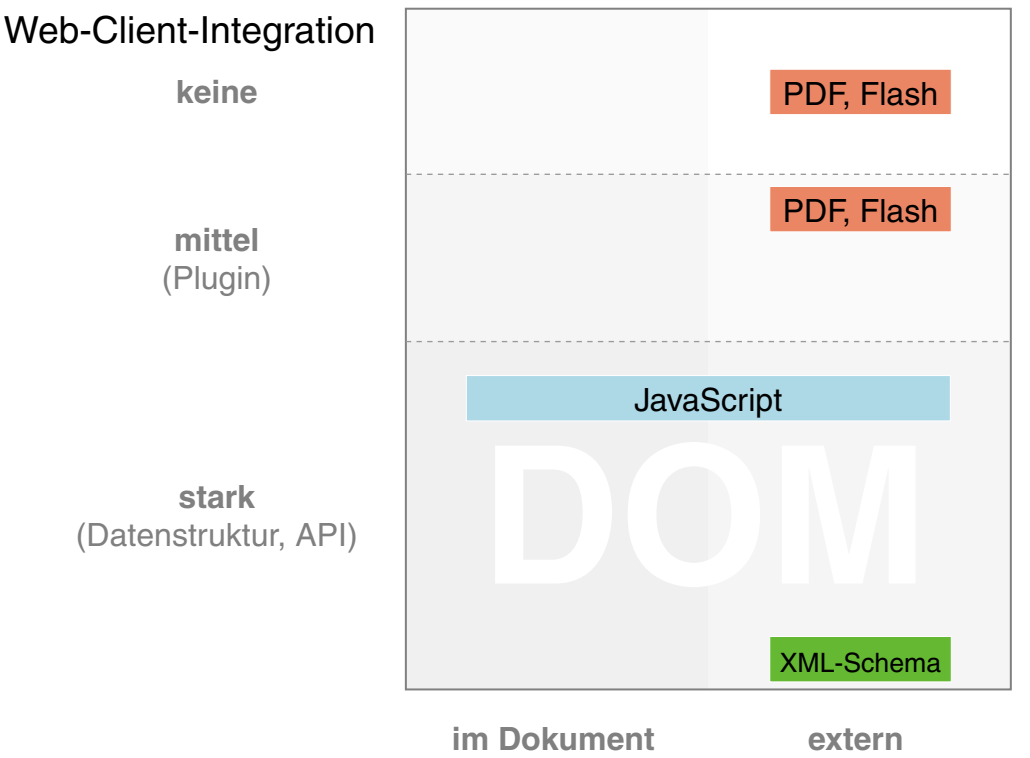
# Kapitel WT:V

## V. Client-Technologien

- Web-Client
- Exkurs: Programmiersprachen
- JavaScript
- VBScript
- Java Applet
- Weitere Client-Technologien

# Web-Client

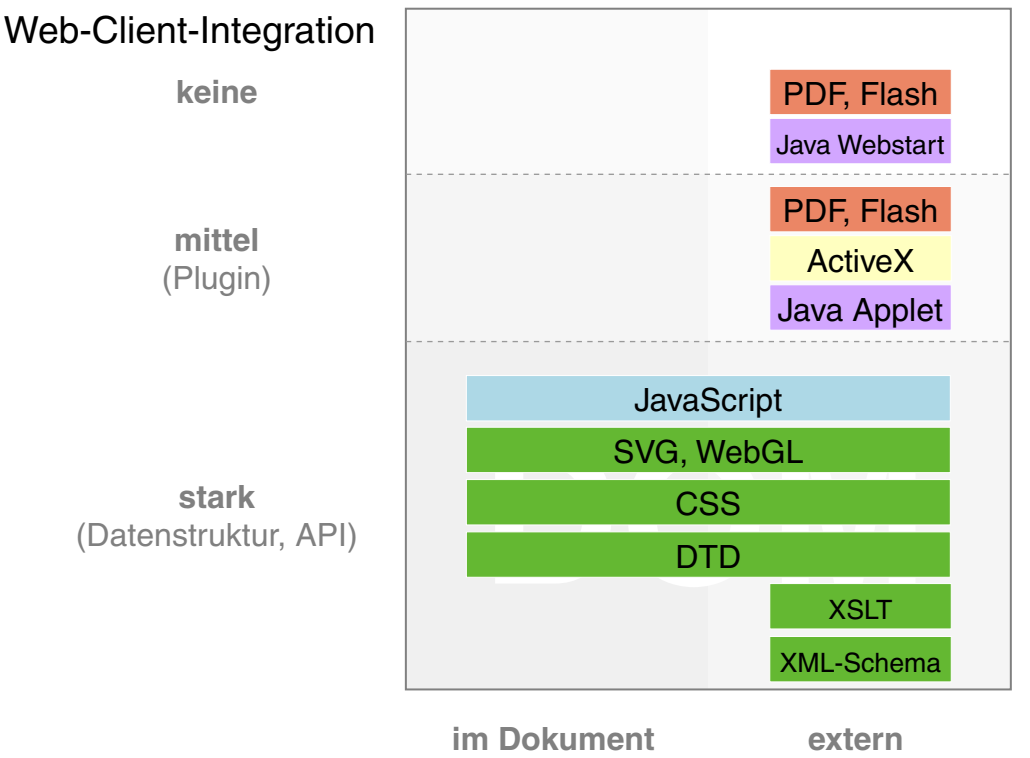
## Einordnung von Client-Technologien [Stein 2012 - 2018]




- x-Achse: Wo befindet sich der Code zur Dokumentdarstellung?
- y-Achse: Wie stark ist die Technologie in den Web-Client integriert?

# Web-Client

## Einordnung von Client-Technologien [Stein 2012 - 2018]



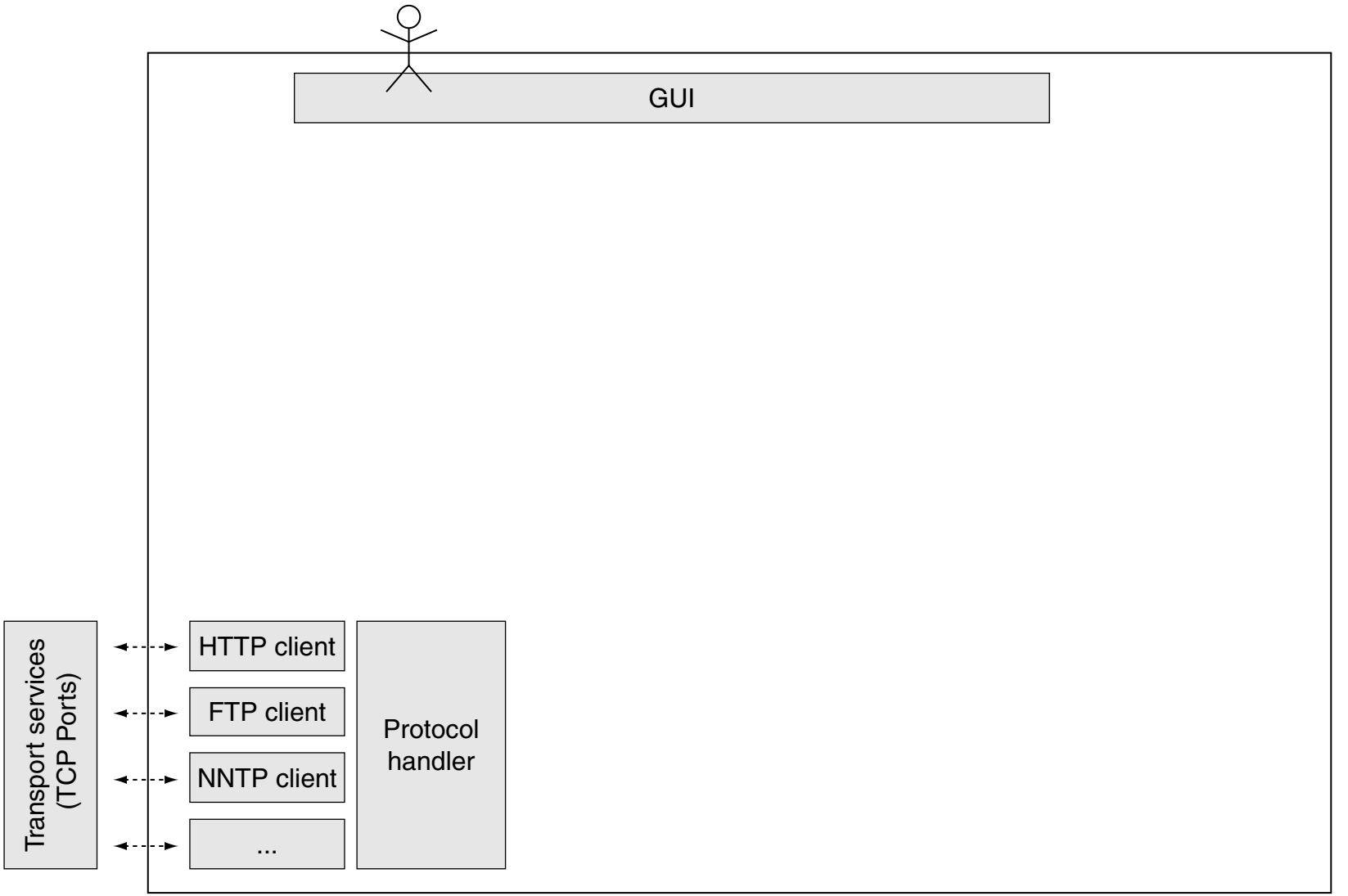

- x-Achse: Wo befindet sich der Code zur Dokumentdarstellung?
- y-Achse: Wie stark ist die Technologie in den Web-Client integriert?

Bemerkungen:

- Client-Technologien dienen zur Realisierung Client-seitig ablaufender Web-Anwendungen.
- Im Vergleich zu Server-seitig ablaufenden Web-Anwendungen erzeugen sie weniger Server-Last und mehr Netzlast.

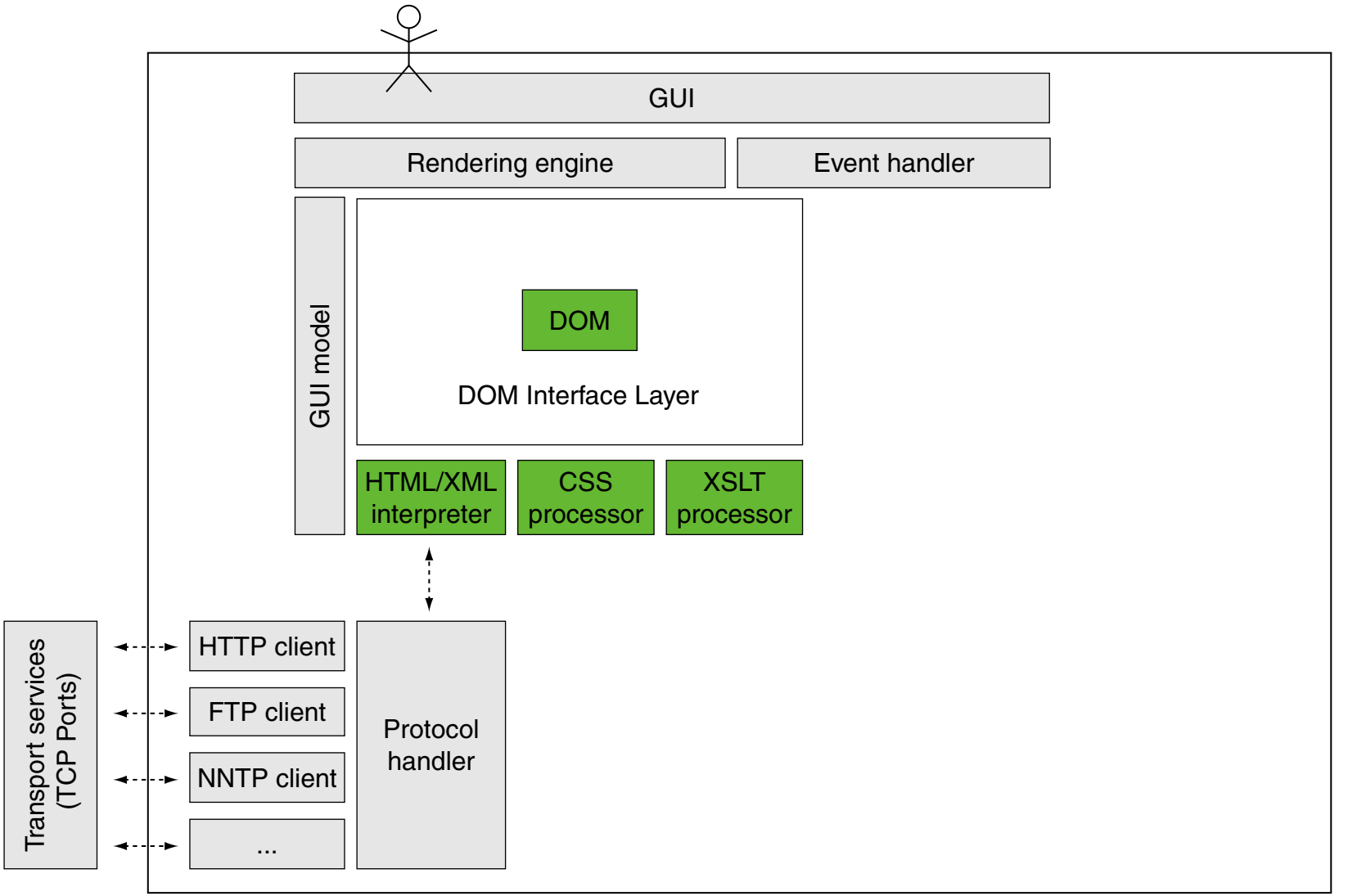
# Web-Client

## Browser-Module [MDN data flow] [W3C parsing model] [how browsers work]



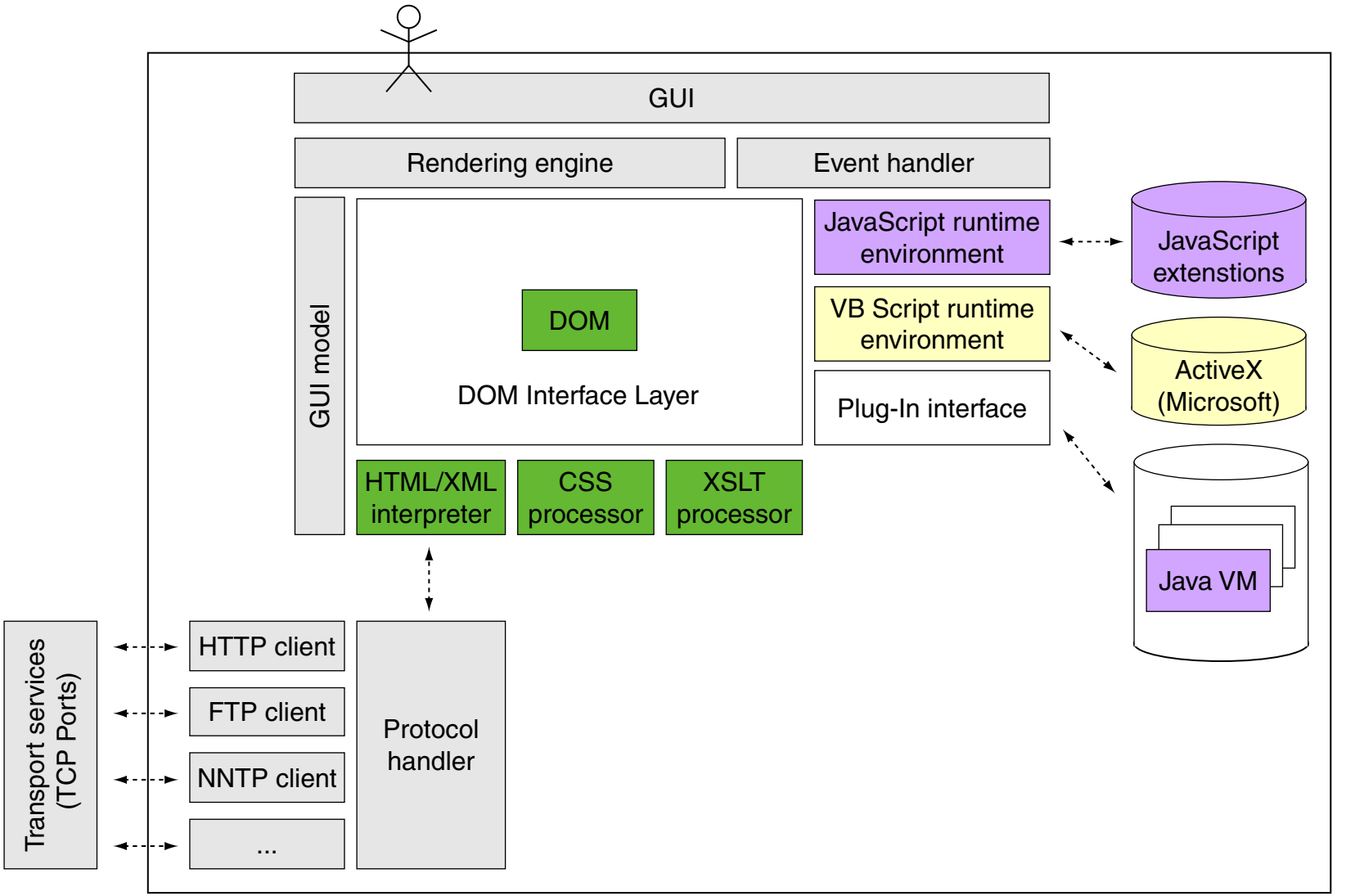
# Web-Client

## Browser-Module [MDN data flow] [W3C parsing model] [how browsers work]

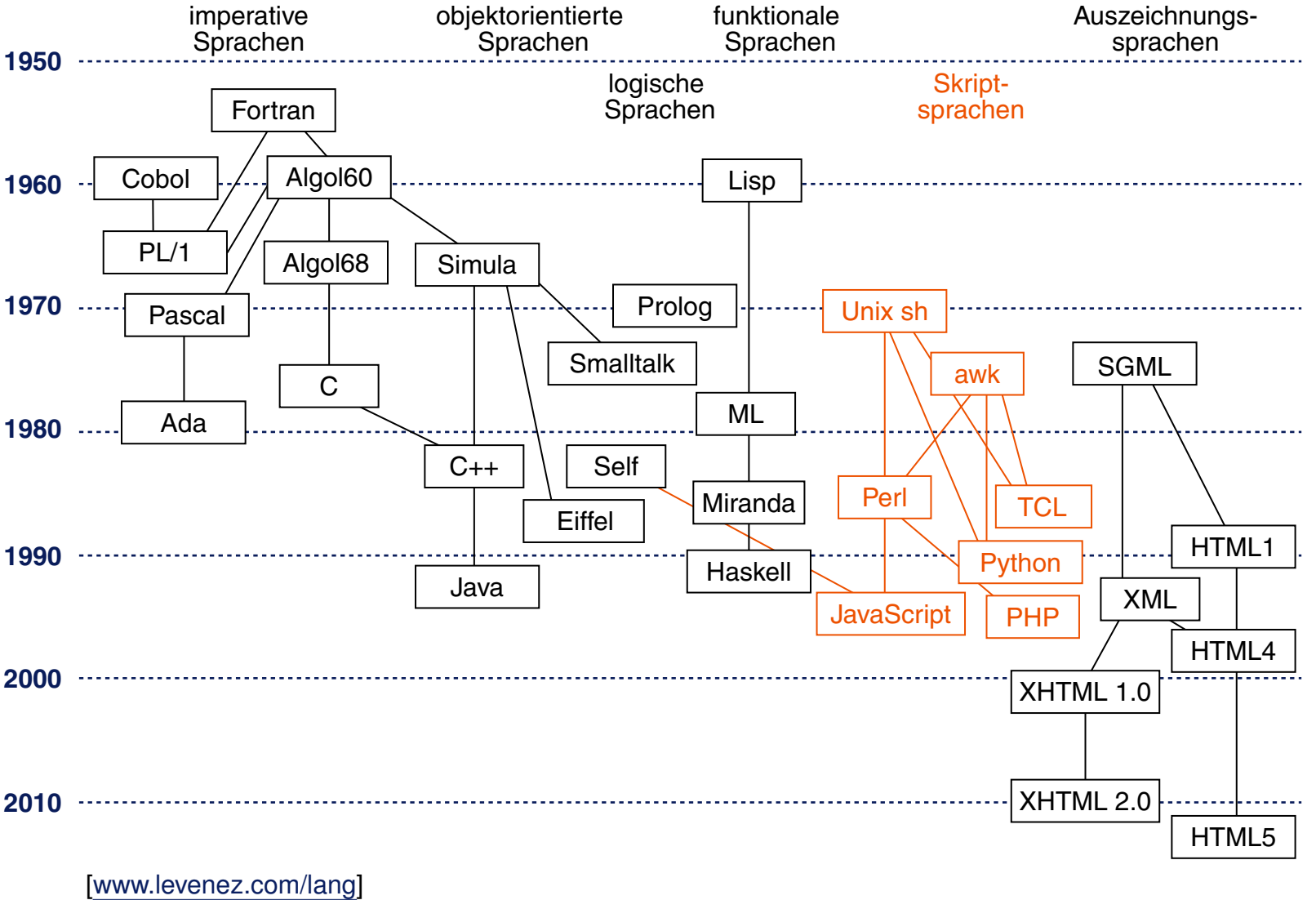


# Web-Client

## Browser-Module [MDN data flow] [W3C parsing model] [how browsers work]



# Exkurs: Programmiersprachen




[www.levenez.com/lang]

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebenen von Spracheigenschaften

Ein Satz einer Sprache ist eine Folge von Zeichen eines gegebenen Alphabets.
Zum Beispiel ist ein PHP-Programm ein Satz der Sprache PHP:

```
$line = fgets ($fp, 64);
```

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebenen von Spracheigenschaften

Ein Satz einer Sprache ist eine Folge von Zeichen eines gegebenen Alphabets.
Zum Beispiel ist ein PHP-Programm ein Satz der Sprache PHP:

```
$line = fgets ($fp, 64);
```

Die Struktur eines Satzes wird auf zwei Ebenen definiert:

1. Notation von Symbolen (Lexemen, Token).
2. Syntaktische Struktur.

Die Bedeutung eines Satzes wird auf zwei weiteren Ebenen an Hand der Struktur für jedes Sprachkonstrukt definiert:

3. Statische Semantik.
   Eigenschaften, die *vor* der Ausführung bestimmbar sind.
4. Dynamische Semantik.
   Eigenschaften, die *erst während* der Ausführung bestimmbar sind.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 1: Notation von Symbolen

Ein Symbol wird aus einer Folge von Zeichen des Alphabets gebildet. Die Regeln zur Notation von Symbolen werden durch reguläre Ausdrücke definiert.

```
$line = fgets ($fp, 64);
```

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 1: Notation von Symbolen

Ein Symbol wird aus einer Folge von Zeichen des Alphabets gebildet. Die Regeln zur Notation von Symbolen werden durch reguläre Ausdrücke definiert.

```
$line = fgets ($fp, 64);
```

Wichtige Symbolklassen in Programmiersprachen:

| **Symbolklasse** | **Beispiel in PHP** |
|---|---|
| Bezeichner *(Identifier)*<br>Verwendung: Namen für Variable, Funktionen, etc. | `$line, fgets` |
| Literale *(Literals)*<br>Verwendung: Zahlkonstanten, Zeichenkettenkonstanten | `64, "telefonbuch.txt"` |
| Wortsymbole *(Keywords)*<br>Verwendung: kennzeichnen Sprachkonstrukte | `while, if` |
| Spezialzeichen<br>Verwendung: Operatoren, Separatoren | `<= = ; { }` |

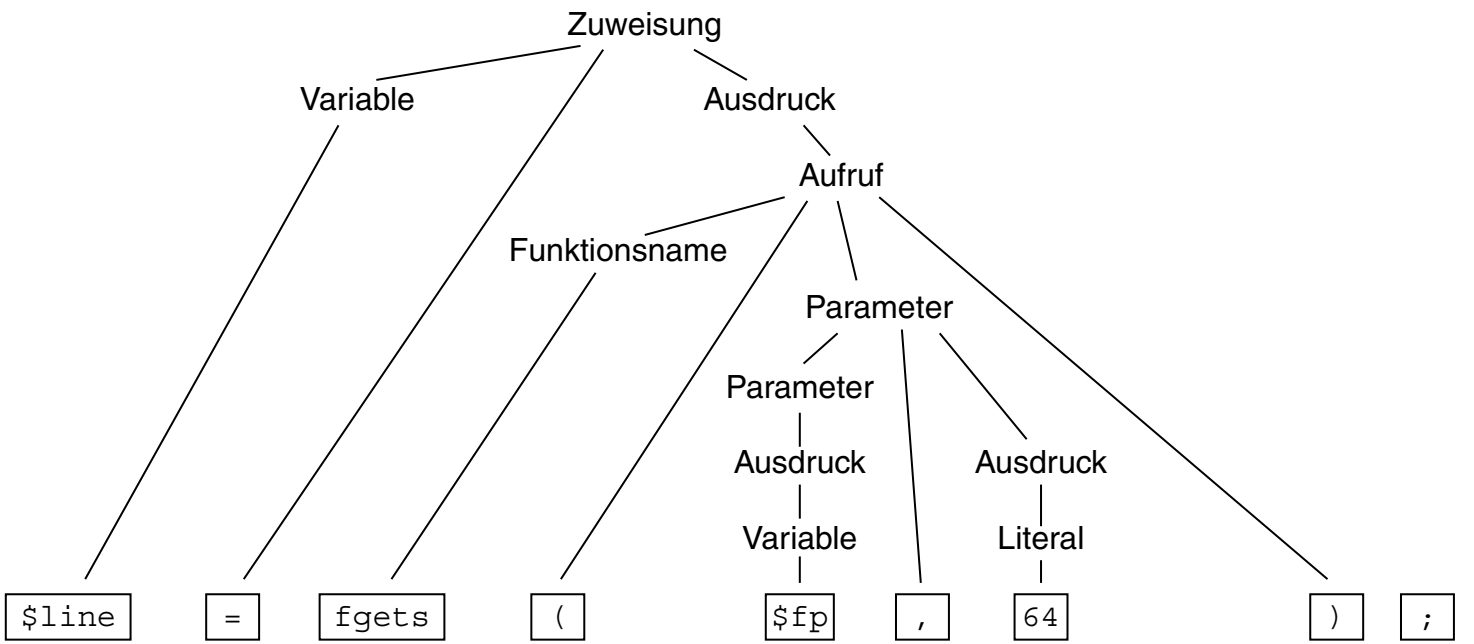
Bemerkungen:

- Zwischenräume, Tabulatoren, Zeilenwechsel und Kommentare zwischen den Symbolen dienen der Lesbarkeit und sind sonst bedeutungslos.
- In Programmiersprachen bezeichnet der Begriff „Literal“ Zeichenfolgen, die zur Darstellung der Werte von Basistypen zulässig sind. Sie sind nicht benannt, werden aber über die jeweilige Umgebung ebenfalls in die Programmressourcen eingebunden. Literale können nur in rechtsseitigen Ausdrücken auftreten. Meist werden die Literale zu den Konstanten gerechnet und dann als literale Konstanten bezeichnet, da beide – im Gegensatz zu Variablen – zur Laufzeit unveränderlich sind.
  Das Wort „Konstante“ im engeren Sinn bezieht sich allerdings mehr auf in ihrem Wert unveränderliche Bezeichner, d.h., eindeutig benannte Objekte, die im Quelltext beliebig oft verwendet werden können, statt immer das gleiche Literal anzugeben. [Wikipedia]

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 2: Syntaktische Struktur

Ein Satz einer Sprache wird in seine Sprachkonstrukte gegliedert; sie sind meist ineinander geschachtelt. Diese syntaktische Struktur wird durch einen Strukturbaum dargestellt, wobei die Symbole durch Blätter repräsentiert sind:




Die Syntax einer Sprache wird durch eine kontextfreie Grammatik definiert. Die Symbole sind die Terminalsymbole der Grammatik.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 3: Statische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Bedeutung (Semantik) beschreiben, soweit sie anhand der Programmstruktur festgestellt werden können, ohne das Programm auszuführen (= statisch).

Elemente der statischen Semantik für übersetzte Sprachen:

- Bindung von Namen.
  Regeln, die einer Anwendung eines Namens seine Definition zuordnen.
  Beispiel: zu dem Funktionsnamen in einem Aufruf muss es eine Funktionsdefinition mit gleichem Namen geben.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 3: Statische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Bedeutung (Semantik) beschreiben, soweit sie anhand der Programmstruktur festgestellt werden können, ohne das Programm auszuführen (= statisch).

Elemente der statischen Semantik für übersetzte Sprachen:

- Bindung von Namen.
  Regeln, die einer Anwendung eines Namens seine Definition zuordnen.
  Beispiel: zu dem Funktionsnamen in einem Aufruf muss es eine Funktionsdefinition mit gleichem Namen geben.

- Typregeln.
  Sprachkonstrukte wie Ausdrücke und Variablen liefern bei ihrer Auswertung einen Wert eines bestimmten Typs. Er muss im Kontext zulässig sein und kann die Bedeutung von Operationen näher bestimmen.
  Beispiel: die Operanden des „$*$“-Operators müssen Zahlwerte sein.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 4: Dynamische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Wirkung beschreiben und erst bei der Ausführung bestimmt oder geprüft werden können (= dynamisch).

Elemente der dynamischen Semantik:

- Regeln zur Analyse von Voraussetzungen, die für eine korrekte Ausführung eines Sprachkonstruktes erfüllt sein müssen.
  Beispiel: ein numerischer Index einer Array-Indizierung, wie in `$var[$i]`, darf nicht kleiner als 0 sein.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 4: Dynamische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Wirkung beschreiben und erst bei der Ausführung bestimmt oder geprüft werden können (= dynamisch).

Elemente der dynamischen Semantik:

- Regeln zur Analyse von Voraussetzungen, die für eine korrekte Ausführung eines Sprachkonstruktes erfüllt sein müssen.
  Beispiel: ein numerischer Index einer Array-Indizierung, wie in `$var[$i]`, darf nicht kleiner als 0 sein.

- Regeln zur Umsetzung bestimmter Sprachkonstrukte.
  Beispiel: Auswertung einer Zuweisung der Form

  *Variable = Ausdruck*

  Die Speicherstelle der Variablen auf der linken Seite wird bestimmt. Der Ausdruck auf der rechten Seite wird ausgewertet. Das Ergebnis ersetzt dann den Wert an der Stelle der Variablen. [SELFHTML]

Bemerkungen:

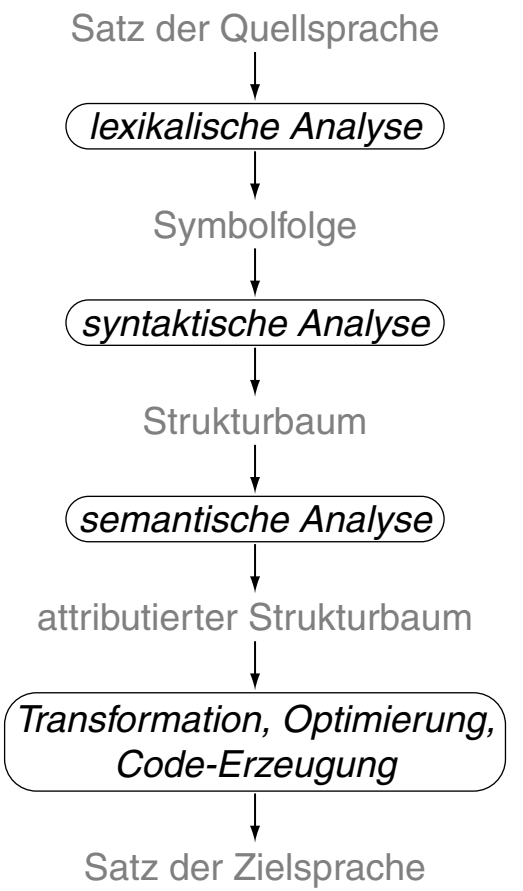
- Auf jeder der vier Ebenen gibt es also Regeln, die korrekte Sätze erfüllen müssen.
- In der Sprache PHP gehören die Typregeln zur dynamischen Semantik, da sie erst bei der Ausführung des Programms anwendbar sind.
- In der Sprache JavaScript gehören die Bindungsregeln zur statischen Semantik und die Typregeln zur dynamischen Semantik.

# Exkurs: Programmiersprachen [Kastens 2005]

## Übersetzung von Sprachen

Ein Übersetzer transformiert jeden korrekten Satz (Programm) der Quellsprache in einen gleichbedeutenden Satz (Programm) der Zielsprache.

- Die meisten Programmiersprachen zur Software-Entwicklung werden übersetzt. Beispiele: C, C++, Java, Ada, Modula.
- Zielsprache ist dabei meist eine Maschinensprache eines realen Prozessors oder einer abstrakten Maschine.
- Übersetzte Sprachen haben eine stark ausgeprägte statische Semantik.
- Der Übersetzer prüft die Regeln der statischen Semantik; viele Arten von Fehlern lassen sich vor der Ausführung finden.



# Exkurs: Programmiersprachen [Kastens 2005]

## Interpretation von Sprachen

Ein Interpretierer liest einen Satz (Programm) einer Sprache und führt ihn aus.

Für Sprachen, die strikt interpretiert werden, gilt:

- sie haben eine einfache Struktur und keine statische Semantik
- Bindungs- und Typregeln werden erst bei der Ausführung geprüft
- nicht ausgeführte Programmteile bleiben ungeprüft

Beispiele: Prolog, interpretiertes Lisp

Moderne Interpretierer erzeugen vor der Ausführung eine interne Repräsentation des Satzes; dann können auch Struktur und Regeln der statischen Semantik vor der Ausführung geprüft werden.

Beispiele: die Skriptsprachen JavaScript, PHP, Perl

Bemerkungen:

- Es gibt auch Übersetzer für Sprachen, die keine einschlägigen Programmiersprachen sind: Sprachen zur Textformatierung (LaTeX$\rightarrow$ PDF), Spezifikationssprachen (UML $\rightarrow$ Java).
- Interpretierer können auf jedem Rechner verfügbar gemacht werden und lassen sich in andere Software (Web-Browser) integrieren.
- Ein Interpretierer schafft die Möglichkeit einer weiteren Kapselung der Programmausführung gegenüber dem Betriebssystem.
- Interpretation kann 10-100 mal zeitaufwändiger sein, als die Ausführung von übersetztem Maschinencode.

# JavaScript

## Einführung [Einordnung]

Charakteristika:

- interpretiert, dynamisch typisiert
- einfache objektorientierte Konzepte
- Notation wie C/C++ und Java, wenig Bezug zu Java
- eng verknüpft mit HTML via DOM-API
- Interpretierer im Web-Browser integriert

# JavaScript

## Einführung [Einordnung]

Charakteristika:

- interpretiert, dynamisch typisiert
- einfache objektorientierte Konzepte
- Notation wie C/C++ und Java, wenig Bezug zu Java
- eng verknüpft mit HTML via DOM-API
- Interpretierer im Web-Browser integriert

Anwendung:

- Programme, die im Web-Browser ausgeführt werden
- dynamischen Web-Seiten, Animationseffekte
- Reaktion auf Ereignisse bei der Interaktion mit Web-Seiten
- Programme, die Server-seitig ausgeführt werden

# JavaScript [Kastens 2005]

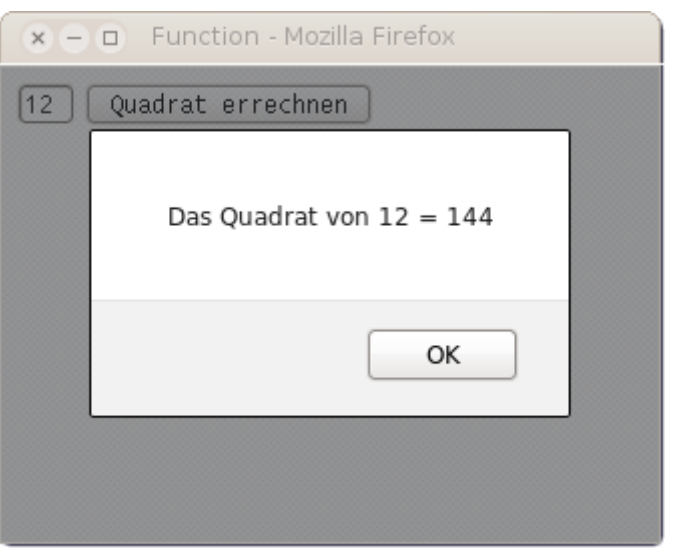
## Einführung (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Function</title>
    <script type="text/javascript">
      function Quadrat() {
        var Zahl = document.QuadratForm.Eingabe.value;
        var Ergebnis = Zahl * Zahl;
        alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
      }
    </script>
  </head>
  <body>
    <form name="QuadratForm" id="QF" action="">
      <input type="text" name="Eingabe" size="3">
      <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
    </form>
  </body>
</html>
```

# JavaScript [Kastens 2005]

## Einführung (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Function</title>
    <script type="text/javascript">
      function Quadrat() {
        var Zahl = document.QuadratForm.Eingabe.value;
        var Ergebnis = Zahl * Zahl;
        alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
      }
    </script>
  </head>
  <body>
    <form name="QuadratForm" id="QF" action="'
      <input type="text" name="Eingabe" size="
      <input type="button" value="Quadrat erre
    </form>
  </body>
</html>
```




[JavaScript-Ausführung]

Bemerkungen:

- JavaScript kompakt:
  1. Historie
  2. Einbindung in HTML-Dokumente
  3. Grundlagen der Syntax
  4. Variablen
  5. Operatoren
  6. Datentypen
  7. Kontrollstrukturen
  8. Funktionsbibliothek
  9. Ereignisbehandlung

# JavaScript

## Historie

1993 NCSA Mosaic-Browser. Bilder im Fließtext, Farben für Links und Text.

1994 Netscape 1. Entwickelt von einer Splittergruppe des Mosaic-Teams.

1996 Netscape 2. Frames, JavaScript von Brendan Eich. JavaScript heißt zunächst LiveWire, dann LiveScript. Art des Dokumentzugriffs entspricht heutigem DOM Level 0.

# JavaScript

## Historie

1993 NCSA Mosaic-Browser. Bilder im Fließtext, Farben für Links und Text.

1994 Netscape 1. Entwickelt von einer Splittergruppe des Mosaic-Teams.

1996 Netscape 2. Frames, JavaScript von Brendan Eich. JavaScript heißt zunächst LiveWire, dann LiveScript. Art des Dokumentzugriffs entspricht heutigem DOM Level 0.

1996 Standardisierung der JavaScript Kernsprache durch die European Computer Manufacturers Association als ECMAScript in der Spezifikation ECMA-262. [MDN]

1997 Netscape 4. „DHTML“, basierend auf W3C CSS 1 und JavaScript 1.2.

1997 Internet Explorer 4. Revolutionäres Konzept für dynamische Webseiten: W3C orientiert sich mit DOM- Entwicklung daran. Microsoft entwickelt JScript als Konkurrenz zu JavaScript.

1998 Netscape-Code wird Open Source, Mozilla-Projekt wird gestartet.

# JavaScript

## Historie

1993 NCSA Mosaic-Browser. Bilder im Fließtext, Farben für Links und Text.

1994 Netscape 1. Entwickelt von einer Splittergruppe des Mosaic-Teams.

1996 Netscape 2. Frames, JavaScript von Brendan Eich. JavaScript heißt zunächst LiveWire, dann LiveScript. Art des Dokumentzugriffs entspricht heutigem DOM Level 0.

1996 Standardisierung der JavaScript Kernsprache durch die European Computer Manufacturers Association als ECMAScript in der Spezifikation ECMA-262. [MDN]

1997 Netscape 4. „DHTML“, basierend auf W3C CSS 1 und JavaScript 1.2.

1997 Internet Explorer 4. Revolutionäres Konzept für dynamische Webseiten: W3C orientiert sich mit DOM- Entwicklung daran. Microsoft entwickelt JScript als Konkurrenz zu JavaScript.

1998 Netscape-Code wird Open Source, Mozilla-Projekt wird gestartet.

2002 Mozilla Phoenix 0.1 (später Firefox). Open Source Rendering-Engine Gecko.

2008 Google Chrome 1. Freie JavaScript-Engine V8 und JavaScript 1.7 $\sim$ ECMAScript 3.

2010 „Letzte“ JavaScript-Version ist 1.8.5. Bezeichnung nun als ECMA-262 Editions. [MDN]

2012 ECMAScript 5. Übersicht über die Browser-Unterstückung: Alle modernen Browser unterstützen ECMAScript 5. [kangax]

2018 Statistiken zur Verbreitung: [tiobe.com] [redmonk.com]

Bemerkungen:

- Die Entwicklung von JavaScript ist eng verknüpft mit dem „Browser-Krieg“ zwischen Microsoft und Netscape. Mehr zur JavaScript-Historie: [Tarquin] [SELFHTML]
- Ziel der W3C-DOM-Initiative war und ist es, die Browser-Entwicklung zu vereinheitlichen. Mittlerweile ermöglichen die Browser-APIs der verschiedenen Hersteller den Zugriff auf das HTML-Dokument gemäß der DOM Level 3 Spezifikation.
- Wiederholung: W3C DOM ist nicht nur für HTML-bezogene Skriptsprachen konzipiert, sondern bezieht sich auf alle Arten von Dokumenten, die in einer SGML-basierten Sprache geschrieben sind. [MDN] [SELFHTML]

# JavaScript

## Einbindung in HTML-Dokumente [SELFHTML]

1. Als Script-Bereich innerhalb eines HTML-Dokuments [JavaScript-Ausführung: 1, 2]:

```
<script type="text/javascript">
  ...
</script>
```

# JavaScript

## Einbindung in HTML-Dokumente [SELFHTML]

1. Als Script-Bereich innerhalb eines HTML-Dokuments [JavaScript-Ausführung: 1, 2]:

```
<script type="text/javascript">
  ...
</script>
```

2. Innerhalb von HTML-Tags [JavaScript-Ausführung]:

   Verwendung im Zusammenhang mit Ereignissen *(Events)*, die ein Bediener auslösen kann.

   (a) Das Ereignis ist als Attribut codiert; der Attributwert ist eine Anweisungsfolge, die beim Eintritt des Ereignisses ausgeführt wird:

   ```
   <input type="button" value="..." onClick="Quadrat()">
   ```
   [Beispiel]

   (b) In einem Anker-Element kann – anstatt einer URL – mit `javascript:` eine Anweisungsfolge angeben werden, die beim Klicken ausgeführt wird:

   ```
   <a href="javascript:Quadrat()">...</a>
   ```

# JavaScript

## Einbindung in HTML-Dokumente [SELFHTML]

1. Als Script-Bereich innerhalb eines HTML-Dokuments [JavaScript-Ausführung: 1, 2]:

```
<script type="text/javascript">
  ...
</script>
```

2. Innerhalb von HTML-Tags [JavaScript-Ausführung]:

   Verwendung im Zusammenhang mit Ereignissen *(Events)*, die ein Bediener auslösen kann.

   (a) Das Ereignis ist als Attribut codiert; der Attributwert ist eine Anweisungsfolge, die beim Eintritt des Ereignisses ausgeführt wird:

   ```
   <input type="button" value="..." onClick="Quadrat()">
   ```
   [Beispiel]

   (b) In einem Anker-Element kann – anstatt einer URL – mit `javascript:` eine Anweisungsfolge angeben werden, die beim Klicken ausgeführt wird:

   ```
   <a href="javascript:Quadrat()">...</a>
   ```

3. In einer separaten Datei [Source, JavaScript-Ausführung]:

```
<script src="usage.js" type="text/javascript"></script>
```

Bemerkungen:

- Das `<script>`-Element kann mehrfach in einem HTML-Dokument verwendet werden.
- Der JavaScript-Code eines Dokuments wird beim Einlesen des Dokuments vom Browser sofort ausgeführt.
- Die Auswertung einer Funktions*definition* erzeugt keine Ausgabe und liefert auch keinen Return-Wert.
- Es gibt keine Vorschrift dafür, an welcher Stelle in einem HTML-Dokument ein JavaScript-Bereich definiert werden darf. Aus Sicht der Ladezeit kann es sinnvoll sein, diesen Bereich am Ende eines HTML-Dokuments zu platzieren.
- Eine separate Datei mit JavaScript-Code sollte die Dateinamenerweiterung `.js` besitzen; insbesondere darf diese Datei nur JavaScript-Code enthalten. Das Encoding der Datei kann im einbindenden `<script>`-Element per `charset`-Attribut spezifiziert werden.

# JavaScript

## Grundlagen der Syntax [PHP]

Die Notation ähnelt in vieler Hinsicht der von C++ und Java.

### Bezeichner

- einheitliche Schreibweise für alle Arten von Bezeichnern:

  ```
  identifier = { letter | $ | _ }{ letter | $ | _ | digit }*
  ```

- Groß-/Kleinschreibung wird unterschieden *(case sensitive)*

# JavaScript

## Grundlagen der Syntax [PHP]

Die Notation ähnelt in vieler Hinsicht der von C++ und Java.

### Bezeichner

- einheitliche Schreibweise für alle Arten von Bezeichnern:

  `identifier = { letter | $ | _ }{ letter | $ | _ | digit }*`

- Groß-/Kleinschreibung wird unterschieden *(case sensitive)*

### Anweisungen

- Ein Semikolon am Zeilenende ist möglich, kann aber entfallen.
- Zwischen Anweisungen in derselben Zeile muss ein Semikolon stehen.
- `//` kommentiert bis Zeilenende aus.
- Balancierte Kommentarklammerung: `/*` *Kommentar* `*/`

# JavaScript

## Variablen und Konstanten [PHP] [MDN] [Wikipedia]

- Zur Deklaration von *Variablen* dienen die Schlüsselworte `var` und `let`.
- Zur Deklaration von *Konstanten* dient das Schlüsselwort `const`.
- Variablen und Konstanten können Werte beliebigen Typs annehmen.
- Unterscheidung von lokalen und globalen Variablen und Konstanten.

- Eine Variable ist lokal für eine Funktion, wenn sie innerhalb des Bindungsbereiches der Funktion mit `var` deklariert wird.
- Eine Variable (Konstante) ist lokal für einen Block, wenn sie innerhalb des Bindungsbereiches des Blocks mit `let` (`const`) deklariert wird.
- Globale Variablen und Konstanten gelten im ganzen Programm – es sei denn, sie werden von einer lokalen Variable oder Konstante überdeckt; lokale Variablen und Konstanten gelten nur in ihrem Bindungsbereich.
- Hoisting: unabhängig von ihrer Position gelten mit `var` eingeführte Variablen und Funktionen eines Codeabschnitts als sofort deklariert.

# JavaScript

## Variablen und Konstanten [PHP] [MDN] [Wikipedia]

- Zur Deklaration von *Variablen* dienen die Schlüsselworte `var` und `let`.
- Zur Deklaration von *Konstanten* dient das Schlüsselwort `const`.
- Variablen und Konstanten können Werte beliebigen Typs annehmen.
- Unterscheidung von lokalen und globalen Variablen und Konstanten.

- Eine Variable ist lokal für eine Funktion, wenn sie innerhalb des Bindungsbereiches der Funktion mit `var` deklariert wird.
- Eine Variable (Konstante) ist lokal für einen Block, wenn sie innerhalb des Bindungsbereiches des Blocks mit `let` (`const`) deklariert wird.
- Globale Variablen und Konstanten gelten im ganzen Programm – es sei denn, sie werden von einer lokalen Variable oder Konstante überdeckt; lokale Variablen und Konstanten gelten nur in ihrem Bindungsbereich.
- Hoisting: unabhängig von ihrer Position gelten mit `var` eingeführte Variablen und Funktionen eines Codeabschnitts als sofort deklariert.

# JavaScript

## Variablen: Illustration von Geltungsbereichen

```
var line, sum;
var col = 2;
var minimum = col,
    maximum = 999;

function compute (n)
{
  var sum = n;
  sum = sum * col;
  return sum;
}
```

# JavaScript

## Variablen: Illustration von Geltungsbereichen

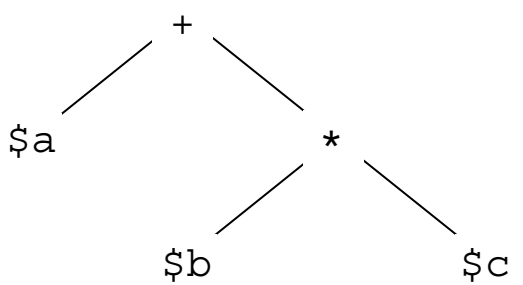
```
var line, sum;
var col = 2;
var minimum = col,
    maximum = 999;

function compute (n) // n ist lokale Variable.
{
  var sum = n;       // sum ist lokale Variable.
  sum = sum * col;   // col ist globale Variable.
  return sum;
}
```

# JavaScript [Kastens 2005]

## Operatoren: Präzedenz, Assoziativität

Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

```
$a + $b * $c
```

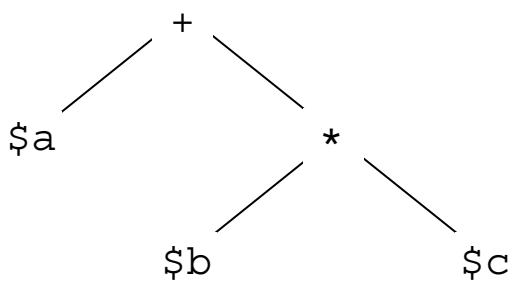


# JavaScript [Kastens 2005]

## Operatoren: Präzedenz, Assoziativität

Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

```
$a + $b * $c
```



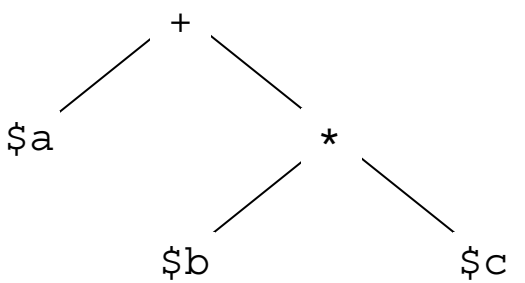

Ein Operator ist linksassoziativ (rechtsassoziativ), wenn beim Zusammentreffen von Operatoren gleicher Präzedenz der linke (rechte) Operator seine Operanden stärker bindet als der rechte (linke).

# JavaScript [Kastens 2005]

## Operatoren: Präzedenz, Assoziativität

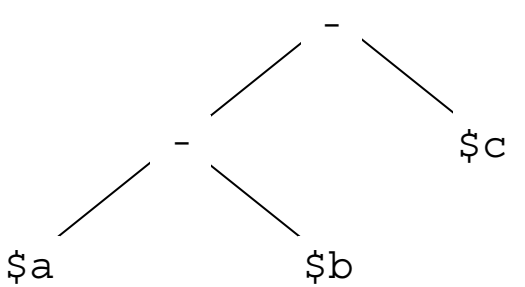
Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

```
$a + $b * $c
```




Ein Operator ist linksassoziativ (rechtsassoziativ), wenn beim Zusammentreffen von Operatoren gleicher Präzedenz der linke (rechte) Operator seine Operanden stärker bindet als der rechte (linke). Beispiel:

```
$a - $b - $c
```



# JavaScript [Kastens 2005]

## Operatoren: Übersicht [SELFHTML]

| Präzedenz | Stelligkeit | Assoziativität | Operatoren | Erklärung |
|---|---|---|---|---|
| 1 | 2 | rechts | `= += -=` | Zuweisungsoperatoren |
| 2 | 3 | links | `?:` | bedingter Ausdruck |
| 3 | 2 | links | `\|\|` | logische Disjunktion |
| 4 | 2 | links | `&&` | logische Konjunktion |
| 5 | 2 | links | `\|` | Bitoperator |
| 6 | 2 | links | `^` | Bitoperator |
| 7 | 2 | links | `&` | Bitoperator |
| 8 | 2 | links | `== != ===` | Gleichheit, Identität |
| 9 | 2 | links | `< <= > >=` | Ordnungsvergleich |
| 10 | 2 | links | `<< >> >>>` | shift-Operatoren |
| 11 | 2 | links | `+ -` | Konkatenation, Add., Subtr. |
| 12 | 2 | links | `* / %` | Arithmetik |
| 13 | 1 | | `! - ~` | Negation (logisch, arithm.) |
| | 1 | | `++ --` | Inkrement, Dekrement |
| | 1 | | `typeof void` | Typabfragen, cast to undefined |
| 14 | 1 | | `() [] .` | Aufruf, Index, Objektzugriff |

# JavaScript

## Datentypen: Primitive [PHP]

`number`

- Keine Unterscheidung zwischen Ganzzahlen und Gleitpunktzahlen.
- Der Wert `NaN` *(not a number)* steht für ein undefiniertes Ergebnis.
- Der Wert `Infinity` steht für einen Wert, der größer als die größte repräsentierbare Zahl ist.

`string`

- Zeichenkettenliterale mit einfachen oder doppelten Anführungszeichen.
- Konkatenation wie in Java: `var s = "Hello" + "world!"`
- Zeichenkettenfunktionen werden in objektorientierter Notation verwendet. Beispiele: `s.length`, `s.indexOf(substr)`, `s.charAt(i)`.

`boolean`

- Literale: `true` und `false` (insbesondere nicht: `True` bzw. `False`)
- Operatoren: Konjunktion `&&`, Disjunktion `||`, Negation `!`

# JavaScript

## Datentypen: Primitive (Fortsetzung)

`undefined`

- Der Wert `undefined` steht dafür, dass eine Variable keinen Wert hat.
- `undefined` wird zurückgegeben, falls (a) eine Variable benutzt wird, die zwar deklariert aber der nie ein Wert zugewiesen wurde oder (b) auf eine Objektkomponente zugegriffen wird, die nicht existiert.

`null`

- Der Wert `null` steht dafür, dass eine Variable keinen gültigen Wert hat.
- Obwohl `null` und `undefined` verschiedene Werte sind, werden sie vom Operator `==` als gleich interpretiert.

# JavaScript

## Datentypen: Funktionen [MDN]

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

Definition von Funktionen:

(a) Durch eine „klassische“ Funktionsdeklaration:

(b) Durch einen Funktionsausdruck bzw. Funktionsliteral [kangax]:

(c) Mit dem Konstruktor `Function()`:

# JavaScript

## Datentypen: Funktionen [MDN]

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

Definition von Funktionen:

(a) Durch eine „klassische“ Funktionsdeklaration:

```
function f(x, y) {
  return x*y;
}
```

(b) Durch einen Funktionsausdruck bzw. Funktionsliteral [kangax]:

(c) Mit dem Konstruktor `Function()`:

# JavaScript

## Datentypen: Funktionen [MDN]

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

Definition von Funktionen:

(a) Durch eine „klassische“ Funktionsdeklaration:

```
function f(x, y) {
  return x*y;
}
```

(b) Durch einen Funktionsausdruck bzw. Funktionsliteral [kangax]:

```
var p = function(x, y) { return x*y; };     // (anonym)
```

oder

```
var q = function g(x, y) { return x*y; };   // (benannt)
```

(c) Mit dem Konstruktor `Function()`:

# JavaScript

## Datentypen: Funktionen [MDN]

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

Definition von Funktionen:

(a) Durch eine „klassische“ Funktionsdeklaration:

```
function f(x, y) {
  return x*y;
}
```

(b) Durch einen Funktionsausdruck bzw. Funktionsliteral [kangax]:

```
var p = function(x, y) { return x*y; };     // (anonym)
```

oder

```
var q = function g(x, y) { return x*y; };   // (benannt)
```

(c) Mit dem Konstruktor `Function()`:

```
var r = new Function("x", "y", "return x*y;");
```

# JavaScript

## Datentypen: Objekte [vordefinierte Objekte]

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

Definition von Objekten [w3schools]:

(a) Durch Verwendung des Block-Statements `{}` als Objektliteral:

(b) Mit dem Konstruktor `Object()`:

# JavaScript

## Datentypen: Objekte [vordefinierte Objekte]

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

Definition von Objekten [w3schools]:

(a) Durch Verwendung des Block-Statements `{}` als Objektliteral:

```
var car = { year:2020, brand:"Tesla" };
```

(b) Mit dem Konstruktor `Object()`:

# JavaScript

## Datentypen: Objekte [vordefinierte Objekte]

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

Definition von Objekten [w3schools]:

(a) Durch Verwendung des Block-Statements `{}` als Objektliteral:

```
var car = { year:2020, brand:"Tesla" };
```

(b) Mit dem Konstruktor `Object()`:

```
var car = new Object();
```

Hinzufügen von Komponenten:

```
car.year = 2020;
car.brand = "Tesla";
```

# JavaScript

## Datentypen: Objekte [vordefinierte Objekte]

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

Definition von Objekten [w3schools]:

(a) Durch Verwendung des Block-Statements `{}` als Objektliteral:

```
var car = { year:2020, brand:"Tesla" };
```

(b) Mit dem Konstruktor `Object()`:

```
var car = new Object();
```

Hinzufügen von Komponenten:

```
car.year = 2020;
car.brand = "Tesla";
```

Zugriff auf Objektkomponenten mit *Objektausdruck.Komponente*:

`car.year` ⇝ `2020`

Bemerkungen:

- JavaScript ist keine objektorientierte Progammiersprache im Sinne von Java oder C++. Es gibt zwar den Datentyp `Object`, aber kein klassenbasiertes Typ- und Vererbungskonzept. Stattdessen kann in JavaScript jedes Objekt als „Prototyp“ (zur Konstruktion neuer Objekte) verstanden werden: Objekte lassen sich kopieren, Komponenten lassen sich hinzufügen und Vererbungshierchachien aufbauen. Stichwort: *prototypbasierte Vererbung* [MDN] [O'Reilly]
- Objektkomponenten können Funktionen sein und heißen dann Methoden. Komponenten, die keine Methoden sind, heißen Eigenschaften oder Attribute.
  - Aufruf von Methoden: *Objektausdruck.Komponente*`()`
  - Zugriff auf Eigenschaft oder Funktionsdefinition: *Objektausdruck.Komponente*
- *Objektausdruck* ist ein JavaScript-Ausdruck, der zu einer Referenz auf ein JavaScript-Objekt evaluiert.

# JavaScript

## Datentypen: Objekte (Fortsetzung)

*Jede* Funktionsdefinition ist als Konstruktor verwendbar [MDN]; die gewünschten Objektkomponenten werden mit dem Schlüsselwort `this` deklariert.

```
function MyCar(a, b) {
  this.year = a;
  this.brand = b;
}

var car = new MyCar(2020, "Tesla");
```

# JavaScript

## Datentypen: Objekte (Fortsetzung)

*Jede* Funktionsdefinition ist als Konstruktor verwendbar [MDN]; die gewünschten Objektkomponenten werden mit dem Schlüsselwort `this` deklariert.

```
function MyCar(a, b) {
  this.year = a;
  this.brand = b;
}

var car = new MyCar(2020, "Tesla");
```

Es funktioniert also auch:

```
function f(x, y) {
  ...
  this.wert = x+y;
  ...
  return x*y;
}


f(3, 4) ⇝ 12
new f(3, 4) ⇝ Object { wert: 7 }
```

# JavaScript

## Datentypen: Objekte (Fortsetzung)

### Definition von Methoden als Funktion oder Funktionsausdruck [JavaScript-Ausführung]:

```
function MyCircle(r) {
  this.radius = r;
  this.area = getArea;
  this.circ = function() { return (Math.PI * this.radius * 2); };
}

function getArea() {
  return (Math.PI * this.radius * this.radius);
}
```

# JavaScript

## Datentypen: Objekte (Fortsetzung)

### Definition von Methoden als Funktion oder Funktionsausdruck [JavaScript-Ausführung]:

```
function MyCircle(r) {
  this.radius = r;
  this.area = getArea;
  this.circ = function() { return (Math.PI * this.radius * 2); };
}

function getArea() {
  return (Math.PI * this.radius * this.radius);
}
```

### Aufruf:

```
c = new MyCircle(3);
document.writeln("Radius = " + c.radius);
document.writeln("Area = " + c.area());
document.writeln("Circumference = " + c.circ());
```

# JavaScript

## Datentypen: Objekte (Fortsetzung)

### Definition von Methoden als Funktion oder Funktionsausdruck [JavaScript-Ausführung]:

```
function MyCircle(r) {
  this.radius = r;
  this.area = getArea;
  this.circ = function() { return (Math.PI * this.radius * 2); };
}

function getArea() {
  return (Math.PI * this.radius * this.radius);
}
```

### Aufruf:

```
c = new MyCircle(3);
document.writeln("Radius = " + c.radius);
document.writeln("Area = " + c.area());
document.writeln("Circumference = " + c.circ());
```

### Zugriff auf Objektkomponenten:

```
c.area ⇝ function getArea() {return (Math.PI * this.radius * this.radius);}
c.circ ⇝ function {return (Math.PI * this.radius * 2);}
```

Bemerkungen:

- Die Syntax einer Konstruktordefinition entspricht der Syntax einer Funktionsdefinition.
- Im Konstruktor wird auf die Komponenten mit dem Qualifier `this` zugegriffen; er bezeichnet die jeweilige mit `new` erzeugte Objektkopie.
- Per Konvention beginnt der Name einer Konstruktorfunktion mit einem Großbuchstaben.

# JavaScript

## Datentypen: Arrays [PHP] [JavaScript-Ausführung]

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert. Arrays sind Objekte vom Typ `Array`.

Erzeugung von Arrays mit dem Konstruktor `Array()`:

(a) Als Liste von Werten, indiziert von 0 an:

(b) Durch Erweiterung eines leeren Arrays:

(c) Als assoziatives Array:

# JavaScript

## Datentypen: Arrays [PHP] [JavaScript-Ausführung]

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert. Arrays sind Objekte vom Typ `Array`.

Erzeugung von Arrays mit dem Konstruktor `Array()`:

(a) Als Liste von Werten, indiziert von 0 an:

```
monatsName = new Array("", "Jan", ..., "Dez");
```

(b) Durch Erweiterung eines leeren Arrays:

```
monatsName = new Array();
monatsName[1]= "Jan"; monatsName[2]= "Feb"; ...
```

(c) Als assoziatives Array:

```
monatsNr = new Array();
monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; ...
```

# JavaScript

## Datentypen: Arrays [PHP] [JavaScript-Ausführung]

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert. Arrays sind Objekte vom Typ `Array`.

Erzeugung von Arrays mit dem Konstruktor `Array()`:

(a) Als Liste von Werten, indiziert von 0 an:

```
monatsName = new Array("", "Jan", ..., "Dez");
```

(b) Durch Erweiterung eines leeren Arrays:

```
monatsName = new Array();
monatsName[1]= "Jan"; monatsName[2]= "Feb"; ...
```

(c) Als assoziatives Array:

```
monatsNr = new Array();
monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; ...
```

Aufzählung aller Elemente mit Schlüssel:

```
for (mname in monatsNr) {
  document.writeln (mname + "->" + monatsNr[mname] + "<br/>");
}
```

# JavaScript

## Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge:
- Bedingte Anweisung:
- Return-Anweisung:
- `while`-Schleife:
- `for`-Schleife [JavaScript-Ausführung] :

# JavaScript

## Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge:

  ```
  { var k = 42; document.writeln (5*k); }
  ```

  Eine Anweisungsfolge definiert keinen *Scope*: eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in der umgebenden Funktion bzw. Programm.

- Bedingte Anweisung:

  ```
  if (a < b) {min = a;} else {min = b;}
  ```

  Bei einzelnen Anweisungen sind die {}-Klammern optional.

- Return-Anweisung:

  ```
  return n*42;     return "*";
  ```

- `while`-Schleife:

- `for`-Schleife [JavaScript-Ausführung] :

# JavaScript

## Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge:

  ```
  { var k = 42; document.writeln (5*k); }
  ```

  Eine Anweisungsfolge definiert keinen *Scope*: eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in der umgebenden Funktion bzw. Programm.

- Bedingte Anweisung:

  ```
  if (a < b) {min = a;} else {min = b;}
  ```

  Bei einzelnen Anweisungen sind die {}-Klammern optional.

- Return-Anweisung:

  ```
  return n*42;     return "*";
  ```

- `while`-Schleife:

  ```
  i = 0; while (i < n) {document.write ("*"); i++;}
  ```

- `for`-Schleife [JavaScript-Ausführung] :

  ```
  for (i = 0; i < n; i++) {document.write ("*");}
  ```

# JavaScript

## Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge:

  ```
  { var k = 42; document.writeln (5*k); }
  ```

  Eine Anweisungsfolge definiert keinen *Scope*: eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in der umgebenden Funktion bzw. Programm.

- Bedingte Anweisung:

  ```
  if (a < b) {min = a;} else {min = b;}
  ```

  Bei einzelnen Anweisungen sind die {}-Klammern optional.

- Return-Anweisung:

  ```
  return n*42;     return "*";
  ```

- `while`-Schleife:

  ```
  i = 0; while (i < n) {document.write ("*"); i++;}
  ```

- `for`-Schleife [JavaScript-Ausführung] :

  ```
  for (i = 0; i < n; i++) {document.write ("*");}
  ```

- Parameterübergabe standardmäßig mittels call-by-value.

# JavaScript

## Funktionsbibliothek [PHP]

Ein Großteil der Funktionsbibliothek ist in Form von Objekten implementiert:

1. Vordefinierte Objekte. Funktionalität unabhängig von Dokument und Browser.

   Beispiele: `Array, Date, Function, Math, Object, RegExp` [MDN]

   Ergänzung durch vordefinierte Funktionen.

   Beispiele: `eval, isFinite, isNaN, parseInt` [MDN]

# JavaScript

## Funktionsbibliothek [PHP]

Ein Großteil der Funktionsbibliothek ist in Form von Objekten implementiert:

1. Vordefinierte Objekte. Funktionalität unabhängig von Dokument und Browser.

   Beispiele: `Array, Date, Function, Math, Object, RegExp` [MDN]

   Ergänzung durch vordefinierte Funktionen.

   Beispiele: `eval, isFinite, isNaN, parseInt` [MDN]

2. DOM-Objekte. Repräsentation von Dokument und Browser. Beispiele:

| Interface | DOM | DOM HTML |
|---|---|---|
| `Node` | [W3C] [WHATWG] [MDN] | (Interface nicht erweitert) |
| `Element, HTMLElement` | [W3C] [WHATWG] [MDN] | [W3C] [WHATWG] [MDN] |
| `Document` | [W3C] [WHATWG] [MDN] | [W3C] [WHATWG] [MDN] |
| `Window` | (Interface nicht vorgesehen) | [W3C] [WHATWG] [MDN] |

Bemerkungen:

- Die vordefinierten Objekte und Funktionen bilden den Kern der JavaScript-Sprache. Sprachreferenzen: [MDN] [w3schools]
- Vordefinierte Objekte werden auch als *Global Objects*, *Objects in the Global Scope* oder *Built-in Objects* bezeichnet.
- DOM-Objekte implementieren die Interfaces der sprachunabhängigen DOM-API. Dabei wird zwischen einem Kern-DOM ein HTML-DOM unterschieden. [MDN] [SELFHTML]

  Die entsprechenden Interface-Referenzen:

  - [W3C DOM, DOM HTML]
  - [WHATWG DOM, DOM HTML] [MDN]

Bemerkungen: (Fortsetzung)

- Die Wurzel der DOM-Objekt-Hierarchie ist das `Window`-Objekt. Eines der Kindobjekte ist das `Document`-Objekt; es bildet die Wurzel des HTML- bzw. XML-Dokuments.
- `Window` und `Document` sind die Objekte (Prototypen) gemäß der Interface-Spezifikation der DOM-API. In einem konkreten Dokument geschieht der Zugriff auf die entsprechenden Instanzen durch die JavaScript-Variablen `window` bzw. `document`; diese werden bei der Erzeugung eines neuen Browsing-Kontextes angelegt und initialisiert. [W3C]
- Eines der Attribute von `Document` heißt `documentElement`; es verweist auf die Objektinstanz, die das Wurzel*element* (`<html>` oder `<xml>`) des HTML- bzw. XML-Dokuments repräsentiert. [W3C] [MDN]
- Illustration von Markup, DOM und gerenderter HTML-Seite im Live-DOM-Viewer. [hixie.ch]

# JavaScript

## Funktionsbibliothek: DOM-Objekte

Konzepte, um auf HTML-Elementobjekte und deren Eigenschaften zuzugreifen:

1. Qualifizierender Name gemäß der DOM-Hierarchie im Dokument.

   Beispiel: `document.QuadratForm.Eingabe`

2. Methoden und Attribute der DOM-API. Beispiele:

   *Dokumentausdruck*`.getElementsByName()`
   *Dokumentausdruck*`.getElementById()`
   *Dokumentausdruck*`.images`
   *Dokumentausdruck*`.forms`

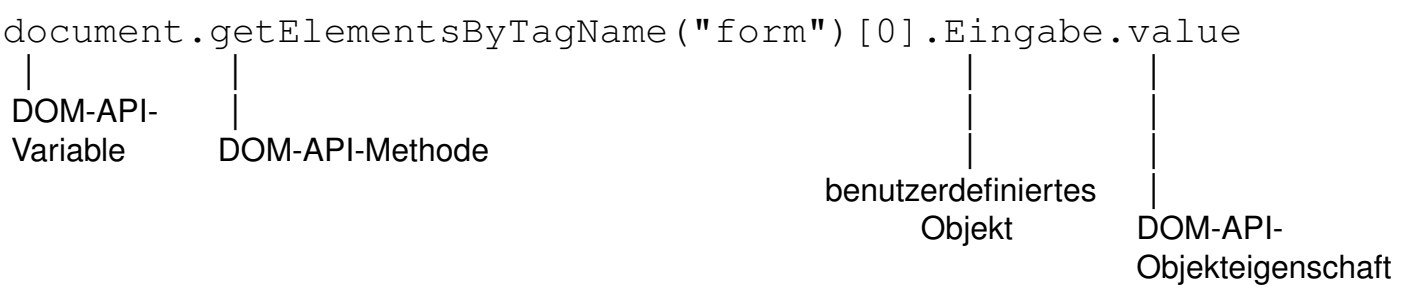
   { *Elementausdruck* | *Dokumentausdruck* }`.getElementsByTagName()`
   { *Elementausdruck* | *Dokumentausdruck* }`.querySelectorAll()`

3. Kombination von DOM-API und qualifizierendem Namen.

   Beispiel: `document.getElementsByTagName("form")[0].Eingabe`

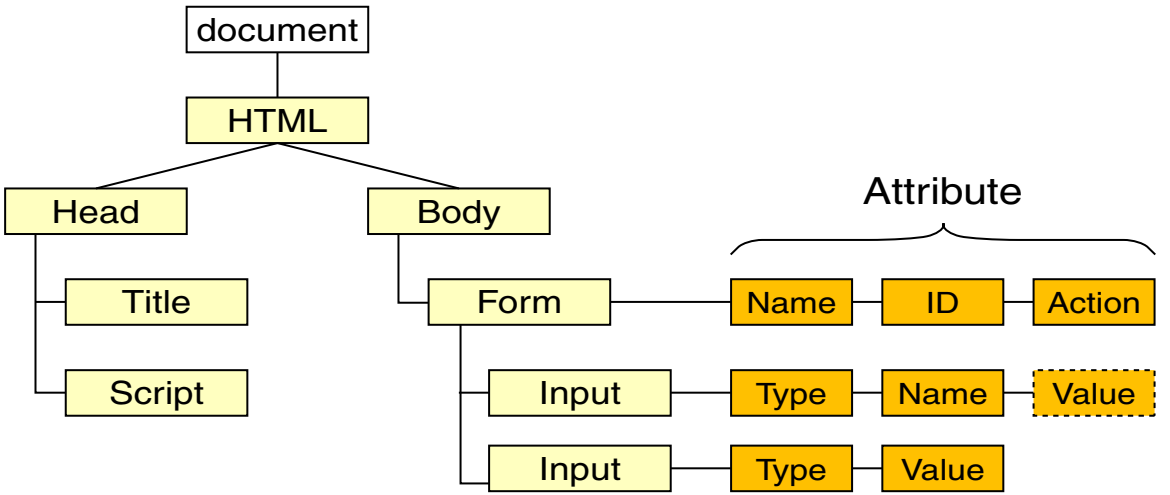
Bemerkungen:

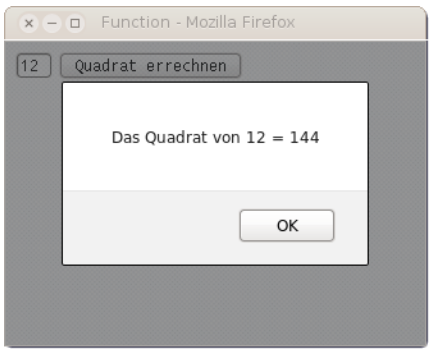
- Beim Parsen eines HTML-Dokuments durch den Browser (also mit dem Laden und Anzeigen eines Dokuments von einer URL) wird das DOM gemäß der Spezifikation der DOM-API instanziiert: neben den Instanzen des `Window`- und `Document`-Objekts wird für jedes HTML-Element eine entsprechende Objektinstanz erzeugt und verlinkt. Damit stehen alle für ein HTML-Element erlaubten Attribute als Objekteigenschaften im DOM zur Verfügung. Diese Datenstruktur bildet die Dokumentrepräsentation im Browser.
- *Elementausdruck* und *Dokumentausdruck* sind Spezialisierungen von *Objektausdruck* gemäß der Semantik der DOM-Hierarchie und evaluieren zu einer Referenz auf ein entsprechendes JavaScript-Objekt.
- *Objektausdruck* notiert einen Pfad in einer Objekthierarchie und kann Objekte, Methoden und Variablen kombinieren (gemäß der DOM-API definierte als auch benutzerdefinierte). Beispiel:



# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [JavaScript-Einführungsbeispiel]

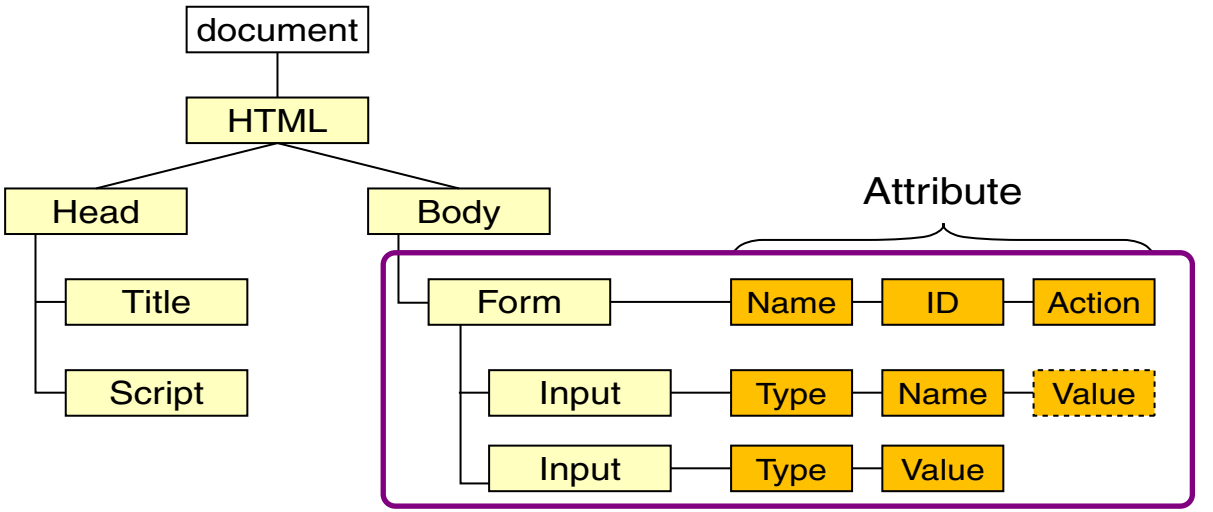



```
<!DOCTYPE html>
<html>
  <head>
    ...
    <script type="text/javascript">
      function Quadrat() {...}
    </script>
  </head>

  <body>
    <form name="QuadratForm" id="QF" action="">
      <input type="text" name="Eingabe" size="3">
      <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
    </form>
  </body>
</html>
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [JavaScript-Einführungsbeispiel]




Zugriffsmöglichkeiten auf das erste Formelement:

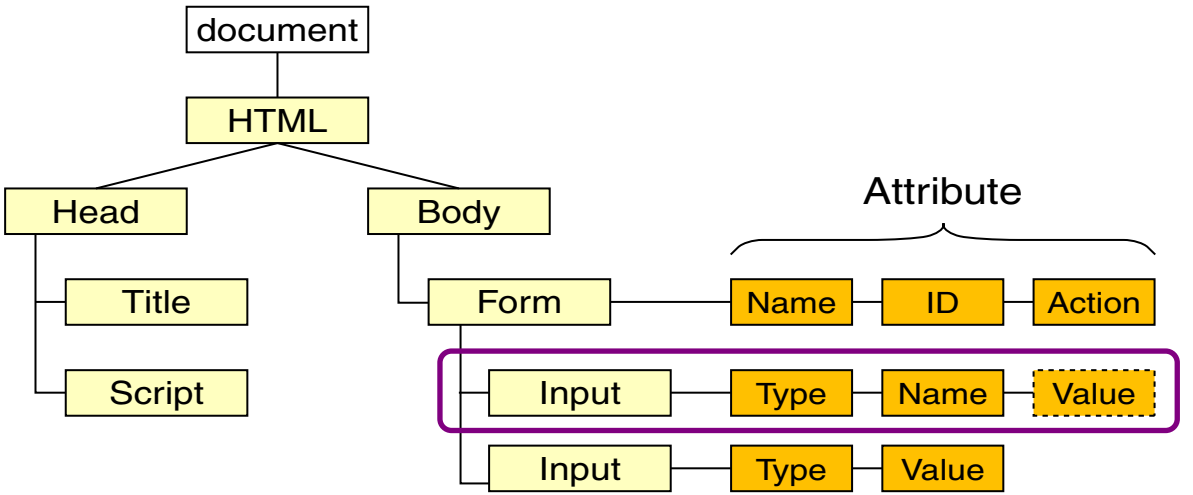
| | |
|---|---|
| `document.QuadratForm` | qualifizierender Name |
| `document.getElementsByTagName("form")[0]` | DOM-API |
| `document.getElementById("QF")` | DOM-API |
| `document.querySelector("body #QF")` | DOM-API |

Kontrollausgaben:

```
document.writeln(document.QuadratForm) ⇝ [object HTMLFormElement]
document.writeln(document.getElementsByTagName("form")) ⇝ [object HTMLCollection]
document.writeln(document.getElementsByTagName("form")[0]) ⇝ [object HTMLFormElement]
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [JavaScript-Einführungsbeispiel]




Zugriffsmöglichkeiten auf das erste Eingabeelement:

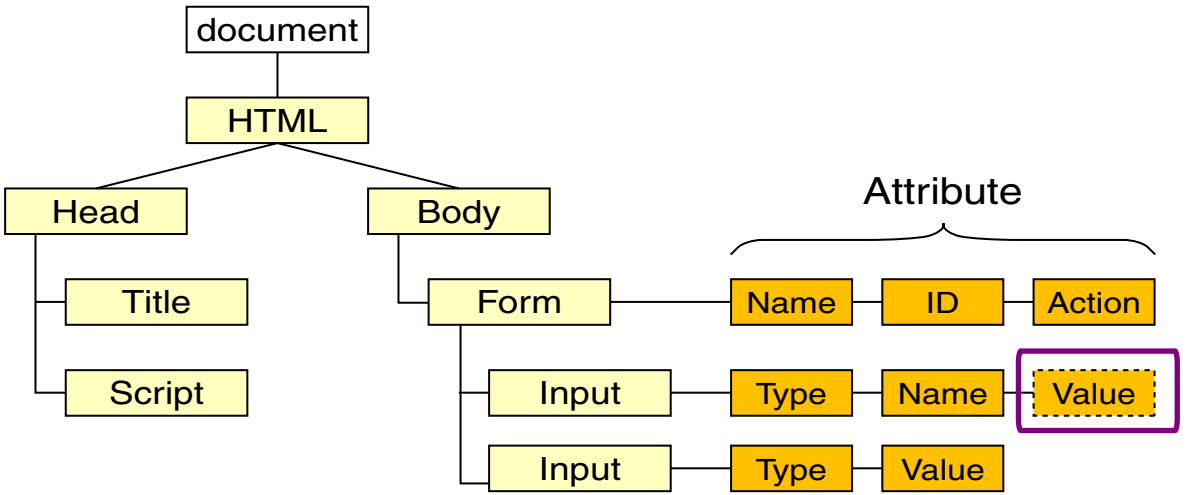
```
document.QuadratForm.Eingabe                          qualifizierender Name
document.getElementsByName("Eingabe")[0]              DOM-API
document.getElementsByTagName("form")[0].Eingabe      Kombination
document.querySelector("body #QF").Eingabe            Kombination
```

Kontrollausgaben:

```
document.writeln(document.QuadratForm.Eingabe) ⇝ [object HTMLInputElement]
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [JavaScript-Einführungsbeispiel]




Zugriffsmöglichkeiten auf das Eingabefeld im ersten Eingabeelement:

```
document.QuadratForm.Eingabe.value
document.getElementsByName("Eingabe")[0].value
document.getElementsByTagName("form")[0].Eingabe.value
document.querySelector("body #QF").Eingabe.value
```

Kontrollausgaben:

```
document.writeln(document.QuadratForm.Eingabe.value) ⇝ 11
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [JavaScript-Einführungsbeispiel]

```
<script type="text/javascript">
  function Quadrat() {
  var Zahl = document.QuadratForm.Eingabe.value;
  var Ergebnis = Zahl * Zahl;
  alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
  }
</script>

<form name="QuadratForm" id="QF" action="">
  <input type="text" name="Eingabe" size="3">
  <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
</form>
```

Genauso wie das Abfragen ist auch das Setzen von Werten möglich:

```
document.QuadratForm.Eingabe.value = 12;
```

# JavaScript

## Ereignisbehandlung

Ein Ereignis *(Event)* ist die Wahrnehmung einer Zustandsänderung. Die ereignisgetriebene Programmierung ordnet den Ereignissen Operationen zu.

# JavaScript [Kastens 2005]

## Ereignisbehandlung

Ein Ereignis *(Event)* ist die Wahrnehmung einer Zustandsänderung. Die ereignisgetriebene Programmierung ordnet den Ereignissen Operationen zu.

Varianten für die Behandlung des Ereignisses „Mausklick“ in `<form>`-Elementen:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event</title>
  </head>
  <body>
    <form name="testForm">
      <input type="button" value="ping" onclick='alert("ping!");'>
      <input type="button" value="pong" name="Knopf">
    </form>
    <script type="text/javascript">
      document.testForm.Knopf.onclick = function(){ alert("pong!") };
    </script>
  </body>
<html>
```

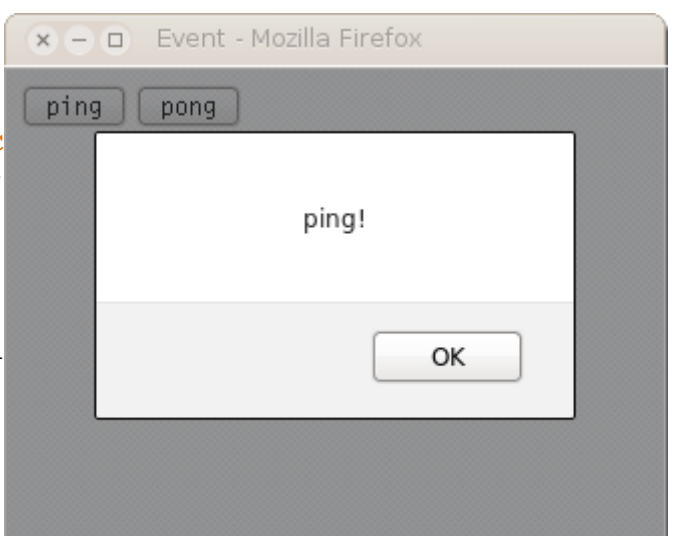
# JavaScript [Kastens 2005]

## Ereignisbehandlung

Ein Ereignis *(Event)* ist die Wahrnehmung einer Zustandsänderung. Die ereignisgetriebene Programmierung ordnet den Ereignissen Operationen zu.

Varianten für die Behandlung des Ereignisses „Mausklick“ in `<form>`-Elementen:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event</title>
  </head>
  <body>
    <form name="testForm">
      <input type="button" value="ping" onclic
      <input type="button" value="pong" name="
    </form>
    <script type="text/javascript">
      document.testForm.Knopf.onclick = functi
    </script>
  </body>
<html>
```




[JavaScript-Ausführung]

Bemerkungen:

- Beachte die unterschiedliche Art und Weise, wie die Funktionen als Wert des `onclick`-Attributes zugewiesen werden.
- Mittlerweile können viele Maus-Events ohne JavaScript mittels CSS realisiert werden. Beispiel: [webis]

# JavaScript

## Ereignisbehandlung: wichtige Ereignisse

| Event-Handler | HTML-Elemente | Semantik |
|---|---|---|
| `onclick` | Knopf, Checkbox, Anker | Element wird angeklickt |
| `onchange` | Textfeld, Textbereich, Auswahl | Wert wird geändert |
| `onkeydown`<br>`onkeyup`<br>`onkeypress` | Dokument, Bild, Anker,Textfeld | Taste gedrückt/losgelassen |
| `onload` | Body | Beim Laden eines Dokuments |
| `onmousedown`<br>`onmouseup` | Dokument, Knopf, Anker | Maustaste gedrückt/losgelassen |
| `onmouseout` | Bereiche, Anker | Mauszeiger verlässt einen Bereich |
| `onmouseover` | Anker | Mauszeiger über Anker |
| `onreset`,<br>`onsubmit` | Formular | Reset/Submit für ein Formular |
| `onselect` | Textfeld, Textbereich | Element wird ausgewählt |
| `onfocus`<br>`onblur` | Fenster, alle Formularelemente | Eingabefokus wird dem Element gegeben/entzogen |

# JavaScript

## Quellen zum Nachlernen und Nachschlagen im Web

- ECMA. *Standard ECMA-262: ECMAScript Language Specification.*
  www.ecma-international.org/publications/standards/Ecma-262
- Kastens. *Einführung in Web-bezogene Sprachen.*
  Vorlesung WS 2005/06, Universität Paderborn.
- MDN. *JavaScript.*
  developer.mozilla.org/en-US/docs/Web/JavaScript
- O'Reilly. *JavaScript. The Definitive Guide*
  docstore.mik.ua/orelly/webprog/jscript
- SELFHTML e.V. *JavaScript.*
  wiki.selfhtml.org/wiki/JavaScript
- W3 Schools. *JavaScript Tutorial.*
  www.w3schools.com/js
- Wenz. *JavaScript und AJAX.*
  openbook.rheinwerk-verlag.de/javascript_ajax