Vorlesung

Webbasierte Informationssysteme

(CS4130)

JavaScript (JS)

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

JavaScript - Anwendungen

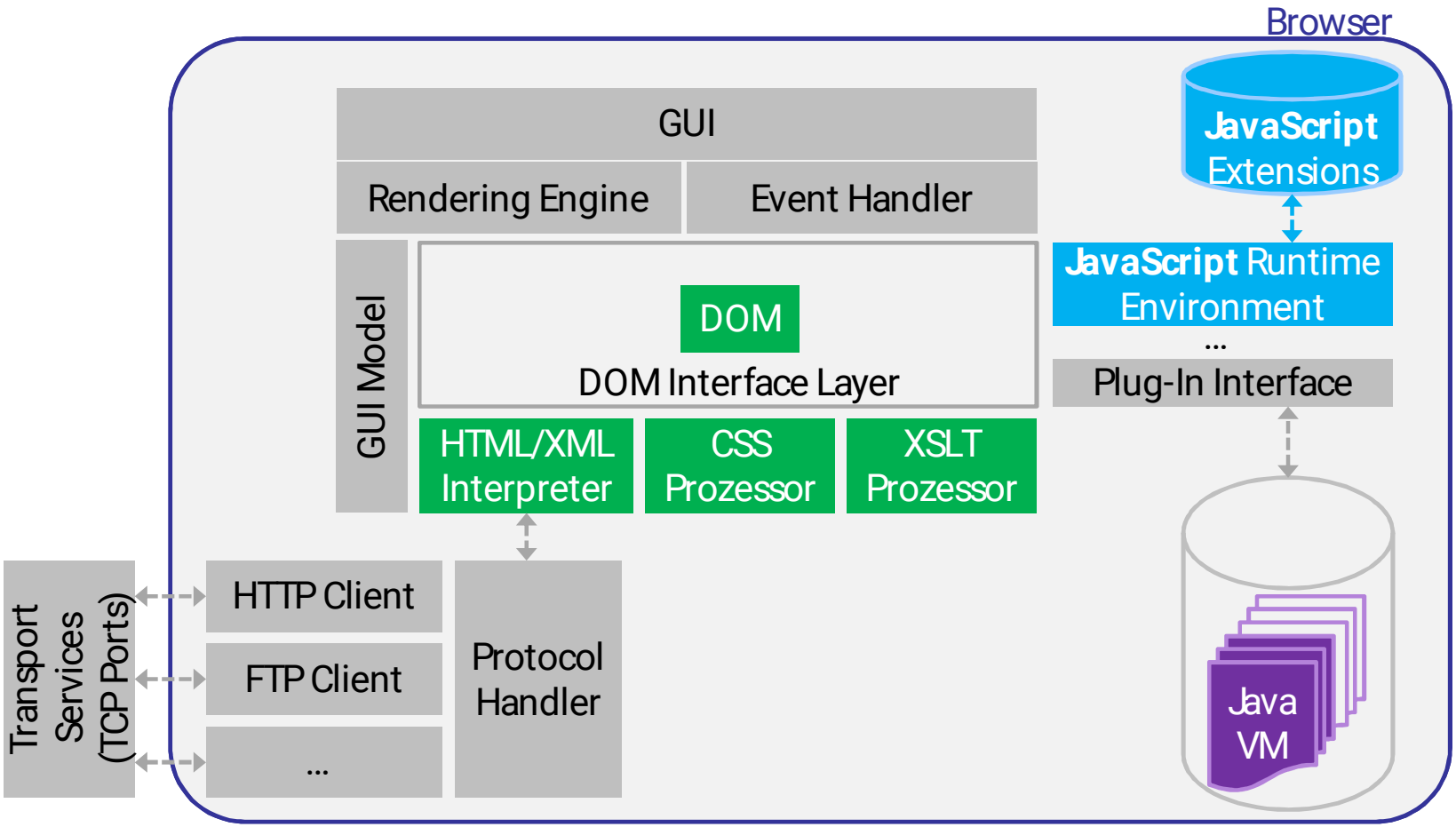
- Realisierung von client-seitig im Browser ablaufenden Web-Anwendungen
- Bedienoberflächen in dynamischen Web-Seiten, Animationseffekte
- Reaktion auf Ereignisse bei der Interaktion mit Web-Seiten
- Dynamische Erzeugung von Formularelementen, Eingabeüberprüfung

Verteilung der Verarbeitung

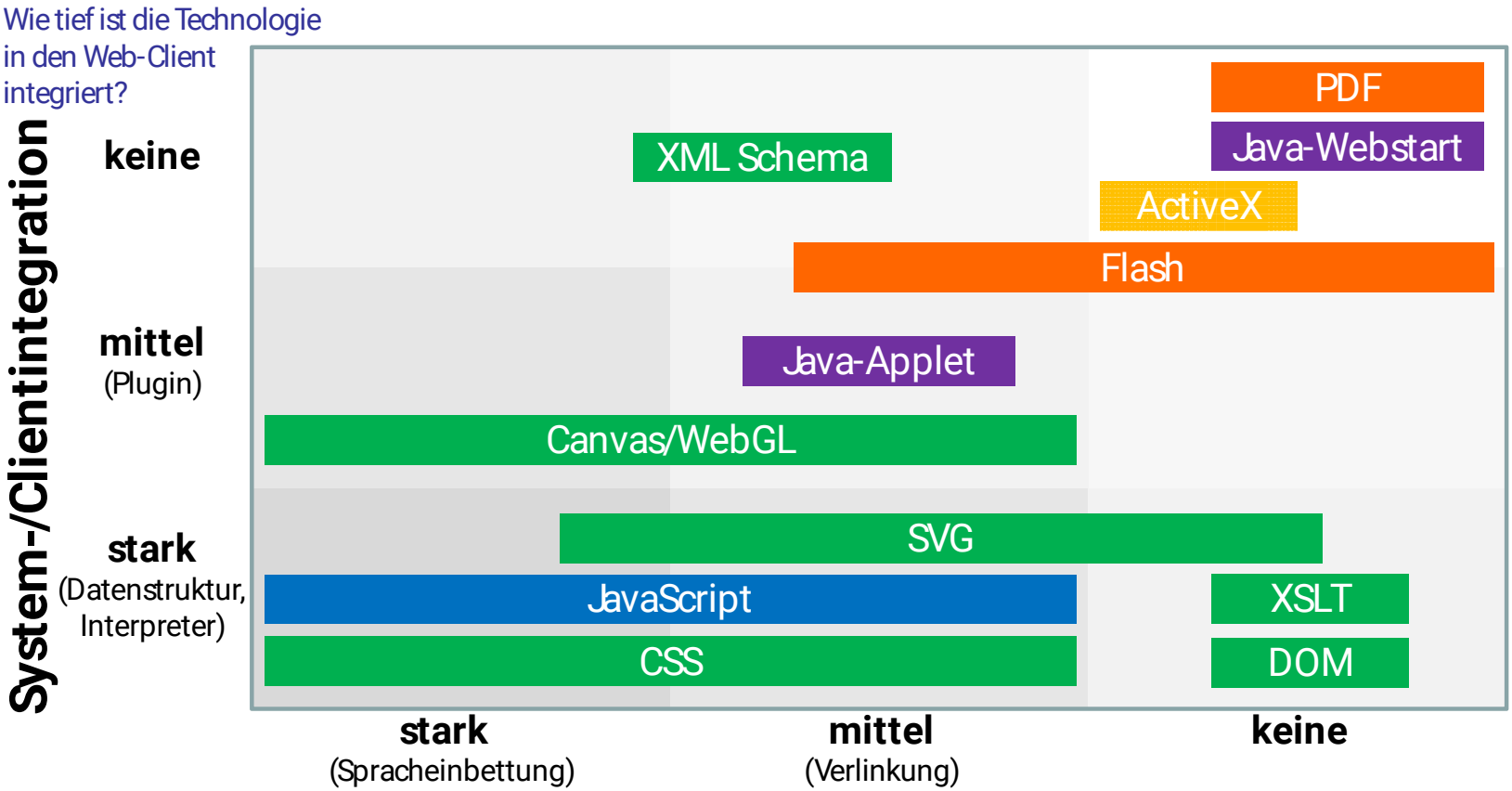
Verteilung der Verarbeitung	Verarbeitete Datenmenge	Art der Anwendung	Server-Last	Netz-Last	Client-Last
serverseitig	klein	einmalig	0	+	++
		interaktiv	- -	- -	++
	groß	einmalig	-	+	++
		interaktiv	- -	- -	++
Teilung der Aufgaben (Server: Ergebnisberechnung, Client: Berechnung der Darstellung)	klein	einmalig	0	++	+
		interaktiv	-	-	+
	groß	einmalig	0	+	+
		interaktiv	-	-	+
clientseitig	klein	einmalig	++	++	0
		interaktiv	++	++	-
	groß	einmalig	+	-	-
		interaktiv	+	-	- - -

einmalig: Einmalige Verarbeitung der Daten **interaktiv:** Jeder Interaktionsschritt erfordert Neuberechnung

Browser Module



Einordnung von Client-Technologien



Nach B. Stein, 2012

Interpretation von Sprachen

- Interpreter
  - Lesen und Ausführen eines Satzes (Programm) einer Sprache

	strikt interpretiert[1]	neuere Interpreter[2]
statische Semantik (& Prüfung dieser vor Ausführung)	Nein	Ja
Prüfung von nicht ausgeführten Programmteilen	Nein	Ja
Prüfung von Bindungs- und Typregeln vor Ausführung	Nein	Je nach Sprache
Generation einer expliziten internen Repräsentation des Satzes/Skripts	Nein	Ja

[1] z.B. Prolog und interpretiertes Lisp [2] z.B. JavaScript, PHP und Perl

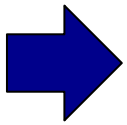
  - Einfache Integration in andere Software (wie z.B. Web-Browser)
  - weitere Kapselung der Programmausführung
  - gegenüber der Ausführung von übersetztem Maschinencode zumeist 10-100 mal zeitaufwändiger

JavaScript - Eigenschaften

- Einfluss anderer Programmiersprachen
  - abgeleitet von Perl
  - C/C++/Java ähnliche Notation
  - aber wenig Bezug zu Java
- interpretiert
- **dynamisch** typisiert
- spezielle objektorientierte Konzepte
- Interpreter in Web-Browsern integriert
- eng verknüpft mit HTML
  - Zugriff auf Elemente des dargestellten Dokuments via DOM-API

Beispiel: Webanwendung mittels Javascript

```
<!DOCTYPE html>
<html>
 <head>
  <title>JavaScript Example</title>
  <script type="text/javascript">
   var number = Math.floor(Math.random()*6)+1;
   function check() {
    var guessed = document.GuessForm.inputfield.value;
    if(guessed==number){
     alert("Congratulations! The dice value is "
           + number + ".");
    } else {
     if(number<guessed) alert("My number is smaller!");
     else alert("My number is bigger!");
    }
   }
  </script>
 </head>
 <body>
  I have thrown a dice. Please guess the number:
  <form name="GuessForm">
   <input type="number" name="inputfield"
                size="1" min="1" max="6"/>
   <input type="button" value="Guess!" onClick="check()"/>
  </form>
 </body>
</html>
```



I have thrown a dice. Please guess the number:

Guess!

Einbindung von JavaScript in HTML

1. Als Script-Bereich innerhalb eines HTML-Dokuments:

```
<script type="text/javascript">
… // your javascript code goes here…
</script>
```

2. Innerhalb von HTML-Tags (zur Definition von Bedienern, die bei bestimmten Ereignissen (Events) ausgelöst werden):
   a. Kodierung des Ereignisses als Attribut: Attributwert enthält eine Anweisungsfolge, die beim Eintritt des Ereignisses ausgeführt wird:

```
<input type="button" value="…" onClick="checkGuess()"/>
```

   b. Angabe einer Anweisungsfolge in einem Anker-Element (anstatt einer URL) mit `javascript:`, die beim Klicken ausgeführt wird:

```
<a href="javascript:checkGuess()">…</a>
```

3. In einer separaten Datei:

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

Bemerkungen zum Einbinden von JavaScript-Code

- Mehrfache Verwendung des `<script>`-Elementes möglich
- Sofortige Ausführung des JavaScript-Codes beim Einlesen des Dokumentes vom Browser
  - Keine sofortige Ausführung von Funktionsdefinitionen sowie Ereignisbedienern
- Angabe mehrerer Anweisungen in HTML-Attributen durch Semikola (;) getrennt möglich, aber unübersichtlich
- Ladezeitverringerung durch Platzierung von JavaScript-Bereichen **am Ende** des Dokumentes
- Separate Datei (ausschliesslich) mit JavaScript-Code: ASCII-Datei mit Dateinamenerweiterung `.js`

Grundlagen der Syntax

Bezeichner

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

- einheitliche Schreibweise für alle Arten von Bezeichnern
- Unterscheidung von Klein-/Großschreibung (Case Sensitive)

Anweisungen

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

- **optionales** Semikolon am Zeilenende
- Semikolon erforderlich zwischen Anweisungen in derselben Zeile
- Eine Anweisungsfolge definiert keinen Scope:
  eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in umgebender Funktion bzw. im umgebenden Programm

Bedingte Anweisung

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

- Bei einzelnen Anweisungen sind die `{}`-Klammern optional

while-Schleife

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

for-Schleife

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

return-Anweisung

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

Parameterübergabe

- standardmäßig mittels **call-by-value**

Kommentare

- Einzeilig: `// Kommentar`
- Mehrzeilig: `/* Kommentar … */`

Variablen

- Variable kann Werte **beliebigen** Typs annehmen
- In der Regel Deklaration mit dem Schlüsselwort **var**
- Lokale Variable
  - Deklaration innerhalb einer Funktion
  - Geltungsbereich: nur innerhalb ihrer Funktion
- Globale Variable
  - Deklaration außerhalb von Funktionen
  - Deklaration innerhalb einer Funktion ohne Verwendung des Schlüsselwortes var: **Achtung**, Fehlerquelle!
  - Geltungsbereich: ganzes Programm
    - Ausnahme: Überdeckung von einer lokalen Variable
- Initialisierung durch Zuweisung eines Anfangswertes in Form eines Literals, einer anderen Variable oder einer komplexen Formel

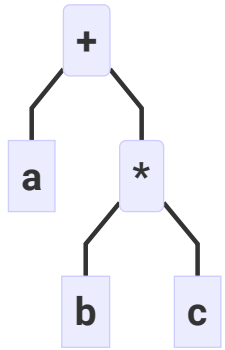
Markiere die Geltungsbereiche von Variablen


```
1  var number = Math.floor(Math.random()*6)+1;
2  function checkGuess() {
3   var guessed = document.GuessForm.inputfield.value;
4   function compare(number, guessed){
5     if(number == guessed){
6       return true;
7     } if(guessed < number){
8       compare = -1;
9     } else {
10      compare = 1;
11    }
12    return false;
13  }
14  isEqual = compare(number, guessed);
15  if(isEqual){
16   alert("Congratulations! The dice value is "+number+".");
17  } else {
18   if(compare > 0){
19    alert("My number is smaller!");
20   } else { alert("My number is bigger!");}
21  }
22 }
```

Operatoren: Präzedenz, Assoziativität

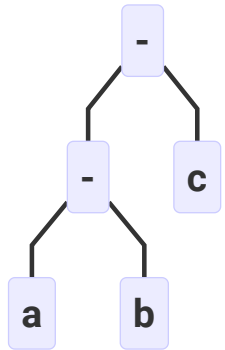
- **Höhere Bindung** seiner Operanden durch einen Operator mit höherer **Präzedenz**
  - Explizite Angabe der Operandenbindung durch Klammerung möglich

Beispiel:
$a + b * c$



- **Linksassoziativität** von Operatoren bei gleicher Präzedenz:
  - Stärkere Bindung des linken Operators als der rechte

Beispiel:
$a - b - c$



- **Rechtsassoziativität**: analog

Präzedenz & Assoziativität in JavaScript

Präzedenz	Stelligkeit	Assoziativität	Operatoren	Bemerkung
1	2	rechts	`= += -= <<= >>= &= ^= |=`	Zuweisungsoperatoren
2	3	rechts	`?:`	bedingter Ausdruck/Elvis-Operator
3	2	links	`||`	logische Disjunktion
4	2	links	`&&`	logische Konjunktion
5	2	links	`|`	Bitoperator (Inklusiv-Oder)
6	2	links	`^`	Bitoperator (Exklusiv-Oder)
7	2	links	`&`	Bitoperator (Und)
8	2	links	`== != === !==`	(Un-)Gleichheit, (Nicht-)Identität
9	2	links	`< <= > >= in instanceof`	Ordnungs-/Existenz-/Klassenprüfung
10	2	links	`<< >> >>>`	Shift-Operatoren
11	2	links	`+ -`	Konkatenation, Add., Subtr.
12	2	links	`* / %`	Arithmetik
13	2	rechts	`! ~ - ++ -- typeof void delete`	Präfixoperatoren
14	2	links	`++ --`	Postfixoperatoren
15	2	links	`()`	Funktionsaufruf

Reihenfolge der Auswertung

von Ausdrücken unter Berücksichtigung der Präzedenz und Assoziativität von JavaScript


```
a = b = 5
```

```
5 == num / 2 && (2 + 2 * num).toString() == '22'
```

Primitive Datentypen

number

- **Keine** Unterscheidung (nach außen) zwischen
  - Ganzzahlen gespeichert in 4 Byte
  - Gleitpunktzahlen gespeichert in 8 Byte nach IEEE754
- Wert `NaN` (not a number): ein undefiniertes Ergebnis
- Wert `Infinity`: Wert, der größer als die größte repräsentierbare Zahl ist

string

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

boolean

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

Spezielle Werte

`undefined`	• Bedeutung: Variable hat **keinen** Wert • Rückgabe falls - Benutzung einer (zuvor deklarierten) Variable, der (bis jetzt) **kein** Wert zugewiesen wurde - Zugriff auf nicht-existente Objektkomponente (s.u.)
`null`	• Bedeutung: Variable hat **keinen gültigen** Wert

- **Aber:** Vergleich `undefined == null` ergibt `true`

Funktionsdatentyp

- Funktionen sind Objekte vom Typ `Function`

	Erzeugung von Funktionen
function-Statement	`function f(x, y) { return x * y; }`

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

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

Objekt-Datentyp

Datentyp	Alle Objekte sind vom Datentyp `Object`
Bestandteile	Komponenten, die jeweils Namen und Wert besitzen
	Erzeugung
Objektliteral	`var student = { matrNr:4711, name:"P. Smell" };`
Standardkonstr. `Object()`	`var student = new Object();`

Benutzer-definierter Konstruktor `Object(…)`	`function MyStudent(m, n) { // Nameskonvention: Groß!`
	`  this.matrNr = m; // this = aktuelle Objektinstanz`
	`  this.name = n;`
	`}`
	`var student = new MyStudent(4711,"P. Smell");`

	Hinzufügen von weiteren/Abändern bestehender Komponenten
Zuweisung	`student.semester = 5; student.matrNr = 4711;`
	Zugriff auf Komponenten
Pfadausdruck	mittles *Objekt.Komponente*: `student.matrNr` liefert `4711`

Methoden

- Funktionen als Objektkomponenten heißen Methoden
  - Bei Aufruf i.d.R. Lesen und Verändern der Objektkomponenten
- Alle andersartigen Komponenten heißen Eigenschaften

Definition

```
function MyCircle(r) {
 this.radius=r;
 this.area=getArea; // Zuweisen von globaler Fkt. 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("Area = " + c.area());
document.writeln("Circumference = " + c.circ());}
```

Klassen und ECMAScript 6

- ECMA standardisiert ECMAScript (Name des offiziellen Standards), JavaScript ist dessen meistbekannteste Implementation
- Weitergehende objektorient. Sprachkonstrukte ab ECMAScript 6
  - Wird **nicht** von jedem Browser unterstützt
  - Alte und neue Sprachkonstrukte parallel verwendbar

```
class Shape {
  constructor(x_pos,y_pos) {
    this.x = x_pos;
    this.y = y_pos;
  }
}
class MyCircle extends Shape {
  constructor(x_pos,y_pos,r) {
    super(x_pos,y_pos);
    this.radius=r;
  }
  getArea() {
    return (Math.PI * this.radius * this.radius);
  }
}
var c = new MyCircle(3);
console.log(c.getArea());
```

Datentypen: Arrays

Datentyp	Jedes Feld (Array) ist vom Datentyp `Array`
Bedeutung	Abbildung von Indizes auf Werte
Einträge	Paare (k, v) mit k numerischer oder String-Schlüssel und zugeordneter Wert v
Erzeugung von Arrays mit dem Konstruktor `Array( )`	
Liste von Werten	`// indiziert von 0 an:` `monatsName = new Array("", "Jan", …, "Dez");`
Erweiterung eines leeren Arrays	`monatsName = new Array();` `monatsName[1]= "Jan"; monatsName[2]= "Feb"; …`

Assoziatives Array	`monatsNr = new Array();`
	`monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; …`
	• fügt Eigenschaften `"Jan"`, `"Feb"` etc. zum Objekt `monatsNr`
	• „Ass. Array“ mittels Objektliteral:
	`monatsNr = {jan:1,feb:2,…};`

Zugriff	
Aufzählung aller Elemente	`for(n in monatsNr) { console.log(n+"=>"+monatsNr[n]); }`

Funktionsbibliothek

- Implementation des Großteils der Funktionsbibliothek in Form von Objekten
- ergänzt durch vordefinierte Funktionen

Art	Beschreibung	Beispiele
Kern-objekte	Funktionalität unabhängig von Browser und Dokument	`Array`, `Date`, `Function`, `Math`, `Object`, `RegExp`
DOM-Objekte	Repräsentation von und Zugriff auf Dokumente. Standardisierte DOM-Objekte nach der W3C Spezifikation mit Zugriff über DOM-API.	`Document`, `Element`, `Node`
Browser-Objekte	Repräsentation von und Zugriff auf Browser-Fenster	`History`, `Location`, `Navigator`, `Screen`, `Window`
Vordefi-nierte Funktionen	Funktionalität unabhängig von Browser und Dokument	`eval`, `isFinite`, `isNaN`, `parseInt`

Bemerkungen zur Funktionsbibliothek

Objekt	Beschreibung
`window`	• höchstes Objekt und Ausgangspunkt der DOM- und Browser-Objekte-Hierarchie
`document`	• Abkürzung für `window.document` • Wurzelknoten des HTML/XML-Dokuments • oberstes Ausgangsobjekt für das Document Object Model (DOM)
`documentElement`	• Abkürzung für `window.document.documentElement` • Wurzelelement des HTML/XML-Dokuments

- Wert einer Eigenschaft kann wiederum ein Objekt sein
  $\Rightarrow$ Mittels Pfadausdruck Zugriff auf tief inneliegende verschachtelte Eigenschaften: `window.document.images.length`
- verbindliche Standards in HTML 5 für Browser-Objekte
- Alternative Bezeichnungen für Kernobjekte (Global Objects, Built-in Objects, Objects in the Global Scope) und vordef. Funktionen (Global Functions, Built-in Functions)

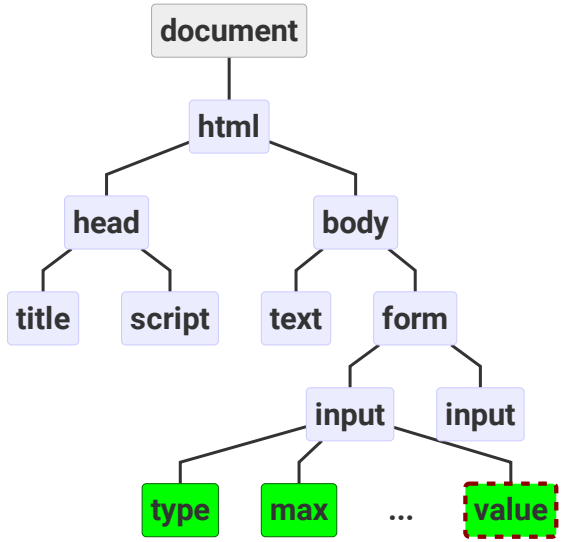
Funktionsbibliothek: Zugriff auf HTML-Elementobjekte und deren Komponenten

- Eindeutig qualifizierender Name gemäß DOM-Dokumentbaum
- DOM-API, um Elementmenge auszuwählen
  - `getElementsByName()`
  - `getElementById()`
  - `getElementsByTagname()`
  - …
- Vordefinierte Arrays, die alle Elemente eines Typs enthalten
  - `document.forms`
  - `document.images`
  - …
- Kombiniert: Auswahl von Elementmenge plus weitere Spezialisierung mittels qualifizierendem Namen

Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

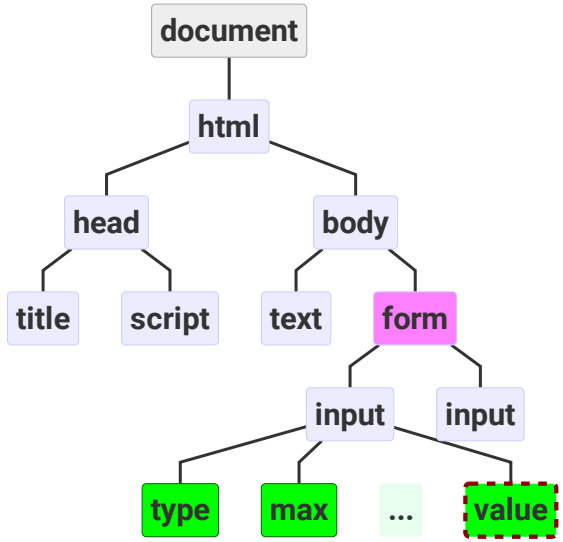
```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
  <script type="text/javascript">
    var number = Math.floor(Math.random()*6)+1;
    function checkGuess() { … }
  </script>
</head>
<body>
  I have thrown a dice. Please guess the number:
  <form name="GuessForm" id="GF">
    <input type="number"
           name="inputfield"
           size="1" min="1" max="6"/>
    <input type="button" value="Guess!"
           onClick="checkGuess()"/>
  </form>
</body>
</html>
```



Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



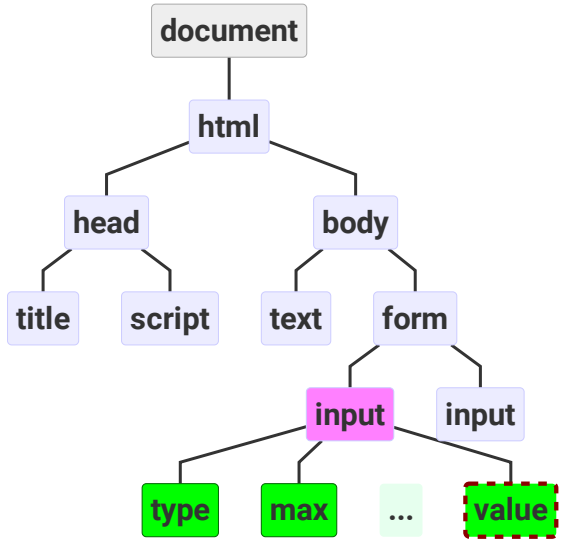
Zugriffsmöglichkeiten auf das (erste) Formelement:

`document.GuessForm`	qualifizierender Name
`document.getElementsByTagName("form")[0]`	DOM-API
`document.getElementById("GF")`	DOM-API

Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



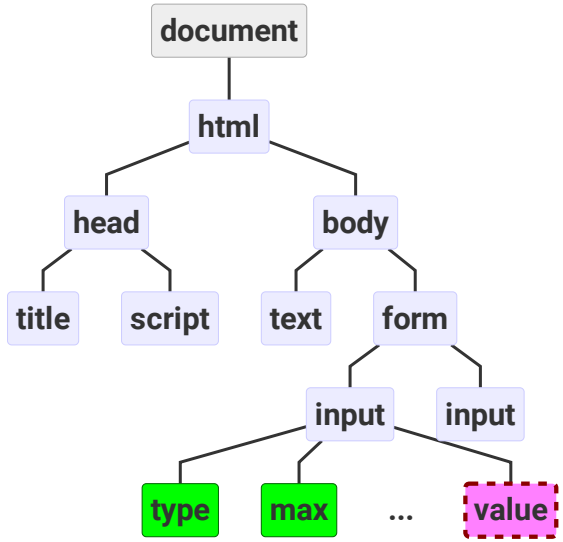
Zugriffsmöglichkeiten auf das (erste) Eingabeelement:

`document.GuessForm.inputfield`	qualifizierender Name
`document.getElementsByTagName("form")[0].inputfield`	DOM-API
`document.getElementsByName("inputfield")[0]`	DOM-API

Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



Zugriffsmöglichkeiten auf das Eingabefeld im ersten Eingabeelement:

```
document.GuessForm.inputfield.value
document.getElementsByTagName("form")[0].inputfield.value
document.getElementsByName("inputfield")[0].value
```

Bemerkungen zu DOM-Objekten

- Zugriffe erfolgen gemäß der DOM-API
  - über Methoden des generischen `Node`-Interfaces wie `getChildNodes()`
  - über Methoden des `Document`-Interfaces wie `getElementsByName()`, `getElementsByTagName()` oder `getElementById()`
- Jedes erlaubte Attribut eines HTML-Elements: Eigenschaft im entsprechenden DOM-Objekt
  - HTML-Element `<input>` hat ein erlaubtes Attribut value
    $\Rightarrow$ DOM-Hierarchie enthält `input`-Elementobjekt mit der Eigenschaft `value`
- Setzen von Werten analog:
  document.GuessForm.Eingabe.value = 2;
- DOM-Methoden für einige der Element-Objekte
  - Z.B. `submit()` und `reset()` im DOM-Objekt des HTML-Elements `<form>`

Ereignisbehandlung

- Ereignis (Event): Wahrnehmung einer Zustandsänderung
  - Ereignisgetriebene Programmierung ordnet Ereignissen Operationen zu
  - Setzen von Bedienern für ein Ereignis:

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

Ereignisbehandlung: wichtige Ereignisse

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

- Ereignisbehandlung (z.B. von Maus-Ereignissen) weitgehend auch mittels **CSS** realisierbar

Quellen zum Nachlernen und Nachschlagen im Web

- ECMA, Standard ECMA-262: ECMAScript Language Specification
- MDN, JavaScript
- SELFHTML e.V. JavaScript
- Tarquin, JavaScript Tutorial
- W3 Schools, JavaScript Tutorial

Zusammenfassung

- Einordnung Client-Technologien
- Interpreter von Sprachen
- Einführung in JavaScript
  - Eigenschaften
  - Anwendungen
  - Sprachkonstrukte
  - Variablen, Datentypen
  - DOM-API (Zugriff, nicht behandelt: Modifikation der Baumstruktur)
  - Ereignisbehandlung
- In der nächsten Vorlesung Vertiefung von JavaScript
  - Ajax
  - JQuery als Beispiel einer gängigen Funktionsbibliothek
  - Web Components als Zukunftstechnologie