Kapitel WT:V

V. Client-Technologien

- Web-Clients
- Exkurs: Programmiersprachen
- JavaScript
- JavaScript Web-APIs
- WebAssembly

JavaScript

Einführung [Einordnung]

Charakteristika:

- interpretiert, dynamisch typisiert
- einfache objektorientierte Konzepte
- Notation ähnlich C++ und Java, konzeptuell 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 ähnlich C++ und Java, konzeptuell 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

Einführung (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Function</title>
    <script>
      function quadrat() {
        let zahl = document.quadratForm.eingabe.value;
        let ergebnis = zahl * zahl;
        alert ("Das Quadrat von " + zahl + " = " + ergebnis);
      }
    </script>
  </head>
  <body>
    <form name="quadratForm" action="">
      <input type="text" name="eingabe" size="3">
      <input type="button" value="Quadrat errechnen" onclick="quadrat()">
    </form>
  </body>
</html>
```

JavaScript

Einführung (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Function</title>
    <script>
      function quadrat() {
        let zahl = document.quadratForm.ei
        let ergebnis = zahl * zahl;
        alert ("Das Quadrat von " + zahl +
      }
    </script>
  </head>
  <body>
    <form name="quadratForm" action="">
      <input type="text" name="eingabe" si
      <input type="button" value="Quadrat
    </form>
  </body>
</html>
```

[JavaScript: Aufruf]

Bemerkungen:

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

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 Mocha, dann 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 Mocha, dann 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 Mocha, dann 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. [Wikipedia]

2016 Übersicht über die Browser-Unterstützung. [kangax]

2022 Statistiken zur Verbreitung: [tiobe.com] [redmonk.com: Q1’22, history, process]

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]
- Ecma International ist eine private, internationale Organisation zur Normung von Informations- und Kommunikationssystemen. Der Name wurde 1994 geändert; der Namensteil „Ecma“ hat seine Bedeutung als Abkürzung (ursprünglich für European Computer Manufacturers Association) verloren. [Wikipedia]

JavaScript

Einbindung in HTML-Dokumente [SELFHTML]

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

```
<script>
  ...
</script>
```

JavaScript

Einbindung in HTML-Dokumente [SELFHTML]

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

```
<script>
  ...
</script>
```

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

   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" ... onclick="quadrat()"> [Einführungsbeispiel]
   ```

   (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: Aufruf 1, Aufruf 2]:

```
<script>
  ...
</script>
```

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

   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" ... onclick="quadrat()"> [Einführungsbeispiel]
   ```

   (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 [JavaScript: js-Datei, Aufruf]:

```
<script src="usage.js"></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 Deklarationen von Variablen und Funktionen im gesamten, zugehörigen Code-Kontext. [MDN]

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 Deklarationen von Variablen und Funktionen im gesamten, zugehörigen Code-Kontext. [MDN]

JavaScript

Variablen: Illustration von Geltungsbereichen

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

function compute(n)
{
  let sum = n;
  if (col > maximum) {
    var m1 = maximum;
    let m2 = m1;
    col = m2;
  }
  sum = sum * col;
  return sum;
}
```

JavaScript

Variablen: Illustration von Geltungsbereichen

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

function compute(n) // n ist lokale Variable.
{
  let sum = n;        // sum ist lokal, Blockkontext = Funktionskontext.
  if (col > maximum) {
    var m1 = maximum;  // m1 ist lokal und in der Funktion sichtbar.
    let m2 = m1;
    col = m2;
  }
  sum = sum * col;    // col ist globale Variable.
  return sum;
}
```

JavaScript

Variablen: Illustration von Geltungsbereichen

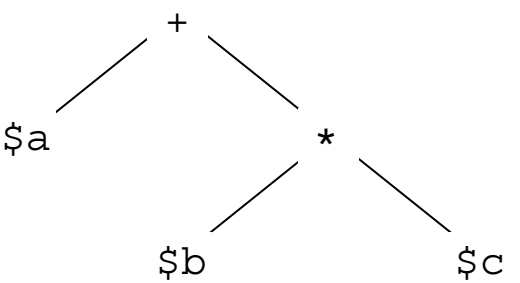
```
let line, sum;
let col = 2;
let minimum = col,
    maximum = 999;

function compute(n) // n ist lokale Variable.
{
  let sum = n;        // sum ist lokal, Blockkontext = Funktionskontext.
  if (col > maximum) {
    var m1 = maximum;  // m1 ist lokal und in der Funktion sichtbar.
    let m2 = m1;       // m2 ist lokal und im zugehörigen Block sichtbar.
    col = m2;
  }
  sum = sum * col;   // col ist globale Variable.
  return sum;
}
```

JavaScript

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
```

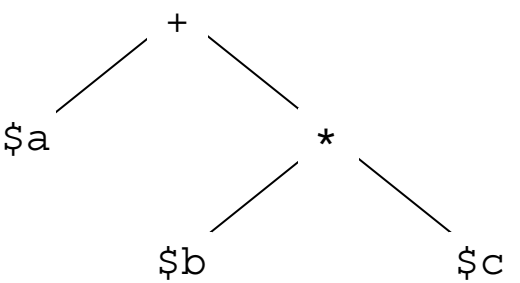


JavaScript

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
```



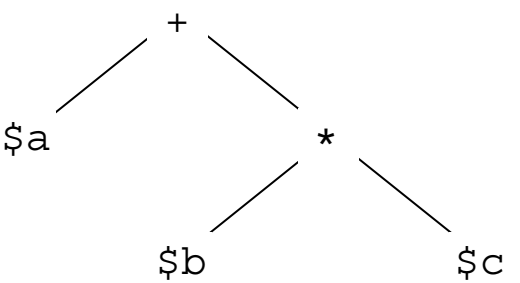
Ein Operator ist linksassoziativ (rechtsassoziativ), wenn beim Zusammentreffen von Operatoren gleicher Präzedenz der linke (rechte) Operator seine Operanden stärker bindet als der rechte (linke).

JavaScript

Operatoren: Präzedenz, Assoziativität

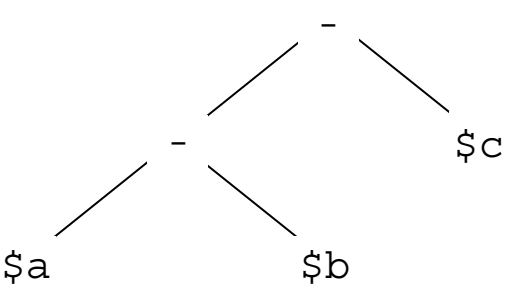
Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

```
$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

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: `let 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.

$\rightsquigarrow$ Für eine typsichere Prüfung auf Gleichheit muss der Operator `===` verwendet werden bzw. `!==` für Ungleichheit.

JavaScript

Datentypen: Funktionen [MDN]

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

(a) Als „klassische“ Funktionsdeklaration:

(b) Als 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`. Definitionsvarianten:

(a) Als „klassische“ Funktionsdeklaration:

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

(b) Als 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`. Definitionsvarianten:

(a) Als „klassische“ Funktionsdeklaration:

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

(b) Als Funktionsausdruck bzw. Funktionsliteral [kangax]:

```
let p = (x, y) => { return x*y; };          // arrow (anonym)
```

oder

```
let q = function(x, y) { return x*y; };     // klassisch, anonym
```

oder

```
let r = function g(x, y) { return x*y; };   // klassisch, 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`. Definitionsvarianten:

(a) Als „klassische“ Funktionsdeklaration:

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

(b) Als Funktionsausdruck bzw. Funktionsliteral [kangax]:

```
let p = (x, y) => { return x*y; };          // arrow (anonym)
```

oder

```
let q = function(x, y) { return x*y; };     // klassisch, anonym
```

oder

```
let r = function g(x, y) { return x*y; };   // klassisch, benannt
```

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

```
let s = new Function("x", "y", "return x*y;");
```

Bemerkungen:

- Funktionen in JavaScript sind First-Class-Objekte und können gespeichert, als Parameter übergeben, zur Laufzeit eines Programms erstellt werden. [Wikipedia] [MDN]
- Im Funktionskontext, d.h., bei „klassischen“ Funktionen wird das Schlüsselwort `this` abhängig vom Ort des Funktionsaufrufs gebunden. Es zeigt auf den umgebenden Kontext (Aufruf als Funktion), das neue Objekt (Aufruf als Konstruktor mit `new`) oder das Ziel des Events (Aufruf als Event-Handler). [MDN]
- Besonderheiten zu Arrow-Funktionen [MDN]:
  - Arrow-Funktionen können nicht als Konstruktor verwendet werden.
  - Besteht der Funktionskörper aus nur einer einzigen Anweisung, so ist die Verwendung der geschweiften Klammern optional und das `return`-Schlüsselwort implizit.
  - Enthält die Parameterliste nur einen Parameter, so ist die Verwendung der runden Klammern optional.

  Beispiel:

```
f = (x) => x*x;     // f(2) ⇝ 4
f =  x  => x*x;     // f(2) ⇝ 4
```

JavaScript

Datentypen: Objekte [vordefinierte Objekte]

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

(a) Mit einer „klassischen“ Funktionsdeklaration als Konstruktor:

(b) Als Objektliteral [w3schools]:

(c) Mit der Funktion `Object()` als Konstruktor:

```
year: 2020

brand: "Tesla"
```

JavaScript

Datentypen: Objekte [vordefinierte Objekte]

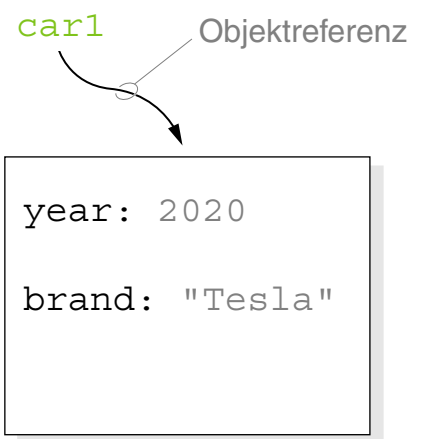
Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`. Definitionsvarianten für Objekte:

(a) Mit einer „klassischen“ Funktionsdeklaration als Konstruktor:

```
function mycar(x) { this.year = x; return x*x };
let car1 = new mycar(2020);
car1.brand = "Tesla"
```

(b) Als Objektliteral [w3schools]:

(c) Mit der Funktion `Object()` als Konstruktor:



JavaScript

Datentypen: Objekte [vordefinierte Objekte]

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

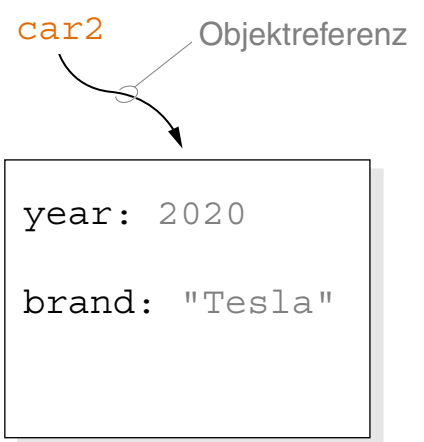
(a) Mit einer „klassischen“ Funktionsdeklaration als Konstruktor:

```
function mycar(x) { this.year = x; return x*x };
let car1 = new mycar(2020);
car1.brand = "Tesla"
```

(b) Als Objektliteral [w3schools]:

```
let car2 = { year:2020, brand:"Tesla" };
```

(c) Mit der Funktion `Object()` als Konstruktor:



JavaScript

Datentypen: Objekte [vordefinierte Objekte]

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

(a) Mit einer „klassischen“ Funktionsdeklaration als Konstruktor:

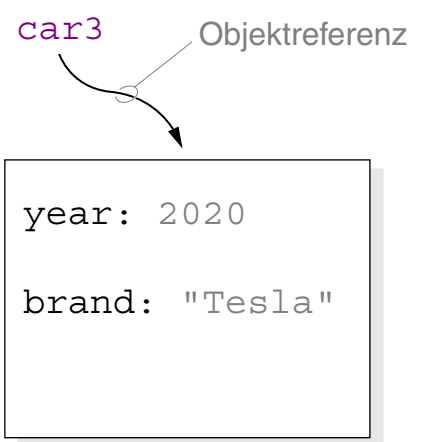
```
function mycar(x) { this.year = x; return x*x };
let car1 = new mycar(2020);
car1.brand = "Tesla"
```

(b) Als Objektliteral [w3schools]:

```
let car2 = { year:2020, brand:"Tesla" };
```

(c) Mit der Funktion `Object()` als Konstruktor:

```
let car3 = new Object();
car3.year = 2020;
car3.brand = "Tesla";
```



JavaScript

Datentypen: Objekte (Fortsetzung)

(d) Mit einem durch `class` definierten Konstruktor [MDN]:

```
class MyCar {
  constructor(x, y) {
    this.year = x;
    this.brand = y;
  }
}
let car4 = new MyCar(2020, "Tesla");
```

JavaScript

Datentypen: Objekte (Fortsetzung)

(d) Mit einem durch `class` definierten Konstruktor [MDN]:

```
class MyCar {
  constructor(x, y) {
    this.year = x;
    this.brand = y;
  }
}
let car4 = new MyCar(2020, "Tesla");
```

Zugriff auf Objektkomponenten mit *Objektausdruck.Komponente*:

`car1.year` ⇝ `2020`

...

`car4.year` ⇝ `2020`

Bemerkungen (Prototypen versus Klassen) :

- JavaScript ist keine objektorientierte Progammiersprache im Sinne von Java oder C++. Es gibt zwar den Datentyp `Object`, das Vererbungskonzept baut allerdings auf Objekt-Prototypen und nicht auf einem (abstrakten) Klassensystem auf.
  In JavaScript kann jedes Objekt als „Prototyp“ (zur Konstruktion neuer Objekte) verstanden werden, wodurch keine strikte Unterscheidung zwischen Klassen und Instanzobjekten existiert: Objekte lassen sich kopieren, Komponenten lassen sich hinzufügen und Vererbungshierchachien aufbauen. Stichwort: *prototypbasierte Vererbung* [MDN] [O'Reilly]
- Im Klassenkontext, also in Konstruktoren und Methoden wird auf die Komponenten (Eigenschaften) mit dem Qualifier `this` zugegriffen; er bezeichnet eine mit `new` erzeugte Objektinstanz.
- *Objektausdruck* ist ein JavaScript-Ausdruck, der zu einer Referenz auf ein JavaScript-Objekt evaluiert.
- Objektkomponenten können Funktionen sein und heißen dann Methoden.
  - Aufruf von Methoden: *Objektausdruck.Komponente*`()`
  - Zugriff auf Wert der Eigenschaft: *Objektausdruck.Komponente*

JavaScript

Datentypen: Objekte (Fortsetzung)

Definition von Methoden als Funktionsausdruck oder als Funktion:

```
class MyCircle {

  constructor(r) {
    this.radius = r;
    this.circ = () => { return (Math.PI * this.radius * 2); };
  }

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

JavaScript

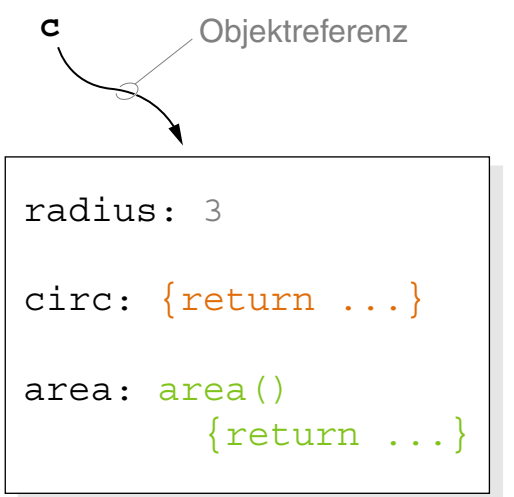
Datentypen: Objekte (Fortsetzung)

Definition von Methoden als Funktionsausdruck oder als Funktion:

```
class MyCircle {

  constructor(r) {
    this.radius = r;
    this.circ = () => { return (Math.PI * this.radius * 2); };
  }

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

Aufruf [JavaScript: Aufruf]:

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



JavaScript

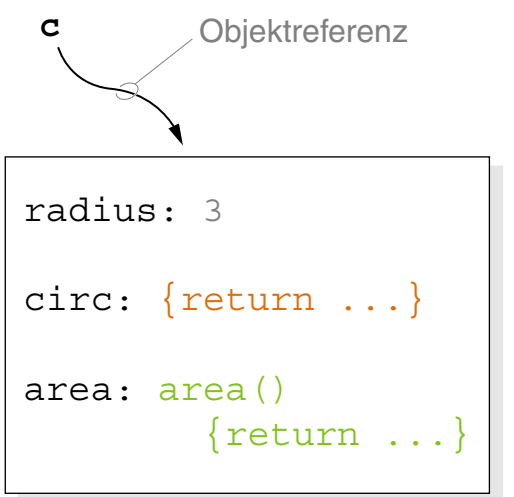
Datentypen: Objekte (Fortsetzung)

Definition von Methoden als Funktionsausdruck oder als Funktion:

```
class MyCircle {

  constructor(r) {
    this.radius = r;
    this.circ = () => { return (Math.PI * this.radius * 2); };
  }

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

Aufruf [JavaScript: Aufruf]:

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

document.writeln("Area = " + c.area);
document.writeln("Circumference = " + c.circ);
```



Bemerkungen (Built-in Objects) :

- Ein Großteil der JavaScripts Kernfunktionalität ist in Form von vordefinierten Objekten implementiert. Beispiele: `Array, Date, Function, Math, Object, RegExp`.

  Sprachreferenzen: [MDN] [w3schools]
- Vordefinierte Objekte werden auch als *Global Objects*, *Objects in the Global Scope* oder *Built-in Objects* bezeichnet.
- Übersicht über die (wenigen) vordefinierten JavaScript-*Funktionen* : [MDN]

JavaScript

Datentypen: Arrays [PHP] [JavaScript: Aufruf]

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: Aufruf]

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:

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

(b) Durch Erweiterung eines leeren Arrays:

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

(c) Als assoziatives Array:

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

JavaScript

Datentypen: Arrays [PHP] [JavaScript: Aufruf]

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:

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

(b) Durch Erweiterung eines leeren Arrays:

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

(c) Als assoziatives Array:

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

Aufzählung aller Elemente mit Schlüssel:

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

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge bzw. Block:
- Bedingte Anweisung:
- Return-Anweisung:
- `while`-Schleife:
- `for`-Schleife [JavaScript: Aufruf] :

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge bzw. Block:

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

  Eine Anweisungsfolge definiert nur für `let`-Deklaration einen *Scope*. Eine `var`-Deklaration gilt auch 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: Aufruf] :

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge bzw. Block:

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

  Eine Anweisungsfolge definiert nur für `let`-Deklaration einen *Scope*. Eine `var`-Deklaration gilt auch 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: Aufruf] :

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

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge bzw. Block:

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

  Eine Anweisungsfolge definiert nur für `let`-Deklaration einen *Scope*. Eine `var`-Deklaration gilt auch 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: Aufruf] :

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

- Parameterübergabe standardmäßig mittels call-by-value.

Kapitel WT:V

V. Client-Technologien

- Web-Clients
- Exkurs: Programmiersprachen
- JavaScript
- JavaScript Web-APIs
- WebAssembly

JavaScript Web-APIs

Es existieren viele spezialisierte Web-APIs, die typischerweise – aber nicht zwangsläufig – im Zusammenhang mit JavaScript verwendet werden. Auswahl:

- Accelerometer API
- Bluetooth API
- Canvas API
- (1) DOM API
- Encoding API
- Fullscreen API
- Geolocation API
- History API
- Image Capture API
- JSON Base API
- Keygen API
- Long Tasks API
- Media Session API
- Navigation Timing API
- Orientation Sensor API
- Permissions API
- QR Code API
- Reporting API
- Sensor API
- Touch Events API
- URL API
- Vibration API
- WebRTC API
- Xkcd API
- YouTube API
- Zipstatic API

Übersichten im Web: [MDN] [W3C] [w3schools] [apilist.fun]

JavaScript Web-APIs

(1) DOM-API

Die DOM-API dient zum Zugriff und zur Manipulation von DOM-Objekten. Die DOM-Objekte repräsentieren Dokument und Browser im Speicher des Web-Clients und implementieren die standardisierten Interfaces der DOM-API.

Interface-Referenzen:

Interface	Kern-DOM	DOM HTML
(alle)	[W3C] [WHATWG] [MDN]	[W3C] [WHATWG] [MDN]
`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]

Seit 2021 ersetzt die WHATWG-Dokumentation die W3C-Dokumentation für die DOM-API.

Bemerkungen:

- DOM-Objekte implementieren die Interfaces der sprachunabhängigen DOM-API. Dabei wird zwischen einem Kern-DOM und dem DOM für HTML unterschieden. [MDN] [SELFHTML]
- 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] [WHATWG]
- 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] [WHATWG] [MDN]
- Illustration von Markup, DOM und gerenderter HTML-Seite im Live-DOM-Viewer. [hixie.ch]

JavaScript Web-APIs

(1) DOM-API: Objektzugriff

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

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

   Einführungsbeispiel: `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`

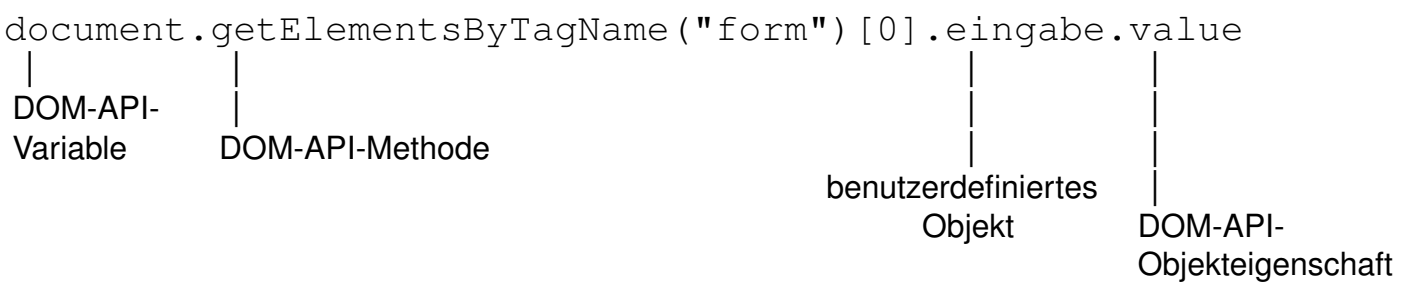
Bemerkungen:

- 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.

  In der Folge stehen alle für ein HTML-Element erlaubten Attribute als Objekteigenschaften im DOM zur Verfügung. Diese Datenstruktur bildet die Dokumentrepräsentation im Browser.

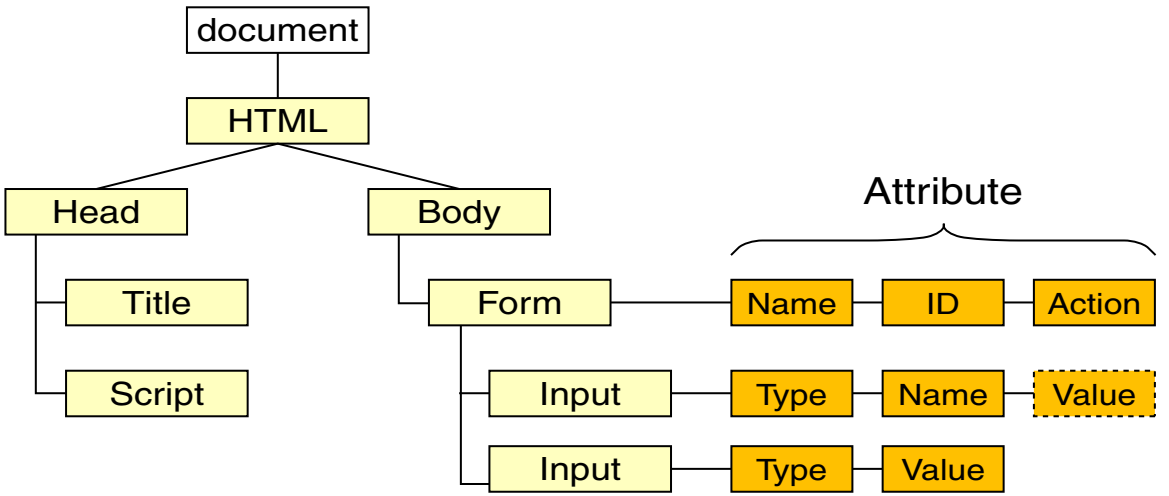
- *Elementausdruck* und *Dokumentausdruck* sind Spezialisierungen von *Objektausdruck* gemäß der Semantik der DOM-Hierarchie und evaluieren zu einer Referenz auf ein entsprechendes JavaScript-Objekt.

- *Objektausdruck* notiert einen Pfad in einer Objekthierarchie und kann Objekte, Methoden und Variablen kombinieren; diese können gemäß der DOM-API als auch benutzerdefiniert sein. Beispiel:



JavaScript Web-APIs

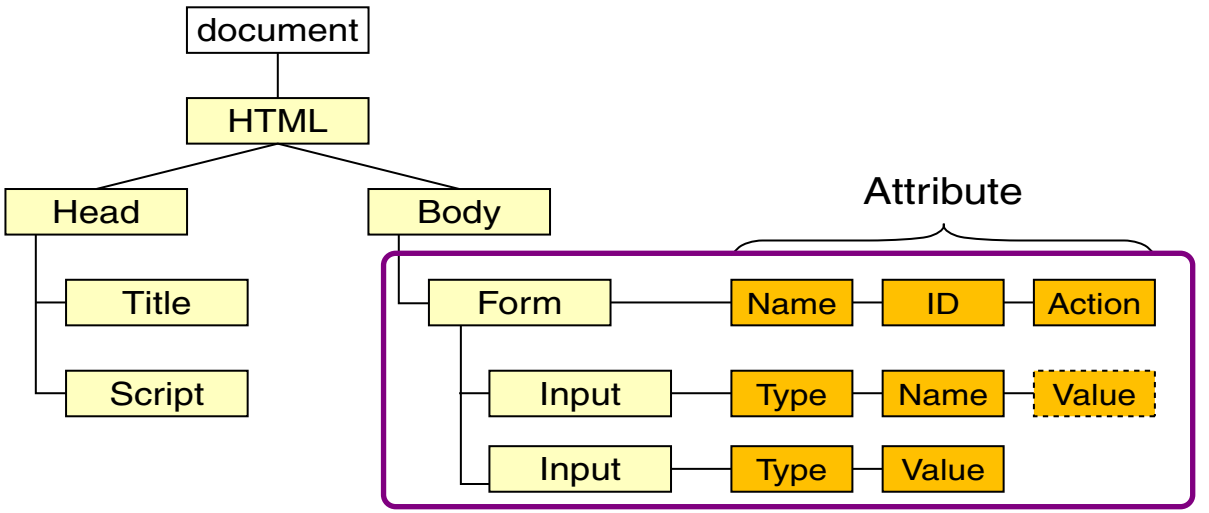
(1) DOM-API: Objektzugriff (Fortsetzung) [Einführungsbeispiel]




```
<!DOCTYPE html>
<html>
  <head>
    ...
    <script>
      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 Web-APIs

(1) DOM-API: Objektzugriff (Fortsetzung) [Einführungsbeispiel]



Zugriffsmöglichkeiten auf das erste Formelement:

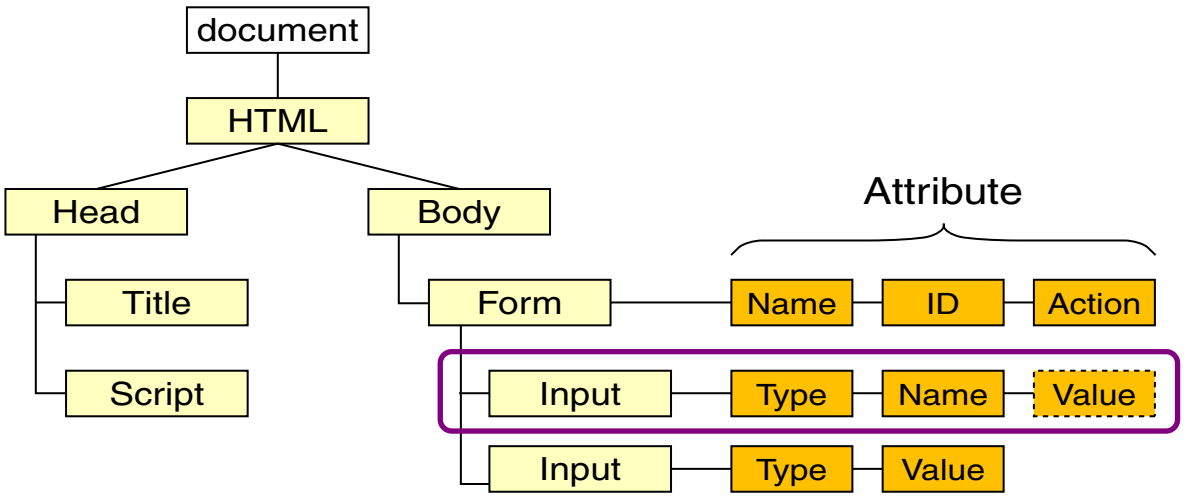
`document.quadratForm`	qualifizierender Name
`document.getElementsByTagName("form")[0]`	DOM-API
`document.getElementById("QF")`	DOM-API
`document.querySelector("body #QF")`	DOM-API

Kontrollausgaben:

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

JavaScript Web-APIs

(1) DOM-API: Objektzugriff (Fortsetzung) [Einführungsbeispiel]



Zugriffsmöglichkeiten auf das erste Eingabeelement:

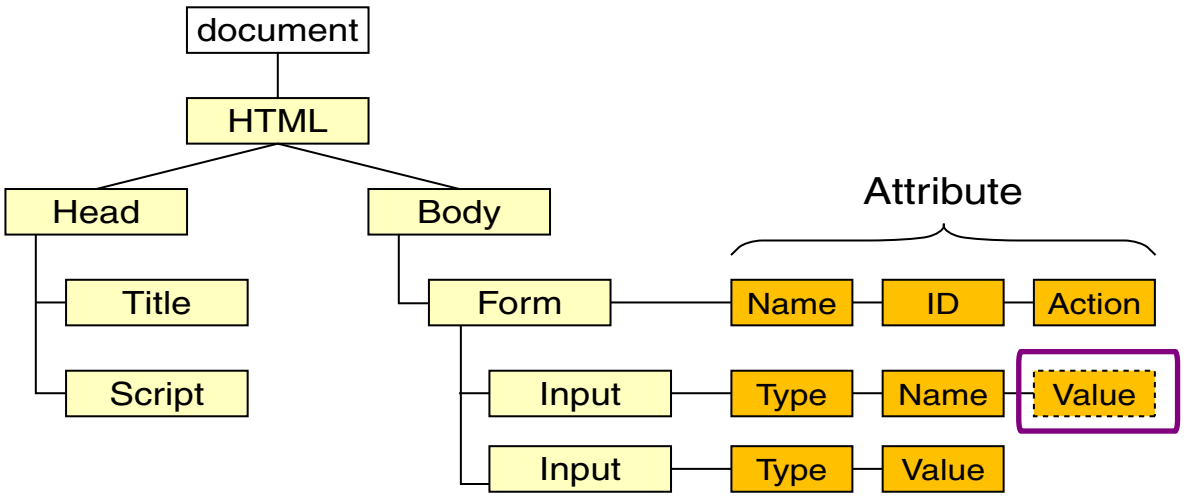
```
document.quadratForm.eingabe                          qualifizierender Name
document.getElementsByName("eingabe")[0]              DOM-API
document.getElementsByTagName("form")[0].eingabe      Kombination
document.querySelector("body #QF").eingabe            Kombination
```

Kontrollausgaben:

```
document.writeln(document.quadratForm.eingabe) ⇝ [object HTMLInputElement]
```

JavaScript Web-APIs

(1) DOM-API: Objektzugriff (Fortsetzung) [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 Web-APIs

(1) DOM-API: Objektzugriff (Fortsetzung) [Einführungsbeispiel]

```
<script>
  function quadrat() {
  let zahl = document.quadratForm.eingabe.value;
  let 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 Web-APIs

(1) DOM-API: Ereignisbehandlung

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

JavaScript Web-APIs

(1) DOM-API: Ereignisbehandlung (Fortsetzung)

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

Varianten für die Behandlung des Ereignisses „Mausklick" :

```
<!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>
      document.testForm.knopf.onclick = function(){ alert("pong!") };
      document.testForm.knopf.addEventListener("click",
                                    function(){ alert("pong!") });
    </script>
  </body>
<html>
```

JavaScript Web-APIs

(1) DOM-API: Ereignisbehandlung (Fortsetzung)

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

Varianten für die Behandlung des Ereignisses „Mausklick" :

```
<!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>
      document.testForm.knopf.onclick = function(){ alert("pong!") };
      document.testForm.knopf.addEventListener("click",
                                        function(){ alert("pong!") });
    </script>
  </body>
<html>
```

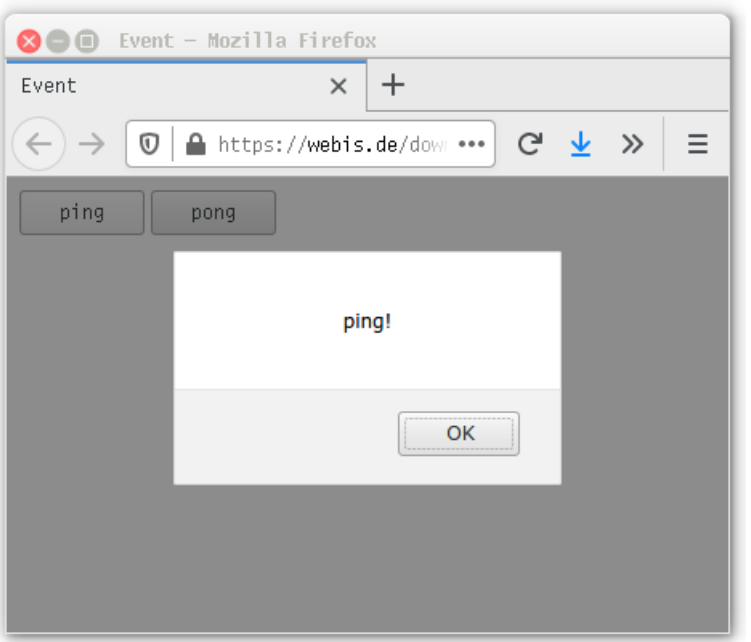
JavaScript Web-APIs

(1) DOM-API: Ereignisbehandlung (Fortsetzung)

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

Varianten für die Behandlung des Ereignisses „Mausklick" :

```
<!DOCTYPE html>
<html>
  <head><title>Event</title></head>
  <body>
    <form name="testForm">
      <input type="button" value="ping" on
      <input type="button" value="pong" na
    </form>
    <script>
      document.testForm.knopf.onclick = fu
      document.testForm.knopf.addEventList
                                        fun
    </script>
  </body>
<html>
```



[JavaScript: Aufruf]

Bemerkungen:

- Beachte, dass Funktionen über Attribute im HTML-Markup (Zeile 6) oder direkt via DOM-Interface (Zeile 10) zugewiesen werden können.
- Bei der Zuweisung via DOM-Interface kann das entsprechende Event-Attribut als Teil der DOM-Hierarchie (Zeile 10) oder die API-Funktion `addEventListener` (Zeile 11) verwendet werden. Die letzte Variante ist zu bevorzugen, da sich hiermit einem Element für das gleiche Ereignis mehrere Handler zuweisen lassen. [WHATWG] [MDN]
- Die MDN-Dokumentation listet (Stand Juli'22) über 350 JavaScript-Ereignisse. [MDN]
- Mittlerweile können viele Maus-Events ohne JavaScript mittels CSS realisiert werden. Beispiel: [Webis: Courses Map]

JavaScript Web-APIs

(1) DOM-API: Erweiterter Objektzugriff mit jQuery

[*TODO*]

JavaScript

Quellen zum Nachlernen und Nachschlagen im Web

- ECMA. *Standard ECMA-262: ECMAScript Language Specification.*
  www.ecma-international.org/publications/standards/Ecma-262
- 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