extern

intern

JS

```
<!DOCTYPE html>
    <head>
        <meta charset="utf-8">
        <title>Demo</title>
        <meta name="description" content="demo,mme2">
    </head>
    <body>
        <p onclick="console.log('Hello World!')">Hello world! This a MME2 demo!</p>

        <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.j

        <script src="js/main.js"></script>
        <script>
            console.log("Ich werde beim rendern ausgeführt!");
        </script>
    </body>
</html>
```

JS


Hello world! This a MME2 demo!
Elements Resources Network Sources Timeline Profiles
Ich werde beim rendern ausgeführt! index.html:14
Hello World! index.html:8

CSS

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Demo</title>
    <meta name="description" content="demo,mme2">
    <link rel="stylesheet" href="css/normalize.css"> <!-- wird zuerst geladen und angewandt-->
    <link rel="stylesheet" href="css/main.css"> <!-- wird im anschluss geladen und kann Werte aus
vorherigem CSS überschreiben -->
    <style>
   body{
  background-color: #000000;
   }
   p{
  color: #f0f0f0;
   }
   h1,h2,h3{
  color: #ffff00;
   }
    </style>  <!-- wird zuletzt geladen und angewandt-->
</head>
<html>
    <body>
   <h1 style="color: #105cb6">Hello World!</h1>
   <p>I’m a paragraph without any sense.</p>
    </body>
</html>
```

CSS im Head

```
<head>
    ...
    <link rel="stylesheet" href="css/normalize.css">
    <link rel="stylesheet" href="css/main.css">
     <style>
   body{
  background-color: #000000;
   }
   p{
  color: #f0f0f0;
   }
   h1,h2,h3{
  color: #ffff00;
   }
    </style>  <!-- wird zuletzt geladen und angewandt-->
</head>
```

extern

intern

CSS inline

inline

```
<html>
    <body>
   <h1 style="color: #105cb6">Hello World!</h1>
   <p>I’m a paragraph without any sense.</p>
    </body>
</html>
```

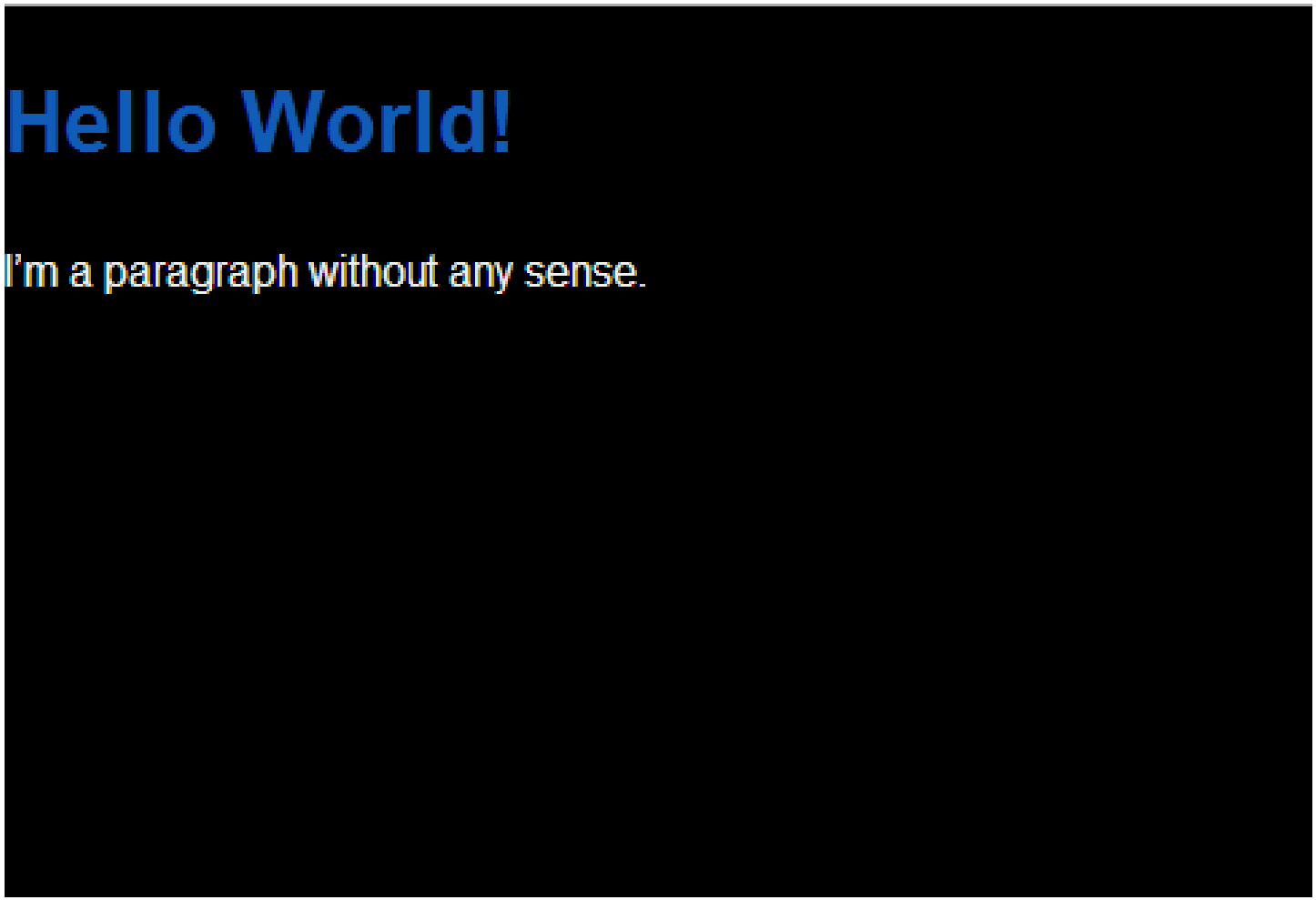
CSS

```
<!DOCTYPE html>
<head>
    ...
    <style>
        body{
            background-color: #000000;
        }
        p{
            color: #f0f0f0;
        }
        h1,h2,h3{
            color: #ffff00;
        }
    </style>  <!-- wird zuletzt geladen und angewandt-->
</head>
<html>
    <body>
        <h1 style="color: #105cb6">Hello World!</h1>
        <p>I'm a paragraph without any sense.</p>
    </body>
</html>

```

CSS

```
<!DOCTYPE html>
<head>
    ...
    <style>
        body{
            background-color: #000000;
        }
        p{
            color: #f0f0f0;
        }
        h1,h2,h3{
            color: #ffff00;
        }
    </style>  <!-- wird zuletzt geladen und angewandt-->
</head>
<html>
    <body>
        <h1 style="color: #105cb6">Hello World!</h1>
        <p>I'm a paragraph without any sense.</p>
    </body>
</html>

```

CSS



CSS – Hierarchie

```
<!DOCTYPE html>
<head>
    ...
    <style>
        body{
            background-color: #000000;
        }
        p{
            color: #f0f0f0;
        }
        h1,h2,h3{
            color: #ffff00;
        }
    </style>  <!-- wird zuletzt geladen und angewandt-->
</head>
<html>
    <body>
        <h1 style="color: #105cb6">Hello World!</h1>
        <p>I'm a paragraph without any sense.</p>
    </body>
</html>
```

1. Inline Style
2. Internal style sheet
    1. #id type.class
    2. #id type
    3. #id
    4. type .class
    5. type
3. Externalstyle sheet
4. Browser default

JavaScript

JS – kurz nochmal die Basis

Primitive Typen

- **`null`** → `var n=null;`
- **`undefined`** → `var u;`
- Zahl → `var z=1; var f=6.4;`
- Wahrheitswert → `var b=true;`
- Zeichenkette → `var s="Hello"; var s='World';`
  - Zwei Möglichkeiten, String anzugeben! Warum? Bsp:
  - `<p onclick="console.log('Hello World!')">Hello world! This a MME2 demo!</p>`

JS – kurz nochmal die Basis

Objekt Typen

- **Object** → `var o=new Object(); var o={};`
- **Array** → `var a=new Array(); var a=[];`
- **Function** → `var f=function(){...};`
- Number → `var n=new Number(3);`
- Boolean → `var b=new Boolean(true);`
- String → `var s=new String("Hello");`
- RegExp → `var r=new RegExp(patt,modi);`
- Date → `var now=new Date();`

JS – Typisierung

- Implizite Typisierung: Typ muss nicht deklariert werden.

```
var x = 5.6;
```

- Schwache Typisierung: keine strenge Prüfung, viele implizite Umwandlungsregeln

```
var x = 4 + 3.2 + “7.3“;
```
→ Ergebnis: “7.27.3“

- Dynamische Typisierung: Typ kann erst zur Laufzeit bekannt werden

```
var x;
if (mayReturnTrueOrFalse()){
  x = 5;
} else {
  x = “hello world“;
};
```
→ Ergebnis: x = Number(5)

→ Ergebnis: x = String(“hello world“)

JS – Funktionen

```
function f(a,b,c) {
  return (a+b)*c;
}
```

- `f` - Funktionsname
- `a,b,c` - Parameter
  - Für Parameter wird kein Typ deklariert.
  - Parameter verhalten sich in der Funktion wie eine lokale Variable.
  - Parameter, zu denen kein Argument übergeben werden, sind `undefined`
- Aufruf:

```
var x = f(1,2,3);
```

JS – Funktionen

```
function f(a,b,c) {
  return (a+b)*c;
}
var x = f(1,2,3);
```

99% äquivalent

```
var f = function(a,b,c) {
 return (a+b)*c;
};
var x = f(1,2,3);
```

Objekt Typen

- . . .
- **Function** → `var f=function(){...};`
- . . .

JS – Funktionen

```
function f(a,b,c) {
  return (a+b)*c;
}
var x = f(1,2,3);
```

99%
äquivalent

```
var f = function(a,b,c) {
 return (a+b)*c;
};
var x = f(1,2,3);
```

- Der Unterschied liegt in der Verfügbarkeit im **Kontext** beim laden

JS – Funktionen

```
<script>
var x = f(1,2,3); ok!
function f(a,b,c) {
  return (a+b)*c;
}
</script>
```

- wird beim parsen initialisiert/verfügbar

```
<script>
var x = f(1,2,3); FEHLER!
var f = function(a,b,c) {
 return (a+b)*c;
};
</script>
```

- wird erst zur Laufzeit initialisiert/verfügbar

JS – Kontext

- Mit Kontext ist der aktuell Verfügbare „Funktionskatalog“ gemeint
  - **`this – lokal`**
  - **`window – global/gilt immer`**

JS – Kontext - window

```
<script>
        console.log(this);
        function topLevel(){
                console.log(this);
        }
        this.topLevel();
        window.topLevel();
        topLevel();
</script>
```

```
▶ Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}   index.html:14
▶ Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}   index.html:16
▶ Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}   index.html:16
▶ Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}   index.html:16
>
```

JS – Kontext - this

```
<script>
        function topLevel(){
                console.log(this);
        }
        document.getElementById("textClick").onclick =
                function(e){
                        console.log(this); ok!
                        window.topLevel(); ok!
                        topLevel(); ok!
                        this.topLevel(); FEHLER!
                };
</script>
```

```
<p id="textClick" onclick="console.log('Hello World!')">Hello world! This a MME2 demo!</p>   index.html:18
▶ Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}   index.html:15
▶ Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}   index.html:15
⊗ ▶Uncaught TypeError: Object #<HTMLParagraphElement> has no method 'topLevel'   index.html:21
>
```

JS – Kontext

- Der Kontext wird uns im späteren Verlauf nochmal begegnen, wenn wir mit inneren Funktionen mit Backbone arbeiten

JS – Closure

```
var f1 = function () {       // eine äußere Funktion f1 definieren ...
    var wert = 22;           // ... und darin einen Namensraum erstellen.
    var f2 = function () {   // eine innere Funktion definieren, ...
        return wert;         // ... die den Namensraum nach außen reicht.
    }
    return f2;    // f2 durch f1 zurückgeben, womit f2 zum closure wird.
}
var a = f1(); // a ist die von f1() zurückgegebene closure-Funktion, ...
document.write(f1() +"<br />"); // ... also: function() {return wert;}
document.write(typeof wert +"<br />"); // ist undefined
document.write(a()+"<br />"); // ergibt 22
document.write(f1()() +"<br />"); // ergibt 22, f2() ist hier aber nicht abrufbar
```

http://de.wikipedia.org/wiki/Closure#JavaScript


jQuery
write less, do more.

jQuery

- Freie JavaScript Bibliothek
- Zwei parallele Versionen

	Internet Explorer	Chrome	Firefox	Safari	Opera
jQuery 1.x	6+	(Current - 1) or Current	(Current - 1) or Current	5.1+	12.1x, (Current - 1) or Current
jQuery 2.x	9+				

jQuery

- Selector-Engine → Basis
- Zahlreiche Hilfsfunktionalitäten (z.B. each)
- AJAX
- Effekte und Animationen
- DOM-Manipulation
- Umfassendes Event-Handling

- PlugIn-Schnittstelle

jQuery

- Kurzer Rückblick
  - Definition einer Funktion?

→ `function myFunction(parameter){..}`

  - Aufruf einer Funktion?

→ `myFunction(parameter);`

jQuery

- Nutzung von jQuery
  - Aufruf Option 1:

→ `jQuery(selector);`

  - Aufruf Option 2:

→ `$(selector);`

```
function jQuery(parameter){..}
```

äquivalent

```
function $(parameter){..}
```

jQuery

Selector Engine

jQuery – Selector Engine

- Sizzle Selector Engine
- http://sizzlejs.com/

- `$(selector)`
- `$("#id")`
- `$(".class ")`
- `usw.`

jQuery – Selector Engine

- → API Documentation - http://api.jquery.com/category/selectors/

- `*` → Alles
- `.class` → Klasse
- `tag` → Tag z.B. `div, h1, p,` usw.
- `#id` → ID des Elements

jQuery – Selector Engine

- Es ist auch wie im CSS möglich, die Selektoren zu kombinieren und verschachteln

Multiple Selektoren

- `$("selector1,selector2,selectorN")`
- `$("#myButton,.labels,div")`

Kombinierte Selektoren

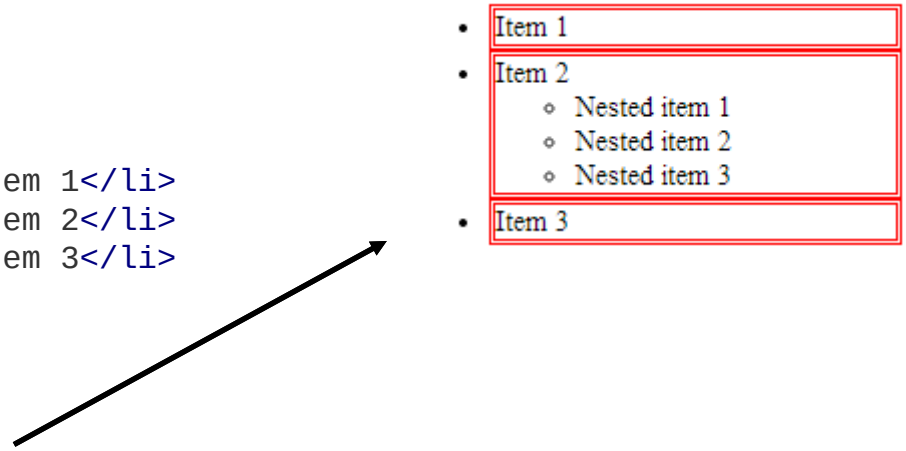
- `$("selector1selektor2")`
- `$("div.labels")`

jQuery – Selector Engine

Verschachtelte Selektoren/Child Selector

- `$("parent > child")`
- `$("ul.topnav > li")`

```
<!doctype html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>child demo</title>
          <style>
               body {
                    font-size: 14px;
               }
          </style>
          <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
     </head>
     <body>
          <ul class="topnav">
               <li>Item 1</li>
               <li>Item 2
                    <ul>
                         <li>Nested item 1</li>
                         <li>Nested item 2</li>
                         <li>Nested item 3</li>
                    </ul>
               </li>
               <li>Item 3</li>
          </ul>
          <script>
          $( "ul.topnav > li" ).css( "border", "3px double red" );
          </script>
     </body>
</html>
```



jQuery

- Wie bei jedem Framework bzw. jeder Bibliothek, sollte man genau wissen, ob Kosten/Nutzen im Verhältnis stehen

  http://jsperf.com/jquery-vs-native-dom-selectors

- Nachteile:
  - Performanceverlust
  - zusätzliche Fehlerquellen
  - Ggf. mangelnde Komptabilität
  - Speicher
- Vorteile:
  - Zeitgewinn in der Entwicklung → geringere Kosten
  - Bessere Lesbarkeit bei gute Dokumentation der(s) Bib/FW

Funktionen

folgen nächste Woche!

Vielen Dank für die Aufmerksamkeit!