



FH Bielefeld
University of
Applied Sciences

Campus Minden



Studiengang Informatik Fachbereich Technik

Webbasierte Anwendungen SS 2015

Clientseitige Implementierungstechnologien: JavaScript

Dozentin: Grit Behrens
mailto:grit.behrens@fh-bielefeld.de

Gutes Tutorial:<http://www.w3schools.com>

Lehrinhaltsübersicht zu WBA

1. Einführung in Webbasierte Anwendungen, selbs
2. Web-Grundlagen, Historie, Struktur von Webseiten: CSS, XML, XHTML, HTML5
- 3. Clientseitige Implementierungstechnologien: JavaScript, DOM, Ajax, (Java-Applet)**
4. Serverseitige Implementierungstechnologien: JSP, Java-Servlet, Webserver Apache, Tomcat
5. WEB-Frameworks: (Struts), JSF

Clientseitige Implementierungstechnologien (I): Javascript

1. Scriptsprachen
2. Einführung in JavaScript
3. Technisches Umfeld
4. Grundlagen : Auswahl
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Clientseitige Implementierungstechnologien (I): Javascript

- 1. Scriptsprachen**
2. Einführung in JavaScript
3. Technisches Umfeld
4. Grundlagen : Auswahl
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Wozu Scriptsprachen?

- Häufig wiederkehrende Aufgaben
- Abstraktionen dieser Aufgaben in Scriptsprachen
 - Sprachmittel auf höherer Ebene (z.B. Schleifen = für alle)
 - z.T. Erreichen der Abstraktion natürlicher menschlicher Sprache
 - mächtige Datenstrukturen als Teil der Sprache (z.B. Listen)
 - umfangreiche integrierte Bibliotheken
- Ziele
 - Geringer Einarbeitungsaufwand
 - Schnelle Implementierung
 - Leicht lesbarer und verständlicher Code
 - Fehlerfreiheit
 - Einfache Wartbarkeit

Eigenschaften von Scriptsprachen I

- Automatische Speicherverwaltung
- Mächtige Datenstrukturen
 - String- und Zeichenkettenverarbeitung
 - statische und dynamische Listen
 - Tupel
 - Wörterbücher, assoziative Arrays
- z.T. Objektorientierung (Python, JavaScript, Php, Perl)
- z.T. Typsysteme und Modulkonzepte

Eigenschaften von Scriptsprachen II

- Umfangreiche Bibliotheken und spezifische Features
 - Web-Anwendungen (html-Integration, HTTP, Sessionverwaltung, XML, REST, JSON, AJAX)
 - Benutzeroberflächen (grafische Elemente u. Interaktivität)
 - Allgemein (reguläre Ausdrücke,...)
 - Kombination bekannter Bausteine (Kontrolle von Ein- und Ausgabe, Prozessverwaltung)

Beispiele für Scriptsprachen

- Kommandozeileninterpreter
 - /bin/sh Bourne Shell
- Sprachen/Tools zur String-Manipulation
 - Sed, Awk
- Vollwertige Programmiersprachen
 - Perl (aus Awk mit Focus auf String-Manipulation)
 - Python (Übernahme von allen allgemeinen Eigenschaften aus Vorgängern Sh, Sed, Awk, Perl)
 - Visual Basic, VBScript, JScript, ...
- Spezialisierung auf bestimmte Anwendungen
 - Php für serverseitige Web-Applikationen
 - JavaScript für clientseitige Web-Applikationen

Clientseitige Implementierungstechnologien (I): Javascript

1. Scriptsprachen
- 2. Einführung in JavaScript**
3. Technisches Umfeld
4. Grundlagen : Auswahl
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Einführung in JavaScript

Entstehung:

- 1995 bei Firma Netscape (damaliger Name: *LiveScript*)
- Bezeichnung **JavaScript** seit Impl. in Netscape Navigator 2

Entwicklungsziele:

- Erweiterung statischer Webseiten mit dynamischen Inhalten
- Attraktivere Gestaltungsmöglichkeiten
- Verbesserte Funktionalität

Beispielhafte Anwendungsfälle:

- Überprüfen von Dateneingaben auf Korrektheit beim Client
- Kleine eigenständige Clientanwendungen (Taschenrechner, Währungsrechner, ...)
- Clientseitige Scripte für dynamisch eingeblendete Schaltflächen, Grafiken
- In Verbindung mit HTML5 – clientseitige Mobile Applikationen

Einführung in JavaScript

Eigenschaften:

- Vollwertige Programmiersprache
- Objektorientierte Basis
- Interpretersprache
- Einfach zu erlernen
- Überwiegend clientseitige Nutzung im Browser
- Serverseitige Nutzung möglich mit *JScript* ; aber sehr selten (weiter verbreitet: PHP, Perl, ASP - Active Server Pages, JSP – Java Server Pages)
- plattformübergreifend (versch. Browser, versch. Betriebssysteme, versch. Architekturen, auch mobile Endgeräte)
- Standardisierungsprobleme bei Browsern: z.T. laufen JavaScript-Programme nicht oder nicht richtig
- Greift auf HTML- Code zu und erzeugt HTML-Elemente
- Wird in HTML-Code eingebunden über Tag `<script>... </script>`:

```
<script type = "text/JavaScript">  
    <!--JavaScript – Anweisungen-->  
</script>
```

Clientseitige Implementierungstechnologien (I): Javascript

1. Scriptsprachen
2. Einführung in JavaScript
- 3. Technisches Umfeld**
4. Grundlagen : Auswahl
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Technisches Umfeld zu JavaScript

1. Sicherheit
2. Im Browser aktivieren
3. Debuggen von Funktionen

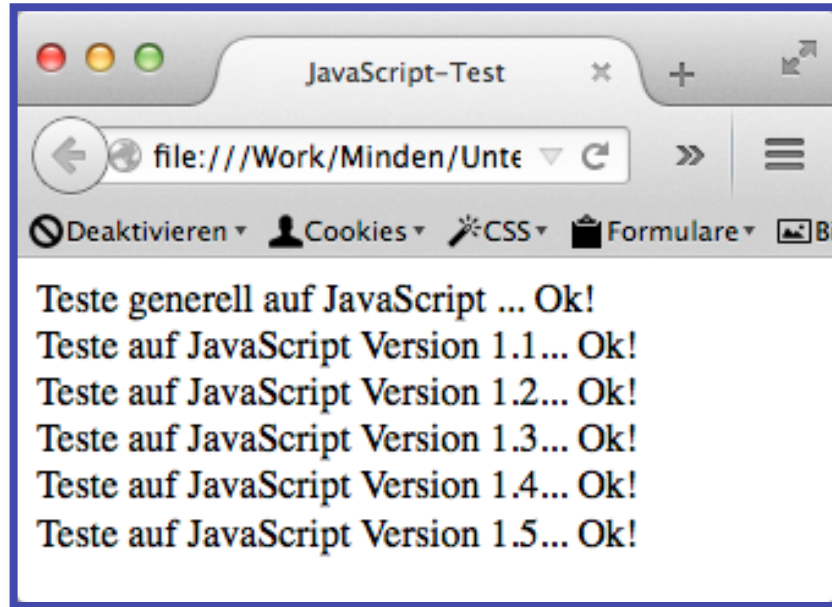
JavaScript und Sicherheit

- Sicherheitslücken in Browserimplementierungen können durch JavaScript-Programme ausgenutzt werden z.B.:
 - unbemerktes Versenden von Emails
 - Auslesen des Browserverlaufs
 - Live-Verfolgungen von Internetsitzungen
 - Erraten von EBAY-Passwörtern
- Anwender deaktivieren daher manchmal das „Ausführen von JavaScript-Code“ im Browser
- JavaScript-Anwendungen laufen im Browser: Sandbox (abgeriegelte Umgebung ohne Zugriff auf Dateien, Benutzerdaten, BS,...)

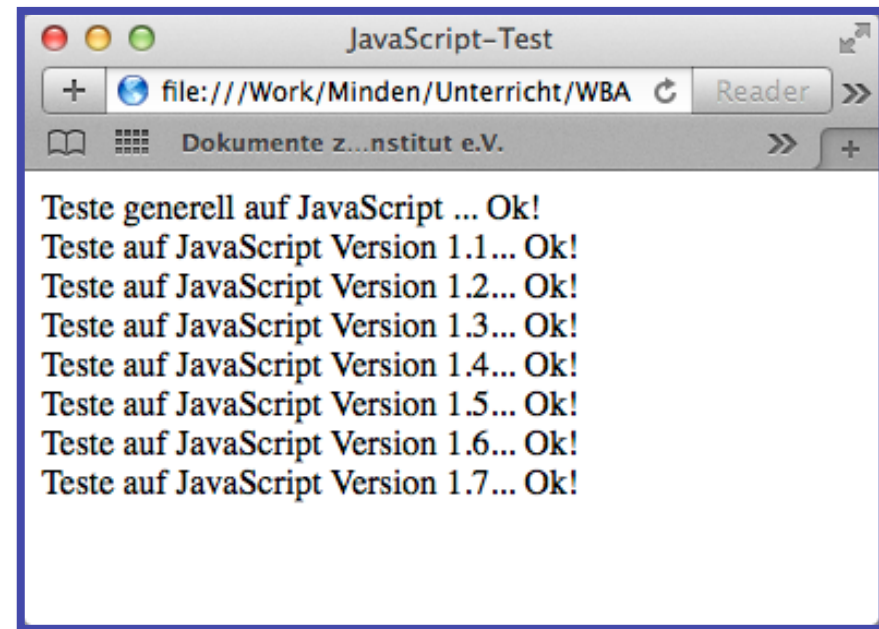
Versionen von JavaScript in verschiedenen Browsern (2015)

Mozilla Firefox 36.0.1 (MAC)

JavaScript Version 1.5



Safari 7.1.4 (Apple) JavaScript Version 1.7

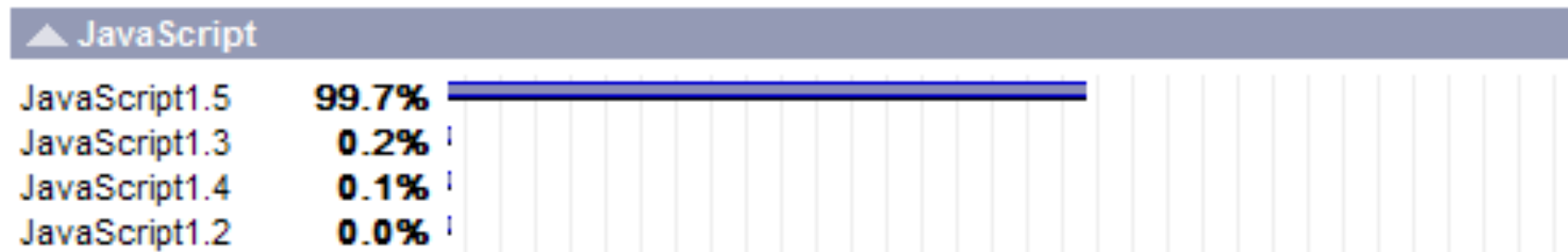


Versionen von JavaScript

```
<html>
<head>
  <title>JavaScript-Test</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      document.write("Teste generell auf
      JavaScript ... Ok!<br>")
    //-->
  </script>
  <noscript>
    Ihr Browser versteht kein JavaScript. Es
    kann nicht ausgeführt werden.
  </noscript>
  <script language="JavaScript1.1">
    <!--
      document.write("Teste auf JavaScript
      Version 1.1... Ok!<br>")
    //-->
  </script>
```

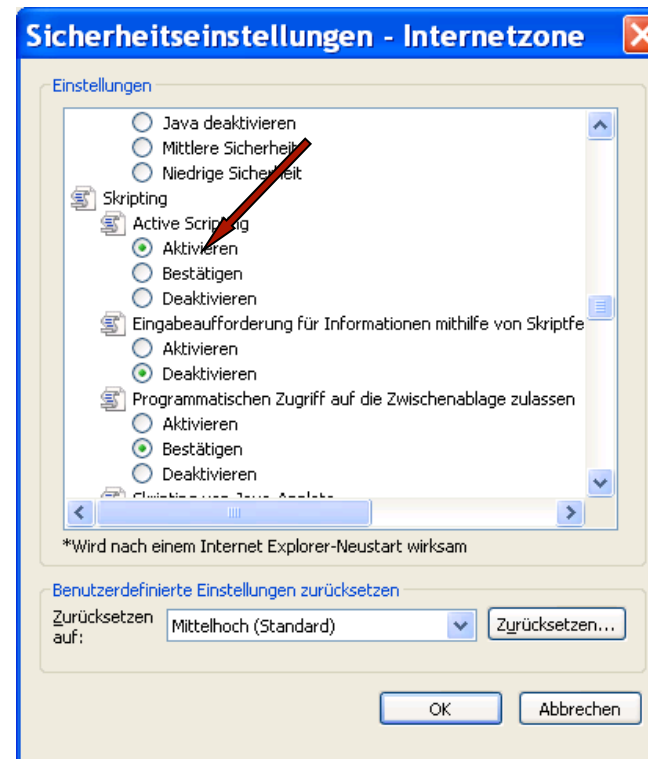
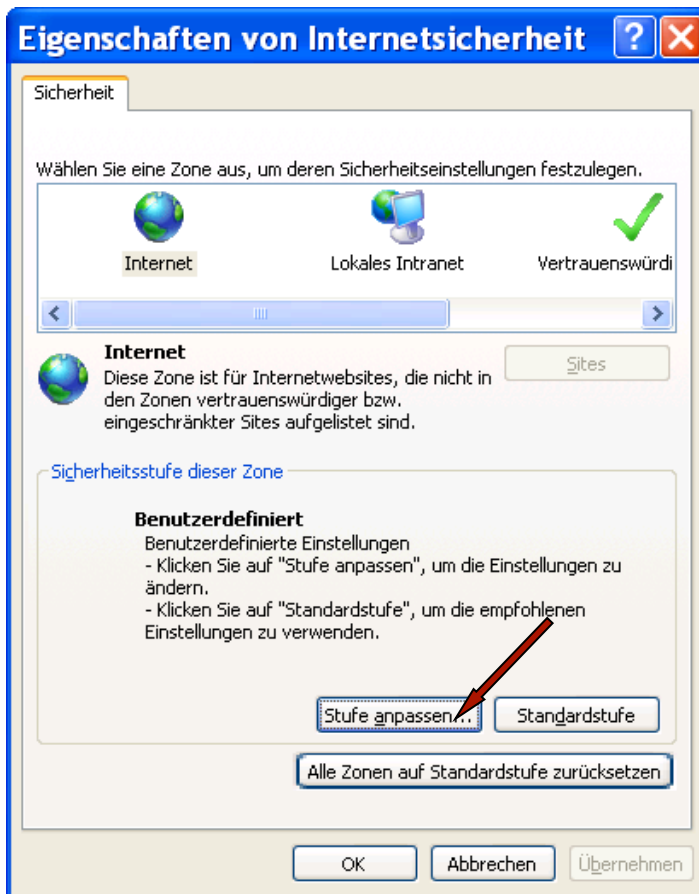
```
    <script language="JavaScript1.2">
      <!--
        document.write("Teste auf JavaScript
        Version 1.2... Ok!<br>")
      //-->
    </script>
    <script language="JavaScript1.3">
      <!--
        document.write("Teste auf JavaScript
        Version 1.3... Ok!<br>")
      //-->
    </script>
    <script language="JavaScript1.4"><!--
      document.write("Teste auf JavaScript
      Version 1.4... Ok!<br>")
    //--></script>
    <script language="JavaScript1.5">
      <!--
        document.write("Teste auf JavaScript
        Version 1.5... Ok!<br>")
      //-->
    </script>
  </body>
</html>
```

Versionen von JavaScript : Statistik über alle Browser im deutschsprachigen Raum



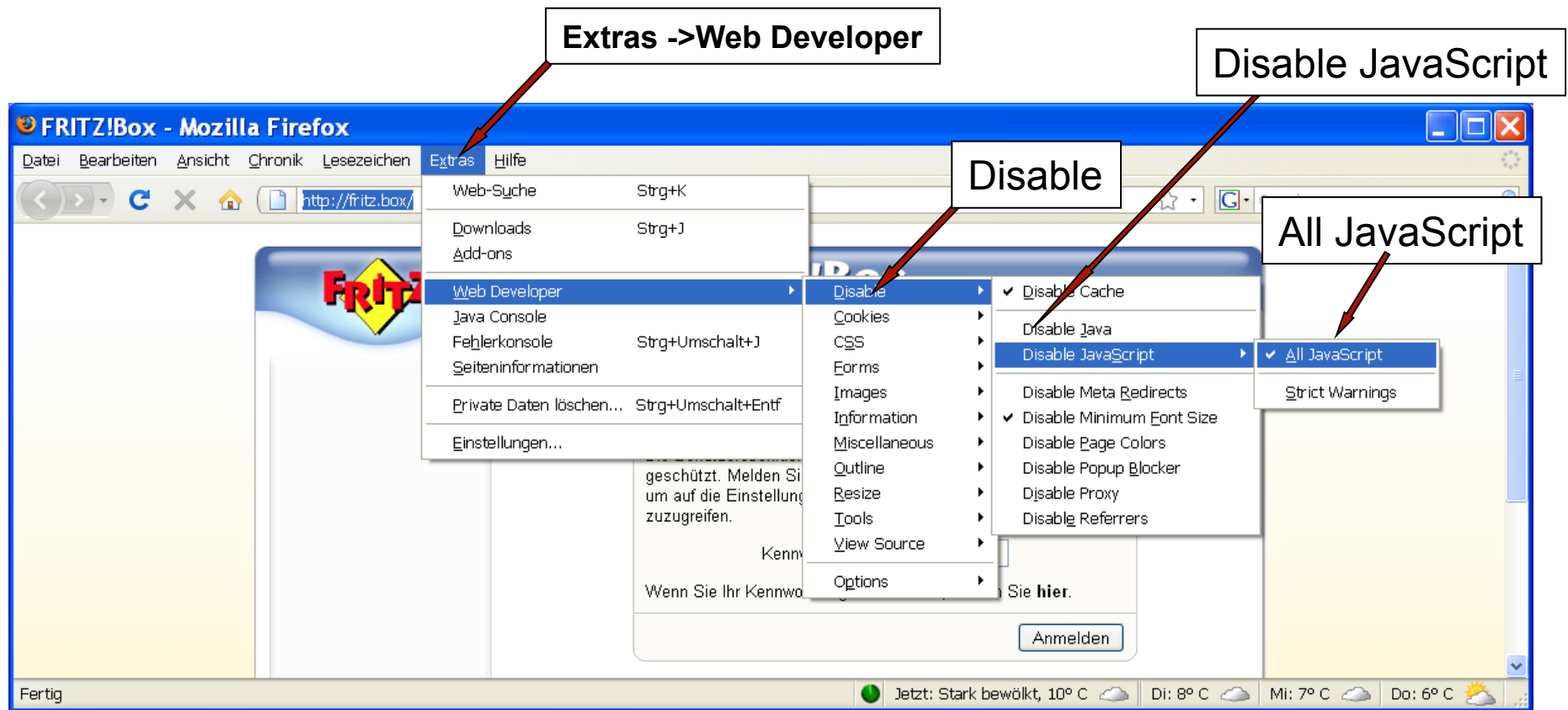
<http://www.webhits.de>

JavaScript im Browser (MSIE) aktivieren



Aktivierung JavaScript im MSIE

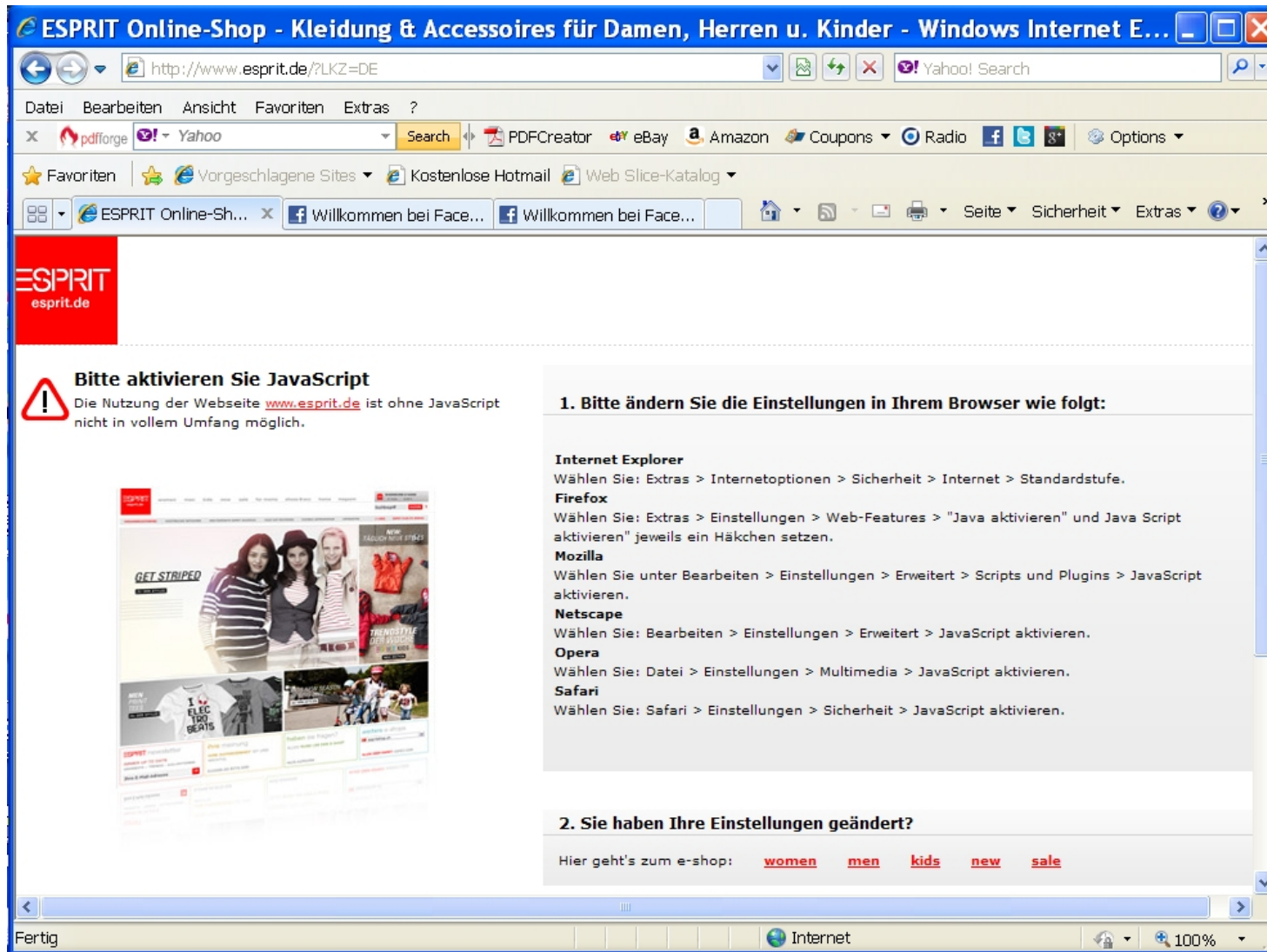
JavaScript ist im Browser (MF) standartmäßig aktiviert -> deaktivieren:



Deaktivierung/Aktivierung JavaScript im Mozilla Firefox 3.0.3

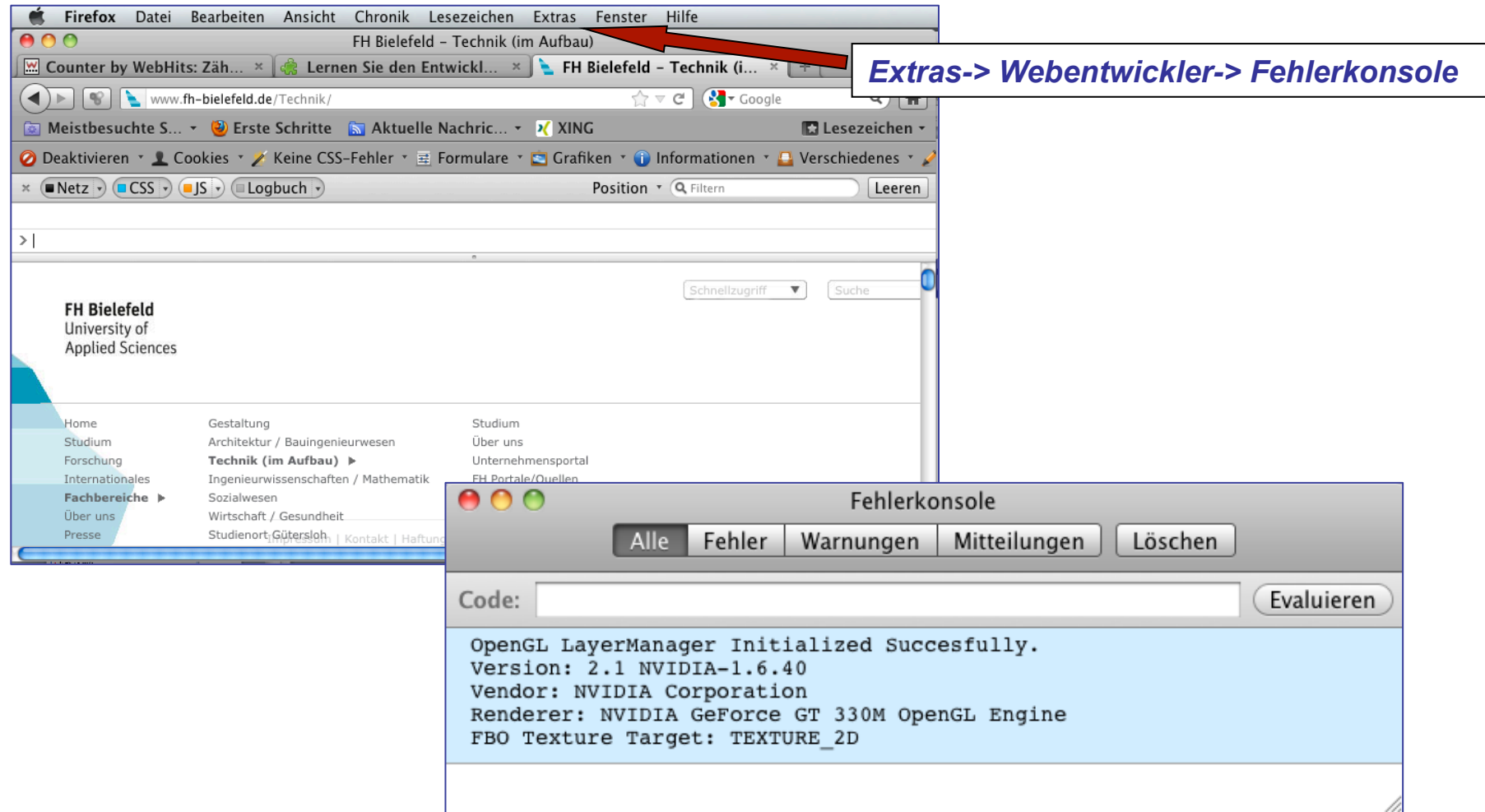
- gleiche Optionen wie bei MF 3.03 auch bei aktuellster Version des Mozilla Firefox 11.0
- vorher WEB-Developer Plugin installieren über **->Extras, ->Webentwickler -> weitere Tools**

Beispiel eines Webshops mit Überprüfung von JavaScript

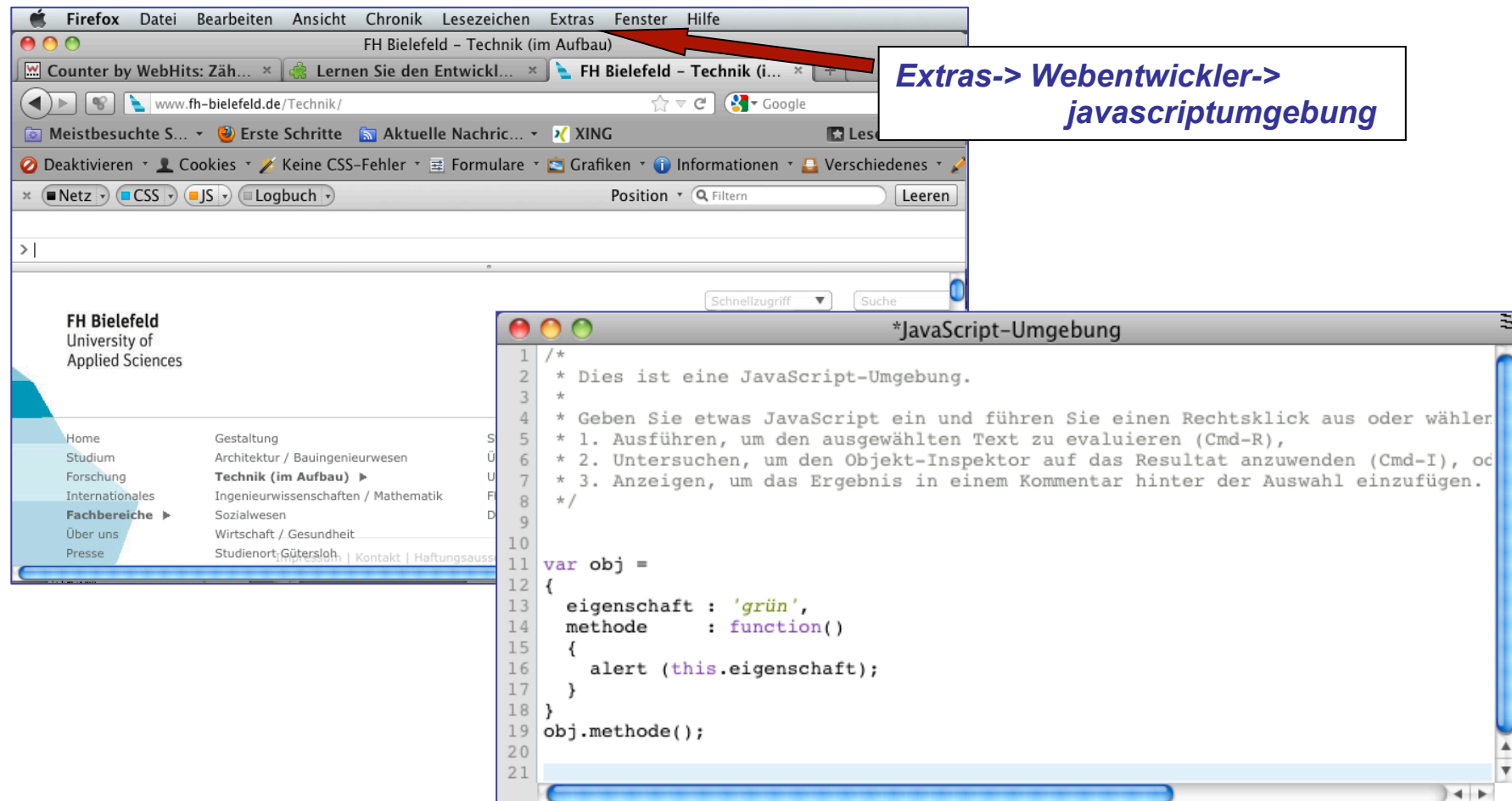


Quelle:
<http://www.esprit.de>

Debuggen von JavaScript - Funktionen (MF)



Entwickeln und Ausführen von JavaScript - Funktionen (MF)



Clientseitige Implementierungstechnologien (I): Javascript

1. Scriptsprachen
2. Einführung in JavaScript
3. Technisches Umfeld
- 4. Grundlagen : Auswahl**
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Ausgewählte Grundlagen von JavaScript

- Variablennamen
- Datentypen
- Operatoren
- Notationen
- Programmsteuerung
- Funktionen
- Objekte

komplettes Tutorial unter:

<http://de.selfhtml.org/javascript/sprache/index.htm> *(leicht veraltet, auf deutsch)*

<http://wiki.selfhtml.org/wiki/Startseite>

<http://www.w3schools.com> (sehr aktuell und komfortabel, auf englisch)

Variablennamen

Regeln für Variablennamen in JavaScript:

- Beginn mit Buchstabe oder Unterstrich (Achtung : case sensitive)
- Rest kann Buchstabe, Ziffer oder das Sonderzeichen „_“ (underscore) sein (keine Leerzeichen, keine anderen Sonderzeichen)
- Maximallänge 32 Zeichen
- es gibt reservierte Wörter wie : var, case, for, ...

Zum Beispiel:

_2_Test_Wert	<i>gültig</i>	
2_Test_Wert	<i>ungültig</i>	<i>Zahl</i>
Email_Adresse	<i>gültig</i>	
@_Adresse	<i>ungültig</i>	<i>Sonderzeichen</i>
Langer_Variablenname	<i>gültig</i>	
Das_ist_noch_laengerer_Variablenname	<i>ungültig</i>	<i>zu lang</i>
Test 1	<i>ungültig</i>	<i>Leerzeichen</i>
Test_1	<i>gültig</i>	

Wertzuweisungen an Variablen

Variablen können durch `var` definiert werden und einen Anfangswert zugewiesen bekommen.

Der Anfangswert legt den Datentyp fest.

Beispiel:

```
var beispiel = 1;  
var auch_beiispiel = 8;  
var zusammen = beispiel+auch_beiispiel;
```

*JavaScript lässt es zu, dass der Programmierer **eine Variable überhaupt nicht definiert**. Durch **dynamische Typisierung** erhält die Variable dann während der Programmausführung vom Interpreter einen Datentyp.*

Achtung: Unterschiedliche Realisierung in Browsern möglich!

Der `typeof` – Operator liefert den aktuellen Typ: string, number oder boolean.

```
var Zahl = 5;  
Typ = typeof Zahl; //number
```

```
var Zeichen = "Hallo";  
Typ = typeof Zeichen; //string
```

```
var istWahr = true;  
Typ = typeof istWahr;  
//boolean
```

Datentypen

Ganzzahlen

- besitzen keine Nachkommastellen
 - Dezimaldarstellung z.B. : 42
 - Oktaldarstellung mit vorangestellter Null „0“ z.B. : 052
 - Hexadezimaldarstellung mit vorangestelltem „0x“ z.B. 0x2A

Gleitkommazahlen

- es wird ein Punkt und kein Komma verwendet
 - z.B. 1E3 (1000.0) , 1.0E3 (1000.0) , 10.0e2 (1000.0), 1004.0E-1 (100.4)

Strings

- Kette von Zeichen, die in Anführungszeichen eingeschlossen ist “ “ oder ‘ ‘
 - z.B. “Zeichenkette“, “Er sagte: ‘Das kann doch nicht wahr sein! ‘, und ging.

Boolesche Werte

- **true** und **false** (meist als Ergebnisse von Vergleichen)

Datentypen: Beispiel

```
<html>
<head>
<Title>Datentypen</Title>
</head>
<body>
<script language="JavaScript">
<!--
var a = 4.2e1
document.write(a,"<br>");

var nummer = 1;
var zeichen = '1';

var ergebnis= nummer+nummer;
document.write(ergebnis,"<br>");

var ergebnis = zeichen+zeichen;
document.write(ergebnis);

// -->
</script>
</body>
</html>
```



Welche Ausgabe generiert dieses Beispielprogramm?

Operatoren

Operatoren	Beispiele	Datentyp
Vergleichsoperatoren	==, !=, <>, <, >, <=, >= Ergebnis: boolescher Wert	Zahlen, Strings
Berechnungsoperatoren	+, -, *, /, % (Modulo), ++ (Inkrement), --(Dekrement)	Zahlen
Konkatenationsoperator	'1 '+ '1 '= '11 ';	Strings
Logische Operatoren	&& - UND, - ODER, ! . NICHT	Boolesche Werte
Bit-Operatoren	& - (1010&0110)= 0010; - ODER, ~ - NICHT, << Linksverschiebung	Zahlen, boolesche Werte
Zuweisungsoperatoren	a=a+5; a+=5; //beide gleich	alle

Bis hierher

Notationen

Alle Anweisungen in JavaScript müssen mit einem Semikolon “;” enden.

Kommentare:

//	bezeichnet einen einzeiligen Kommentar
/* */	bezeichnet einen mehrzeiligen Kommentar

Programmsteuerung

Bedingte Ausführungen und Schleifen dienen dazu, die lineare Abarbeitung eines Programmes aufzubrechen.

Anweisungsblöcke werden in geschweifte Klammern eingeschlossen.

Beispiel:

```
function Addiere(Zahl1,Zahl2)
{
    var Ergebnis = Zahl1 + Zahl2;
    return Ergebnis;
}
```

Programmsteuerung

Wenn-Dann-Anweisungen werden durch den **if-Befehl** abgebildet. Bedingungen können auch ineinander verschachtelt sein.

```
if (Bedingung)
{
    Anweisung 1;
    Anweisung 2;
    ...
}
```

Die alternative Bedingung wird durch den **else-Befehl** abgebildet.

```
if (Bedingung)
{
    Anweisung A1;
    Anweisung A2;
}
else
{
    Anweisung B1;
    Anweisung B2;
}
```

Programmsteuerung

Mehrstufige Bedingungen werden durch den `else-if`-Befehl dargestellt.

```
if (Bedingung 1)
{
    Anweisungsblock A;
}
else if (Bedingung 2)
{
    Anweisungsblock B;
}
else
{
    Anweisungsblock C;
}
```

Einfache Entweder-Oder-Abfragen für zwei einzelne Anweisungen können mit dem Entweder-Oder-Operator “ ? : ” realisiert werden.

```
(Bedingung) ? Erfüllt_Anweisung : NichtErfüllt_Anweisung;
```

Programmsteuerung

Die Mehrseitige Auswahl wird mit der `switch-case-Anweisung` geschrieben.

```
switch (Variable)
{
  case "Wert 1" : Anweisungsblock A;
  break;
  case "Wert 2" : Anweisungsblock B;
  break;
  ...
  default : Anweisungsblock X;
}
```

Schleifen

while Schleife wird nur ausgeführt, wenn eine Bedingung erfüllt ist,

do-while Schleife wird erst einmal ausgeführt, bevor Bedingung am Ende eines jeden Durchlaufs getestet wird oder

for gezählte Schleifen mit Anfangswert, Endwert und Schrittweite des Zählers

break

continue

Funktionen

Funktionen können den Code übersichtlicher gestalten. Sie werden oft von Ereignissen ausgelöst und bringen so Dynamik auf die Web-Seite.

Syntax für Funktionsdefinition:

Eine Funktion ist ein Anweisungsblock, der mit dem Schlüsselwort **function**, dem Namen der Funktion und **runden Klammern** eingeleitet wird.

```
function Funktionsname (Parameter 1, Parameter 2, ...)  
    {  
    ... // JavaScript - Anweisungen  
    }
```

Syntax für Aufruf der Funktion in einem Anweisungsblock :

Anweisung 1;

Anweisung 2;

X = Funktionsname(Parameter 1, Parameter 2, ...);

Anweisung 3;

Funktionen

Welche Ausgabe wird im Browserfenster erzeugt?

Beispiel Funktionen aufrufen:

```
<html>
<head>
<Title>Funktionen</Title>
</head>
<body>
<script language="JavaScript">
<!--
function Produkt (x, y) //Funktion definiert 2 Parameter
{
    var Ergebnis = x * y;
    return Ergebnis;
}

var MeinProdukt = Produkt (33, 11); //Aufruf der Funktion mit 2 Parametern
document.write("Das Produkt aus 33 und 11 ergibt: ",MeinProdukt);
// -->
</script>
</body>
</html>
```



Objekte I

In JavaScript werden Objekte über eine Konstruktorfunktion erzeugt, nicht mit Klassen wie in Java.

Ein einzelnes neues Objekt wird mit **new** erzeugt:

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

```
obj.eigenschaft = 'grün'; // Objekt obj erhält Eigenschaft grün
```

```
obj.methode = function() //Objekt erhält Methode methode
{
    alert('Hallo Grün');
}
```

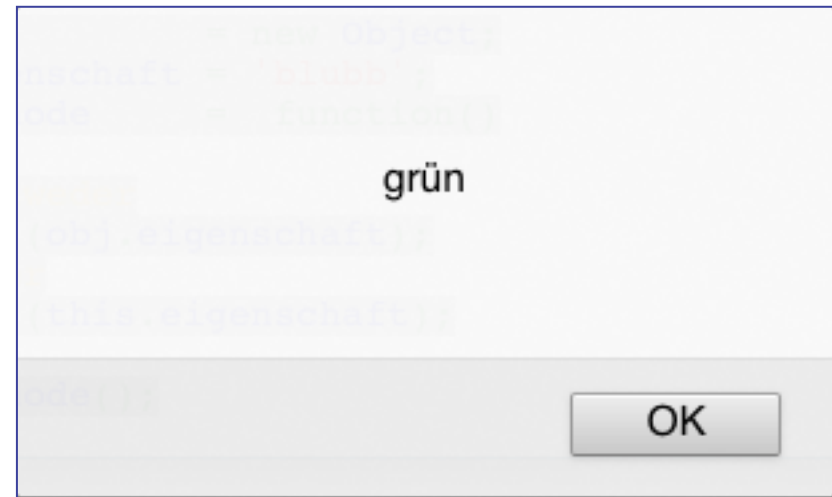
Der Zugriff auf Eigenschaften und Methoden erfolgt über den Namen der Referenzvariablen des Objekts:

```
alert (obj.eigenschaft);
obj.methode();
```

Objekte II

Der Zugriff auf Eigenschaften in Methoden erfolgt über den Namen der Referenzvariablen des Objekts oder über `this`:

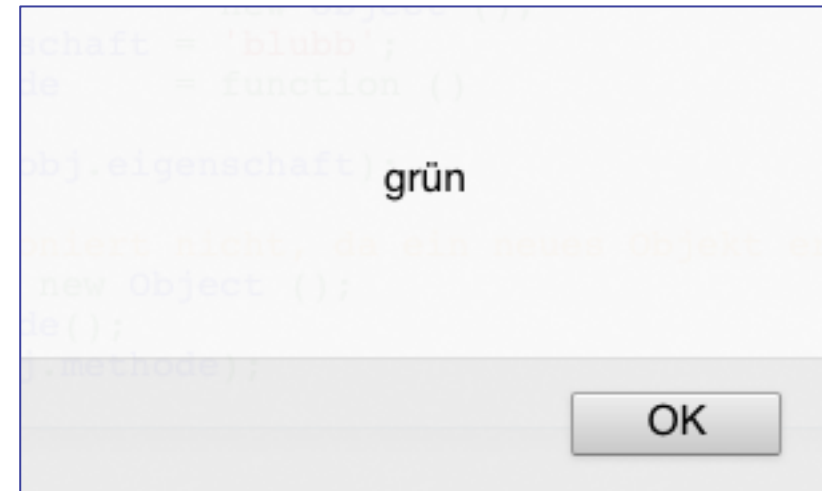
```
var obj          = new Object;  
obj.eigenschaft = 'grün';  
obj.methode      = function()  
{  
  // entweder  
  alert (obj.eigenschaft);  
  // oder  
  alert (this.eigenschaft);  
}  
obj.methode();
```



Objekte III

Andere Objekte haben Zugriff auf die Eigenschaften und Methoden über den Namen der Referenzvariablen des Objekts:

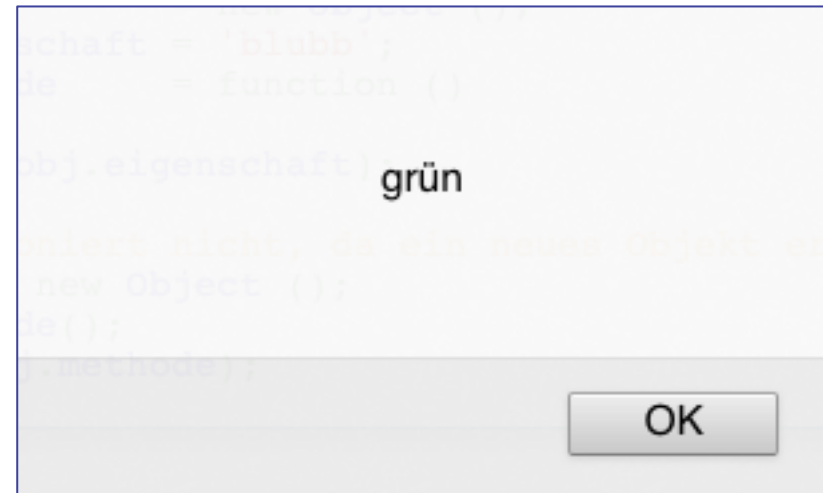
```
var obj_1      = new Object ();
obj_1.eigenschaft = 'grün';
obj_1.methode   = function ()
{
    alert (obj_1.eigenschaft);
};
var obj_2      = new Object();
obj_2.methode = function ()
{
    alert (obj_1.eigenschaft);
}
obj_2.methode();
```



OOP: Objektdекlaration mit Doppelpunkt

In anderer Schreibweise wird das Objekt von geschweiften Klammern umgeben, Eigenschaften und Methoden werden durch Namen und Doppelpunkt deklariert:

```
var obj =  
{  
  eigenschaft : 'grün',  
  methode      : function()  
  {  
    alert (this.eigenschaft);  
  }  
}  
obj.methode();
```



Clientseitige Implementierungstechnologien (I): Javascript

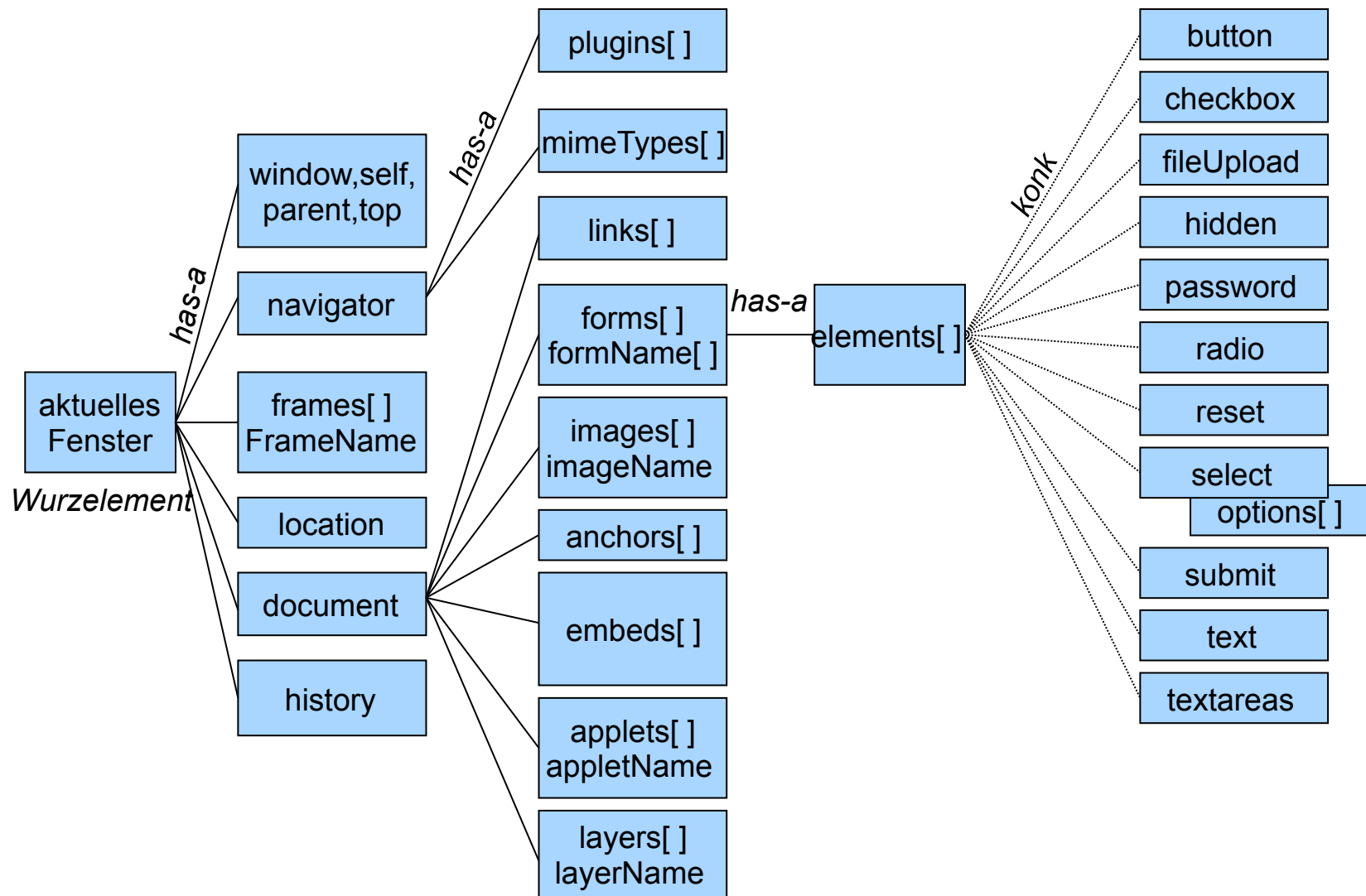
1. Scriptsprachen
2. Einführung in JavaScript
3. Technisches Umfeld
4. Grundlagen : Auswahl
- 5. Objektmodell**
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Vordefinierte Objekte in JavaScript

JavaScript – nimmt die Browserumgebung über Objekte wahr und tritt über diese Objekte in Wechselwirkung mit seiner Umgebung

- Objekte beschreiben **Eigenschaften und Funktionalitäten** von z.B.:
 - einem Taster
 - der URL-Eingabezeile eines Browsers (Location)
 - von Bildern
- Objekte sind **hierarchisch** angeordnet
 - *window* – aktuelles Browserfenster ist Wurzelobjekt
 - *window* **enthält (has-a-Beziehung)** :
 - *navigator* (Browsereigenschaften)
 - *history* (Aufzeichnungspfad)
 - *document* (Repräsentation des Dokuments)
 - ...
- **Konkretisierung (Ausprägungsbeziehung)** zwischen *elements*:
 - *button, checkbox, password, radio, reset, ...*

Das Objektmodell von JavaScript



Vordefinierte Objekte

1. Navigator

2. Window

3. Document

4. History

5. Location

Objekt Navigator

Das Objekt ***navigator*** erlaubt es, die Eigenschaften des Browsers auszulesen.

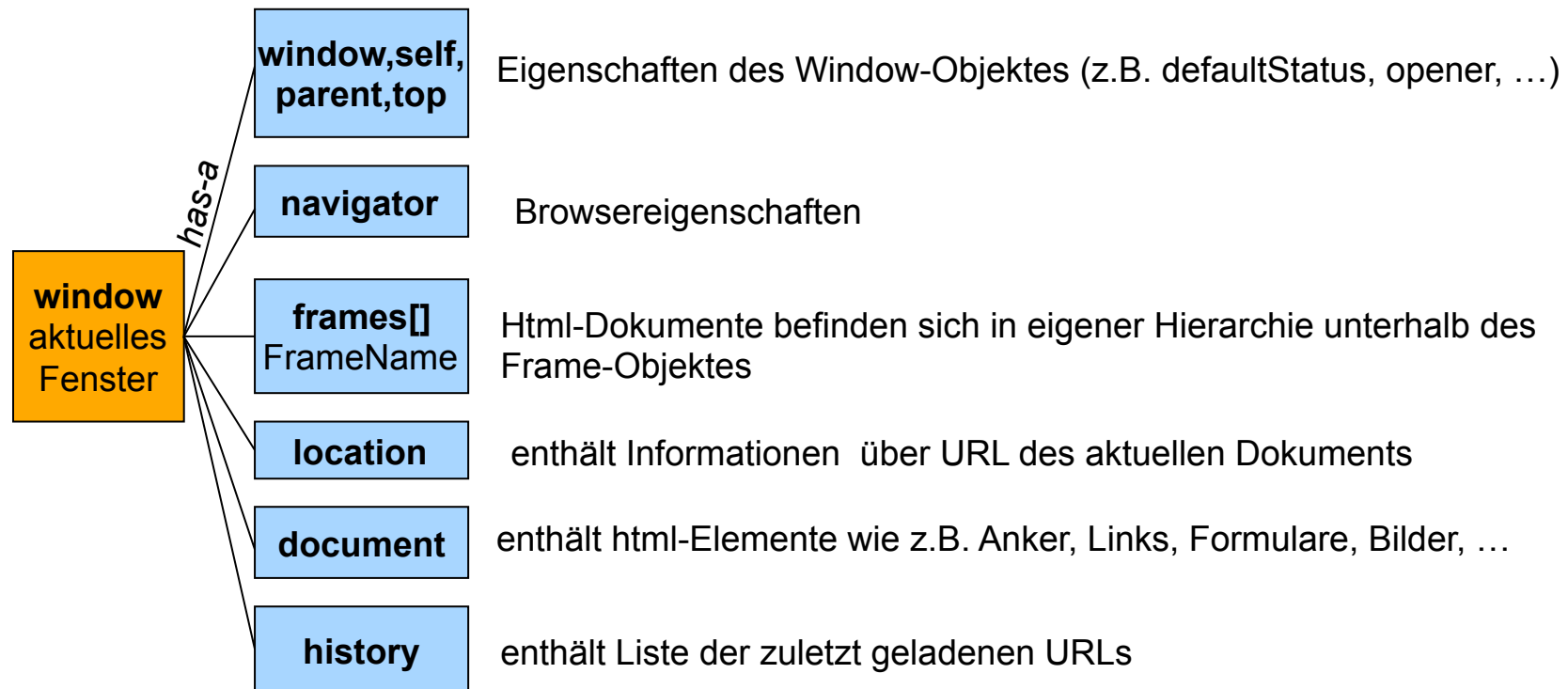
<i>appCodeName</i>	Eigenschaft liefert Arbeitsname des Browsers beim Hersteller
<i>appName</i>	Eigenschaft liefert Name des Browsers beim Nutzer
<i>appVersion</i>	Eigenschaft liefert Version des Browsers, Länderkennz. u. Plattform
<i>userAgent</i>	vollständige standardisierte Browserbezeichnung
<i>platform</i>	verwendete Computerplattform
<i>language</i>	Spracheinstellung des Clientcomputers (nur Netscape)
<i>plugins</i>	liefert Feld mit allen installierten Plugins (nur Netscape)
<i>mimeTypes</i>	liefert Feld mit allen MIME-Typen, die v. Browser akzeptiert werden (nur Netscape)
<i>javaEnabled()</i>	liefert true, wenn Java-Unterstützung vorhanden, sonst false

Vordefinierte Objekte

1. Navigator
- 2. Window**
3. Document
4. History
5. Location

Objekt Window

Das Objekt **window** steht in der Objekthierarchie an oberster Ebene. Seine Eigenschaften sind Objekte, die das Fenster und das Dokument im Fenster beschreiben und festlegen.



Ausgewählte Methoden und Eigenschaften des Window - Objekts

Methoden:

<i>alert()</i>	Meldungsfenster wird angezeigt
<i>blur()</i>	Browserfenster wird deaktiviert und in den Hintergrund verschoben
<i>close()</i>	Schließen des Browserfensters
<i>confirm()</i>	Anzeigen eines Bestätigungsfensters
<i>focus()</i>	Aktivieren des Browserfensters und Verschieben in den Vordergrund
<i>open()</i>	Öffnen eines neuen Browserfensters: <i>open("URL", "Fenstername", "Optionen")</i>
<i>prompt()</i>	Anzeigen eines Texteingabefensters

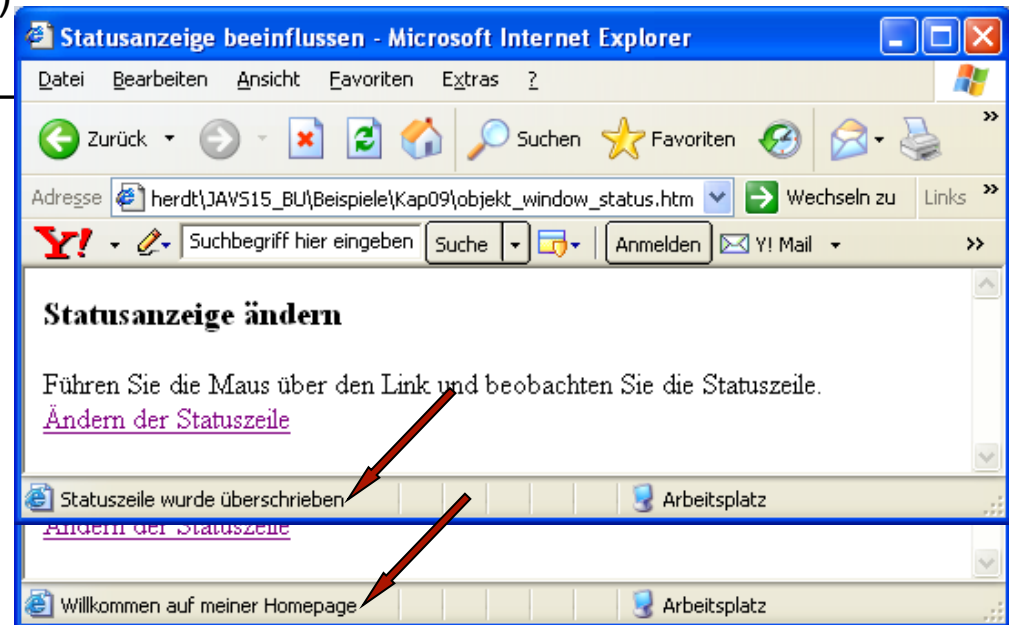
Eigenschaften:

<i>defaultStatus</i>	Festlegen von Standarttext für Statuszeile des Browsers
<i>opener</i>	enthält Referenz auf Fenster, welches das aktuelle Fenster geöffnet hat
<i>self</i>	enthält Referenz auf das aktuelle Objekt
<i>status</i>	Ändern des Textes in der Statuszeile

Statusanzeige mit *window.status* und *window.defaultStatus*

- Beispiel: individuelles Gestalten der Statusanzeige bei Überfahren eines Links mit der Maus (funktioniert nur richtig gut bei MSIE und Opera, schlechter mit Netscape und gar nicht mit Mozilla Firefox 1.7.5)

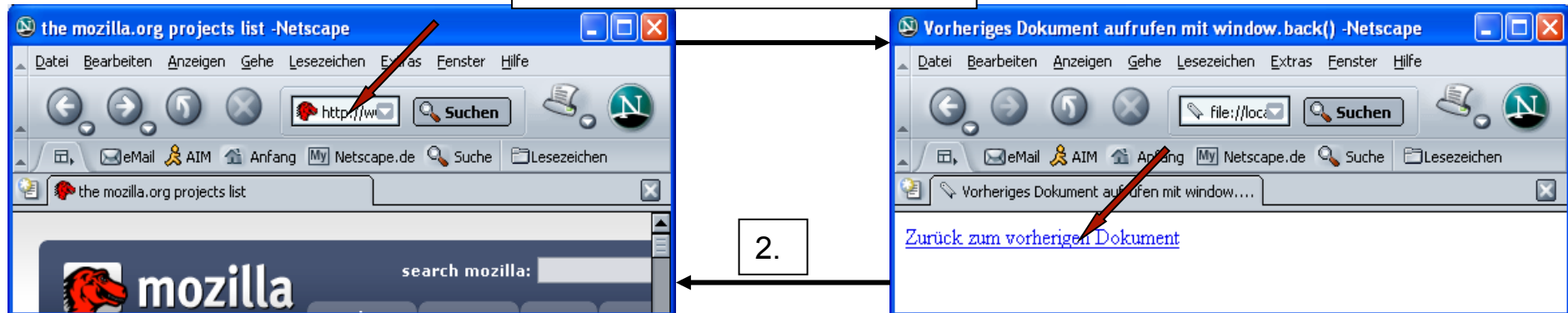
```
<html>
<head>
  <title>Statusanzeige beeinflussen</title>
  <script type="text/javascript">
    <!--
      window.defaultStatus="Willkommen
        auf meiner Homepage";
    //-->
  </script>
</head>
<body>
  <h3>Statusanzeige &uuml;ndern</h3>
  F&uuml;hren Sie die Maus über den Link und beobachten Sie die Statuszeile.<br>
  <a href="#" onmouseover="window.status='Statuszeile wurde &uuml;berschrieben'; return true;">
    &Auml;ndern der Statuszeile</a>
</body>
</html>
```



Vorheriges Dokument aufrufen mit *window.back()*

- Beispiel: Aufruf der Methode `back()` des Window – Objekts.
(funktioniert nur richtig mit Opera, Netscape, Mozilla Firefox , nicht mit MSIE)

1. Aufruf:
javascript-window_back.html



Beispiel: *javascript-window_back.html*

```
<html>
<head>
<title> Vorheriges Dokument aufrufen mit window.back()</title>
</head>
<body>
<a href="javascript: window.back()">Zurück zum vorherigen Dokument</a>
</body>
</html>
```

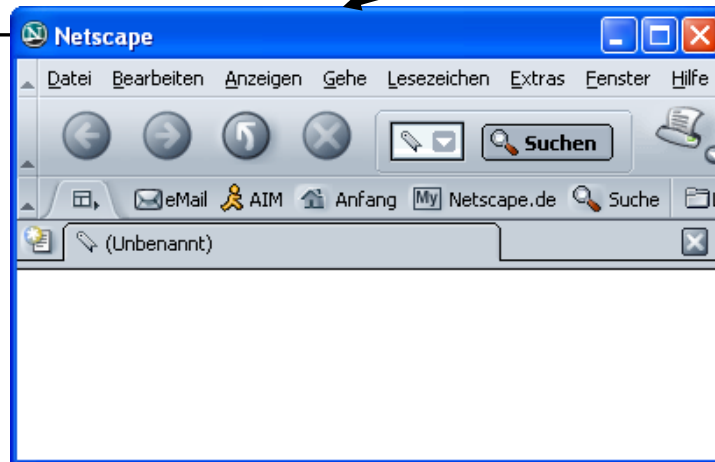
Fenster öffnen mit *window.open()*

```
window.open("URL", "Fenstername", [Optionen])
```

- angegebene **URL** wird in einem Fenster mit dem angegebenen **Fenstername** geöffnet
- bei **leerer Zeichenkette für URL** wird ein leeres Fenster geöffnet
- über die **eindeutigen Fenstername** werden die Inhalte der geöffneten Fenster angesprochen
- Rückgabewert ist die Referenz auf das window - Objekt des erstellten Fensters

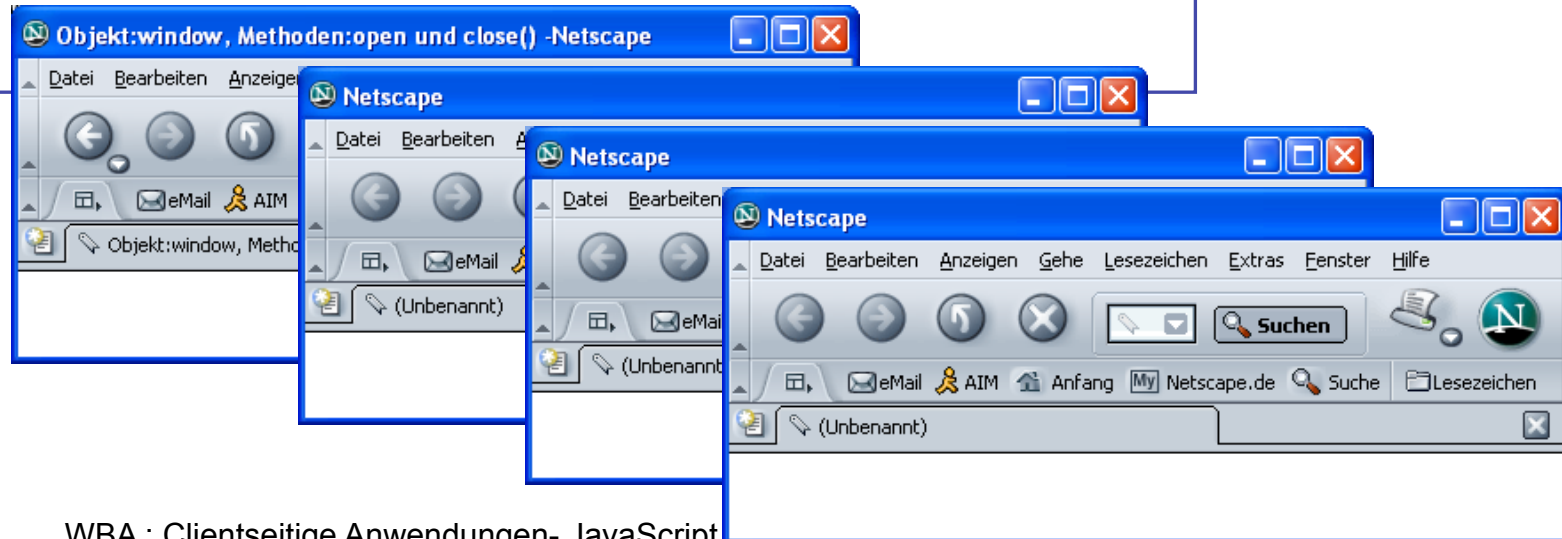
Fenster öffnen mit *window.open()*

```
<html>
<head>
  <title>Objekt:window, Methode:open()</title>
</head>
<body>
  <h3>Das Hauptfenster</h3>
  <script type="text/javascript">
    <!--
      Fenster1 = window.open("frames.html", "FensterEins");
      Fenster2 = window.open("about:blank", "FensterZwei");
    </script>
  </body>
</html>
```



Fenster schließen mit *window.close()*

```
<html>
<head>
  <title>Objekt:window, Methoden:open und close()</title>
</head>
<body>
  <script type="text/javascript">
    <!--
    Fenster1 = window.open("", "Neues_Fenster1");
    Fenster2 = window.open("", "Neues_Fenster2");
    Fenster3 = window.open("", "Neues_Fenster3");
    Fenster1.setTimeout("close()", 2000); # Zeitverzögerung 2000 ms
    Fenster2.setTimeout("close()", 5000); # Zeitverzögerung 5000 ms
    Fenster3.close();
    -->
  </script>
</body>
</html>
```



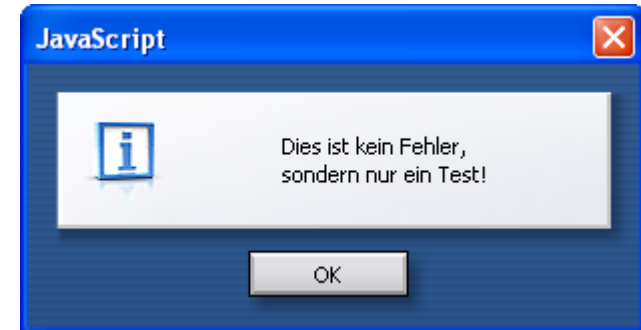
Interaktionen über Meldungsfenster mit *window.alert()*

***window.alert* ("Meldung")**

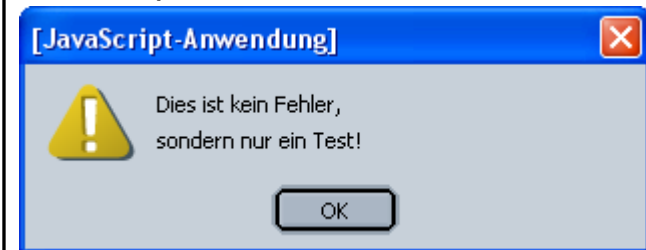
Meldungsfenster zeigen eine Information an, die der Benutzer mit einem Klick in die Schaltfläche **OK** bestätigen muß.

```
<html>
<head>
  <title>Interaktion - Dialogfenster</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      alert("Dies ist kein Fehler, \nsondern nur ein Test!");
    //-->
  </script>
</body>
</html>
```

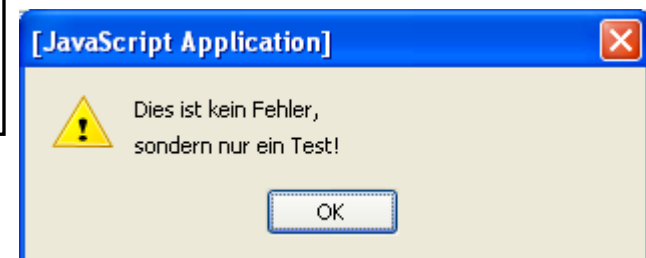
Opera



Netscape



Mozilla Firefox



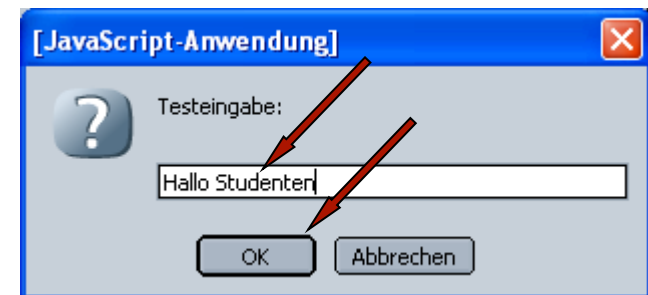
Eingabefenster mit *window.prompt()*

window.prompt ("Anzeige", "Vorgabewert")

- erzeugt Texteingabefenster
- Methode gibt Eingabewert [OK] , bzw. Null [Abbruch] zurück
- geht in allen gängigen Browsern ähnlich gleich gut.

```
<html>
<head>
  <title>Interaktion</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      testvar = prompt("Testeingabe:", " ");
      alert("Es wurde " + testvar + " eingegeben.");
    //-->
  </script>
</body>
</html>
```

Beispiel im Netscape:



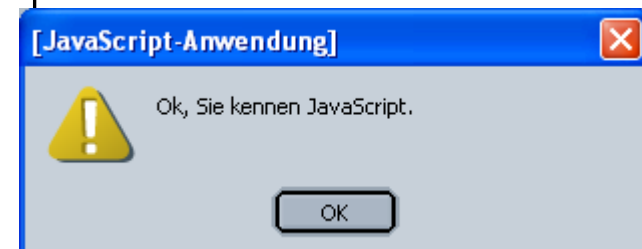
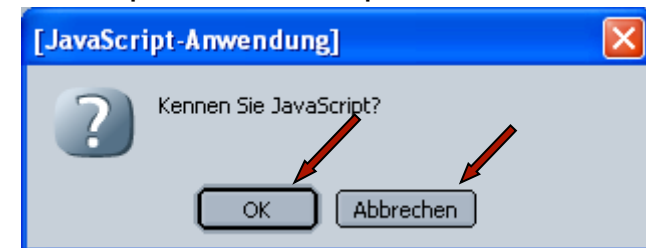
Bestätigungsfenster mit *window.confirm()*

window.confirm() (*"Frage"*)

- Methode erzeugt ein Dialogfenster mit zwei Buttons **[OK]** , **[Abbrechen]**
- Rückgabewert „true“, bei **OK**
- -“- „false“, bei **Abbrechen**
- funktioniert mit gängigen Browsern gleich gut.

```
<html>
<head>
  <title>Interaktion mit window.confirm()</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      auswahlvar = confirm("Kennen Sie JavaScript?");
      if(auswahlvar == true)
        alert("Ok, Sie kennen JavaScript.")
      else
        alert("Sie kennen leider kein JavaScript.");
    //-->
  </script>
</body>
</html>
```

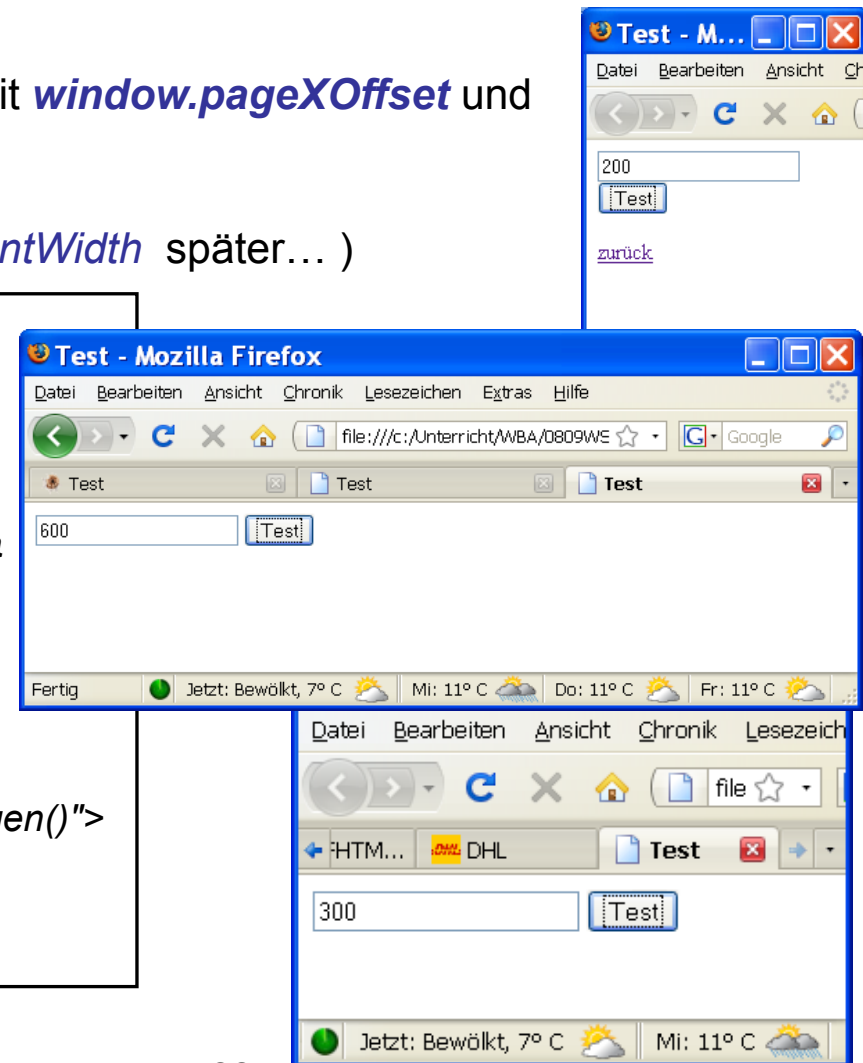
Beispiel im Netscape:



Aktuelle Fensterbreite des Browsers mit ***window.innerWidth*** und ***window.pageXOffset*** bzw. ***window.pageYOffset***

- für absolutes Positionieren von Elementen
- Fensterbreite mit ***window.innerWidth***
- Korrektur von Scrolling in vertikal oder horizontal mit ***window.pageXOffset*** und ***window.pageYOffset***
- funktioniert mit Mozilla Firefox nicht mit MSIE
- (MSIE nutzt document-objekt ***document.body.clientWidth*** später...)

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
function BreiteFestlegen ()
{ window.innerWidth = document.Eingabe.Feld.value; }
</script>
</head>
<body>
<form name="Eingabe" action="">
<input type="text" name="Feld">
<input type="button" value="Test" onclick="BreiteFestlegen()">
</form>
</body>
</html>
```



Clientseitige Implementierungstechnologien (I): Javascript

1. Scriptsprachen
2. Einführung in JavaScript
3. Technisches Umfeld
4. Grundlagen : Auswahl
- 5. Objektmodell**
6. Zugriff auf HTML-Dokumente
7. Event-Handler

Vordefinierte Objekte

1. Navigator
2. Window
- 3. Document**
4. History
5. Location

Das Dokument - Objekt

Das Dokumentobjekt besitzt Eigenschaften, über welche auf die **Elemente des HTML-Dokumentes** zugegriffen werden kann, z.B. für Formulare (`<forms></forms>`), Verweisanker (*anchor*), Grafiken (*images*) und Verweise (*link*-Objekte).

Eigenschaften:

<i>bgColor</i>	Auslesen und Verändern der Hintergrundfarbe
<i>fgColor</i>	Auslesen und Verändern der Textfarbe
<i>linkColor</i>	Auslesen und Verändern der Textfarbe eins noch nicht besuchten Hyperlinks
<i>alinkColor</i>	Festlegen der Textfarbe eines bereits besuchten Hyperlinks
<i>title</i>	Auslesen und Setzen des Dokumententitels
<i>referrer</i>	liefert URL des Dokumentes, welches per Hyperlink das aktuelle Dokument aufgerufen hat
<i>lastModified</i>	Datum der letzten Änderung des aktuellen Dokuments
<i>cookie</i>	Auslesen und Verändern der lokal gespeicherten Cookies
<i>URL</i>	enthält URL des geladenen Dokuments <i>document.URL == document.location</i>

Das Dokument - Objekt

Methoden:

write(Text)

schreibt Text in das HTML-Dokument

writeln(Text)

wie *write()* mit Zeilenumbruch am Ende (Zeilenumbruch nur im Sourcecode der HTML-Seite sichtbar, nicht auf der generierten HTML-Seite)

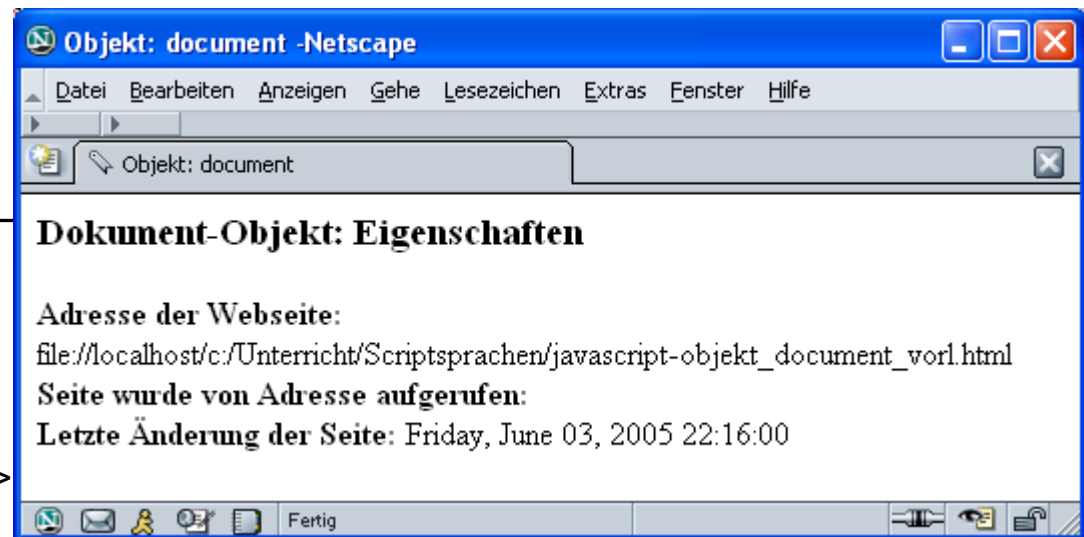
close()

Schließen des Dokumentes

clear()

Ablöschen des Inhalts der aktuellen Webseite

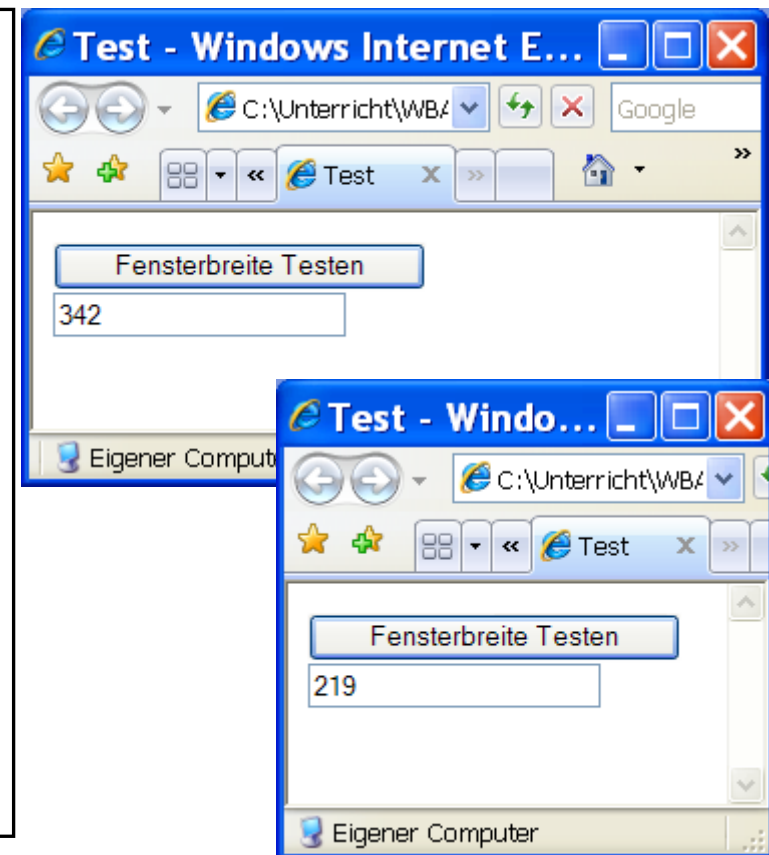
```
<html>
<head>
  <title>Objekt: document</title>
</head>
<body>
  <h3>Dokument-Objekt: Eigenschaften</h3>
  <script type="text/javascript"><!--
    document.write("<b>Adresse der Webseite:</b> " + document.URL + "<br>");
    document.write("<b>Seite wurde von Adresse aufgerufen:</b> „+ document.referrer + "<br>");
    document.write("<b>Letzte Änderung der Seite:</b>" + document.lastModified + "<br>");
  //--></script>
</body>
</html>
```



Aktuelle Fensterbreite des Browsers mit ***document.body.clientWidth***

Für ein absolutes Positionieren von Elementen kann die **Fensterbreite** mit ***document.body.clientWidth*** bestimmt werden. Veränderte Breiten durch Scrolling in vertikaler und horizontaler Richtung werden mit ***document.body.scrollLeft*** und ***document.body.scrollTop*** korrigiert. (funktioniert mit MSIE)

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
function BreiteTesten ()
{
document.Teste.Ausgabe.value = document.body.clientWidth;
} //MSIE
</script>
</head>
<body>
<form name="Teste" action="">
<input type="button" value="Fensterbreite Testen"
onclick="BreiteTesten()">
<input type="text" name="Ausgabe" readonly>
</form>
</body>
</html>
```

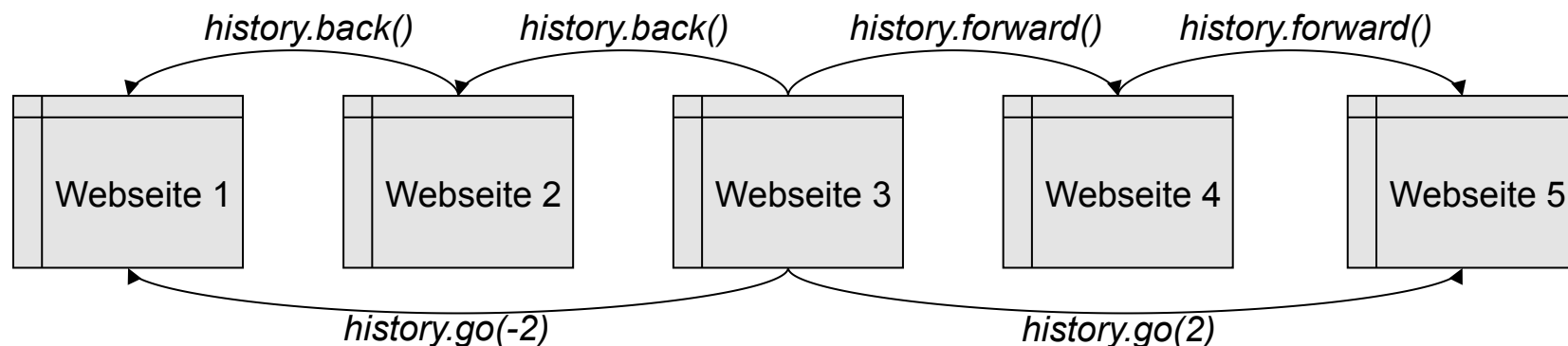


Vordefinierte Objekte

1. Navigator
2. Window
3. Document
- 4. History**
5. Location

Objekt *history*

Das Objekt ***history*** ermöglicht vorwärts und rückwärts einen Zugriff auf die im aktuellen Browserfenster bereits aufgerufenen URLs.



Eigenschaft:

length

Anzahl der History-Einträge im Browser

Methoden:

back()

Zugriff rückwärts: Laden des zuvor besuchten Dokuments

forward()

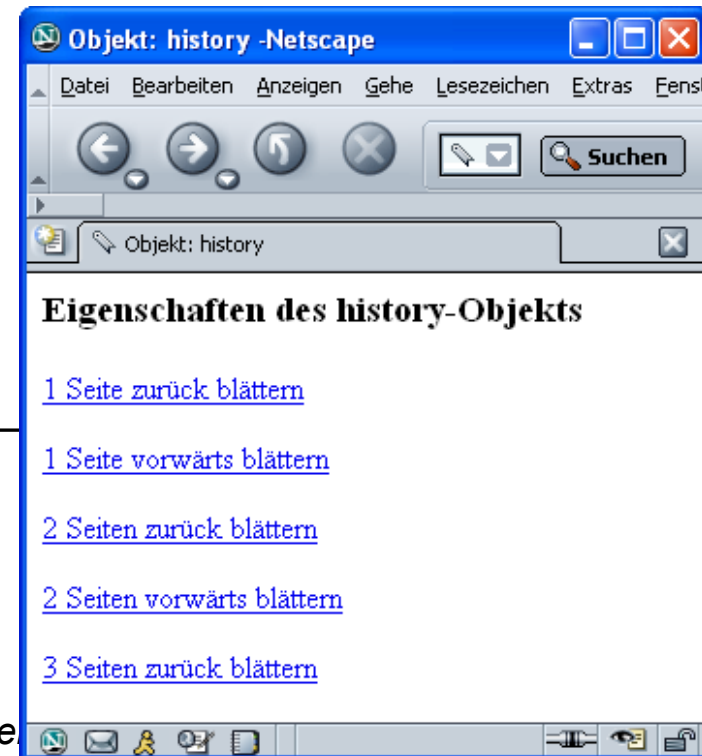
Zugriff vorwärts: Laden des nächsten Dokuments laut History-Obj.

go(steps)

Zugriff vorwärts (steps=positive Zahl) und rückwärts (steps=negative Ziel) mit Überspringen von (Anzahl=steps) Seiten

Objekt *history*: Beispiel

```
<html>
<head>
  <title>Objekt: history</title>
</head>
<body>
  <h3>Eigenschaften des history-Objekts</h3>
  <p> <a href="javascript:history.back();"> 1 Seite zurück blättern</a>
  <p> <a href="javascript:history.forward();">1 Seite vorwärts blättern</a>
  <p> <a href="javascript:history.go(-2);">2 Seiten zurück blättern</a>
  <p> <a href="javascript:history.go(2);">2 Seiten vorwärts blättern</a>
  <p> <a href="javascript:history.go(-3);">3 Seiten zurück blättern</a>
</body>
</html>
```



Vordefinierte Objekte

1. Navigator
2. Window
3. Document
4. History
- 5. Location**

Objekt *location*

Das Objekt ***location*** erlaubt es, die URL des aktuellen Fensters auszulesen und zu bearbeiten.

Format einer URL:

protocol://hostname[:port][/path]/[filename][#section]

Eigenschaft:

<i>protocol</i>	Ausgabe des Internet-Protokolls, mit dem die aktuelle Webseite aufgerufen wurde (z.B. http, ftp, ...)
<i>hostname</i>	Auslesen des Hostnamens, Domainnamens oder der IP-Adresse
<i>port</i>	Ausgabe der Portnummer des Servers
<i>pathname</i>	Auslesen der Pfadangabe des Dokuments
<i>search</i>	Ausgabe des Parameterstrings
<i>hash</i>	Ausgabe des Verweisankers
<i>href</i>	Auslesen der kompletten URL
<i>host</i>	Ausgabe von <i>hostname</i> und <i>port</i>

Objekt *location*: Beispiel

```
<html>
<h3>location-Objekt:Eigenschaften</h3>
<script type="text/javascript">
  <!--
    document.write("<b>protocol: </b>" + window.location.protocol+"</br>");
    document.write("<b>hostname:</b>" + window.location.hostname+"</br>");
    document.write("<b>port:</b>" + window.location.port+"</br>");
    document.write("<b>pathname:</b>" + window.location.pathname+"</br>");
    document.write("<b>search:</b>" + window.location.search+"</br>");
    document.write("<b>hash:</b>" + window.location.hash+"</br>");
    document.write("<b>href:</b>" + window.location.href+"</br>");
    document.write("<b>host:</b>" + window.location.host+"</br>");

  //-->
</script>
</html>
```



Bis hierher Teil 1

JavaScript

1. Einführung in JavaScript
2. Technisches Umfeld
3. Objektmodell
- 4. Zugriff auf HTML-Dokumente**
5. Event-Handler

Zugriff auf HTML – Dokumente

Grundlagen:

- **document-Objekt** ermöglicht Zugriff auf html-Seite
- **document -Objekt** ist Element des **window-Objekts**
- **document-Objekt** besitzt mehrere Unterobjekte
- Vorhandensein des Tag **<forms> ... </forms>** ermöglicht Zugriff auf
Formularelemente und Formulareigenschaften einer Webseite
- Aufbau der Webseite bestimmt Vorhandensein von Instanzen eines
Unterobjekts

HTML (Erinnerung) : Formulare

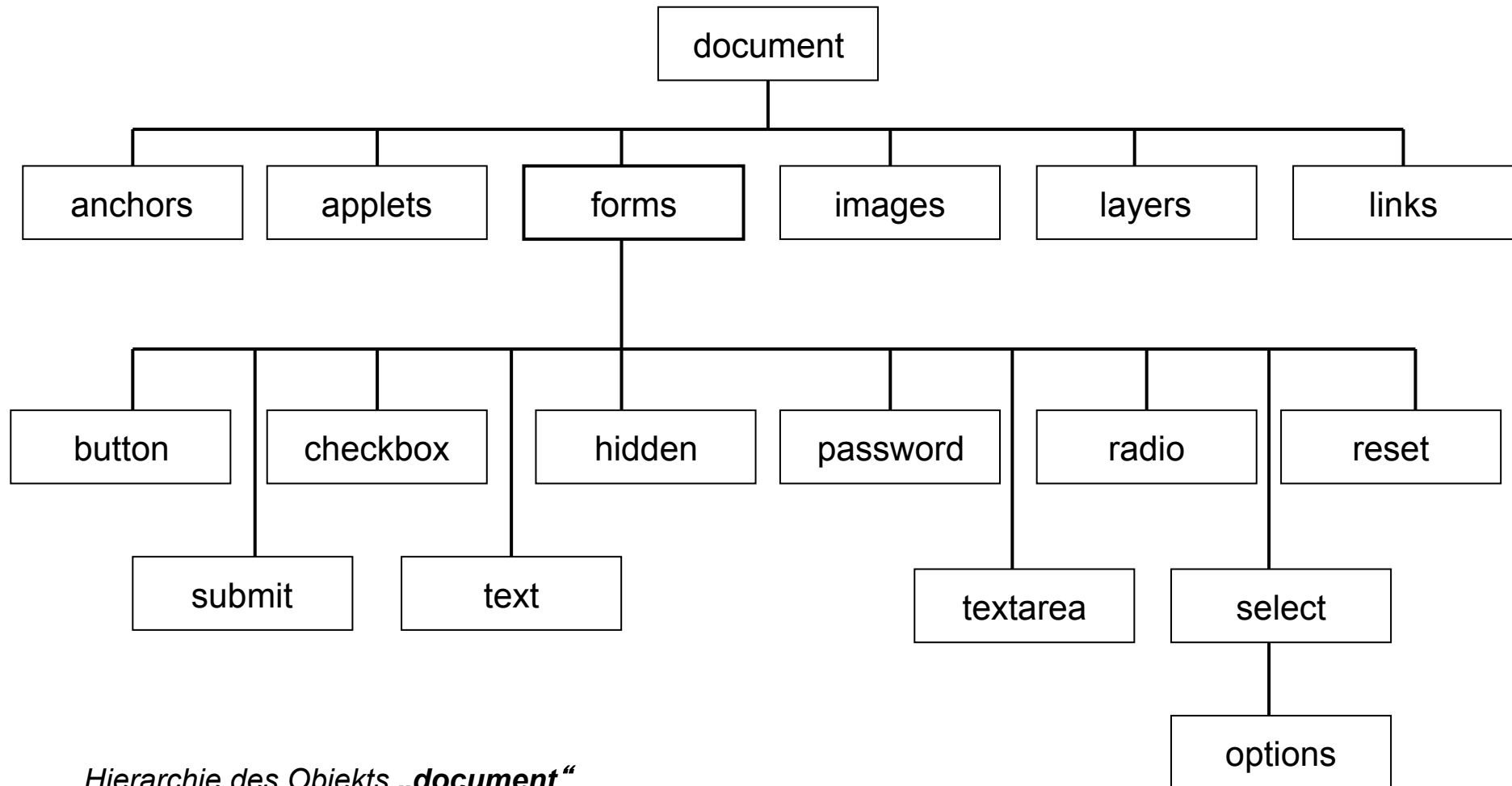


Formulardefinition:

```
<form name="Formularname" action=mailto:irgendwer@webmail.de  
method="POST|GET" enctype="text/plain" target="Ziel">  
...Formularelemente...  
</form>
```

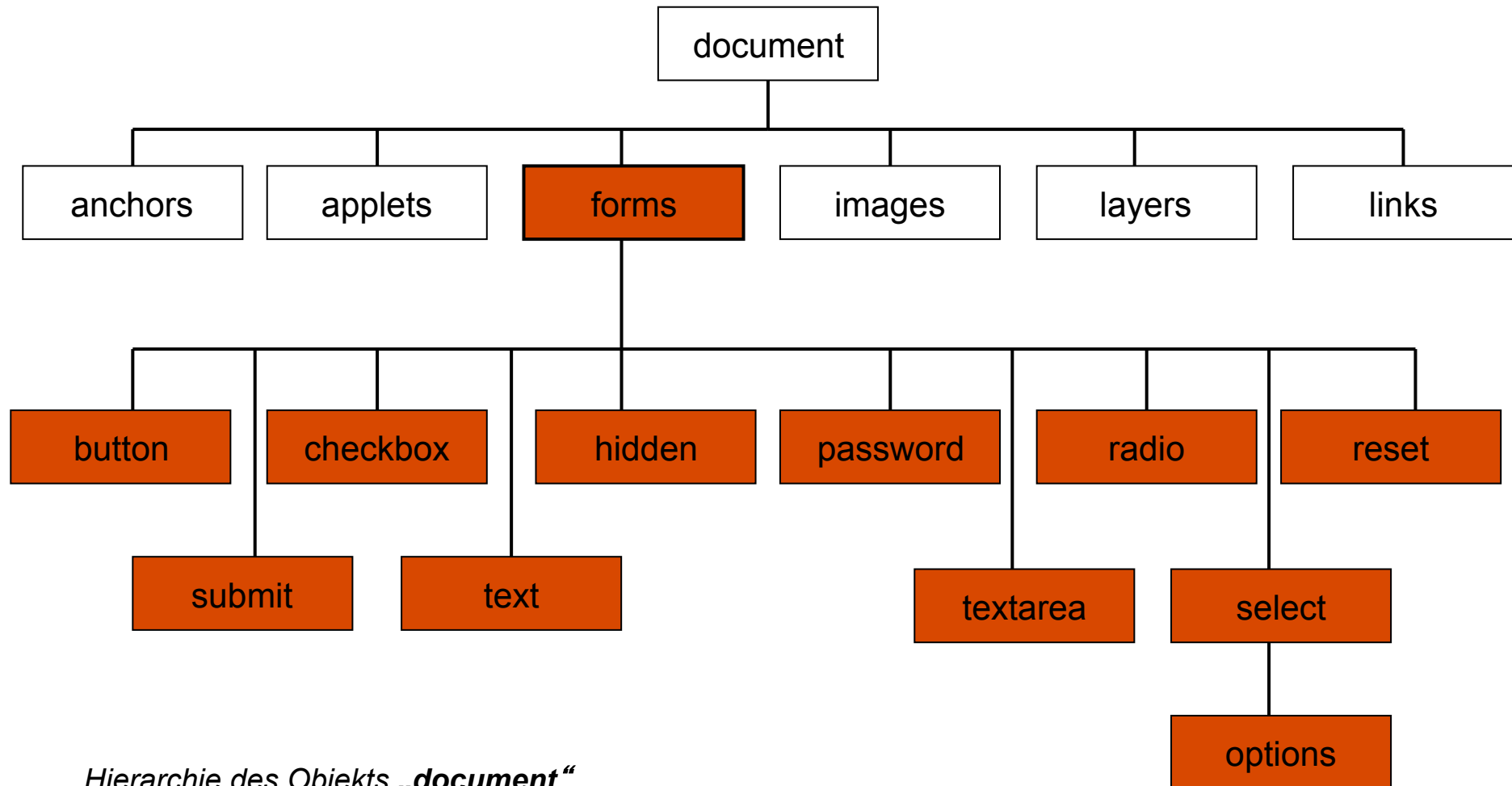
- Tag **<form>** #engl. Formular – leitet ein Formular ein. In ihm können Schaltflächen, Eingabefelder und Auswahllisten definiert werden; Verschachtelung des Tags <form> ist nicht möglich.
- Attribut **name** – legt Formularname fest; kann z.B. von Javascript aufgerufen werden
- Attribut **target** – legt das Zielfenster für Rückmeldung des Skriptes fest
- Attribut **method** bestimmt Versendemethode (Post/Get)
- Attribut **action** – gibt Programm an, das nach Abschicken der Daten ausgeführt wird und zu dem die Daten gesendet werden
- Attribut **enctype** – Mimetype, z.B. "text/plain"

Zugriff auf HTML - Dokumente



Hierarchie des Objekts „document“

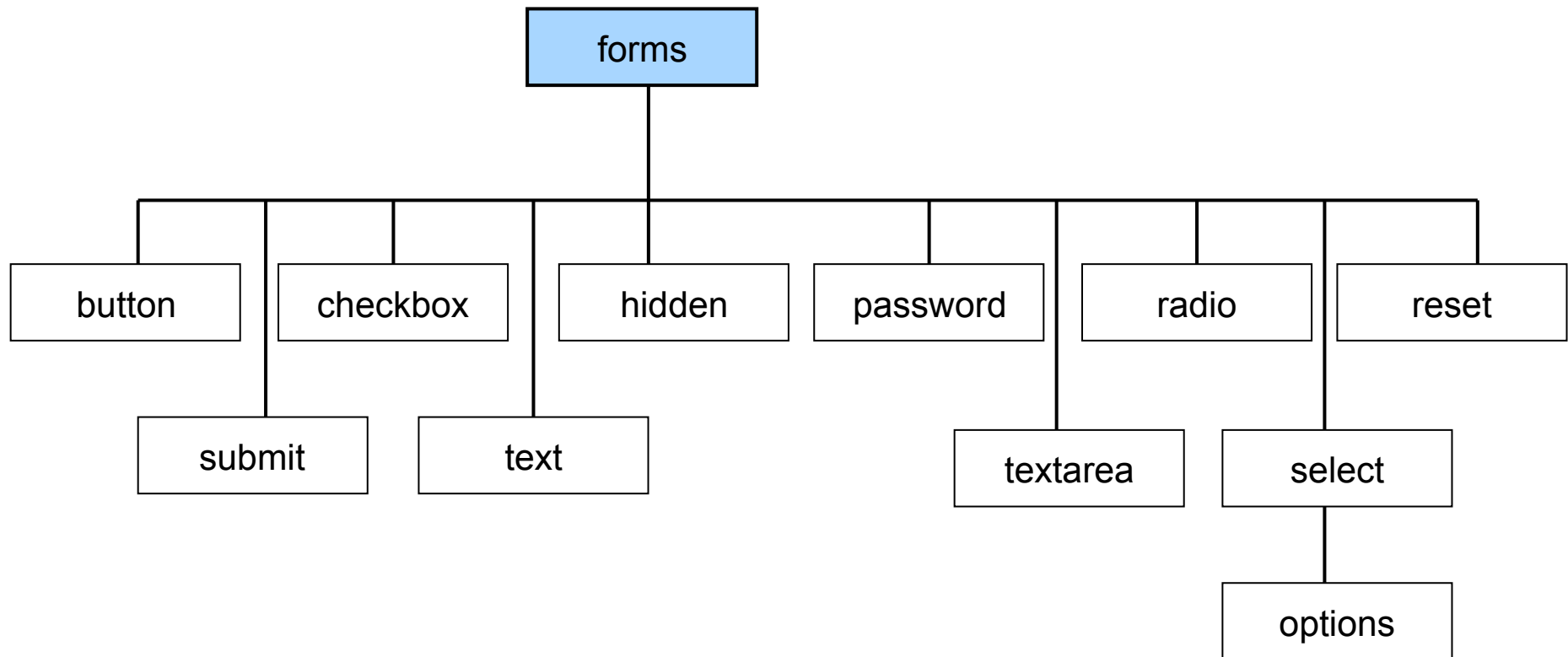
Zugriff auf HTML - Dokumente



Hierarchie des Objekts „document“

Unterobjekt *forms*

Kann auf einer Webseite dazu verwendet werden, Informationen zu erfassen und an einen Empfänger zu übertragen. Das Forms-Objekt besitzt Unterobjekte, die den Unterelementen des HTML-Tags `<form>` entsprechen.



Unterobjekt *forms*: *Eigenschaften und Methoden des forms-Objekts*

Eigenschaften:

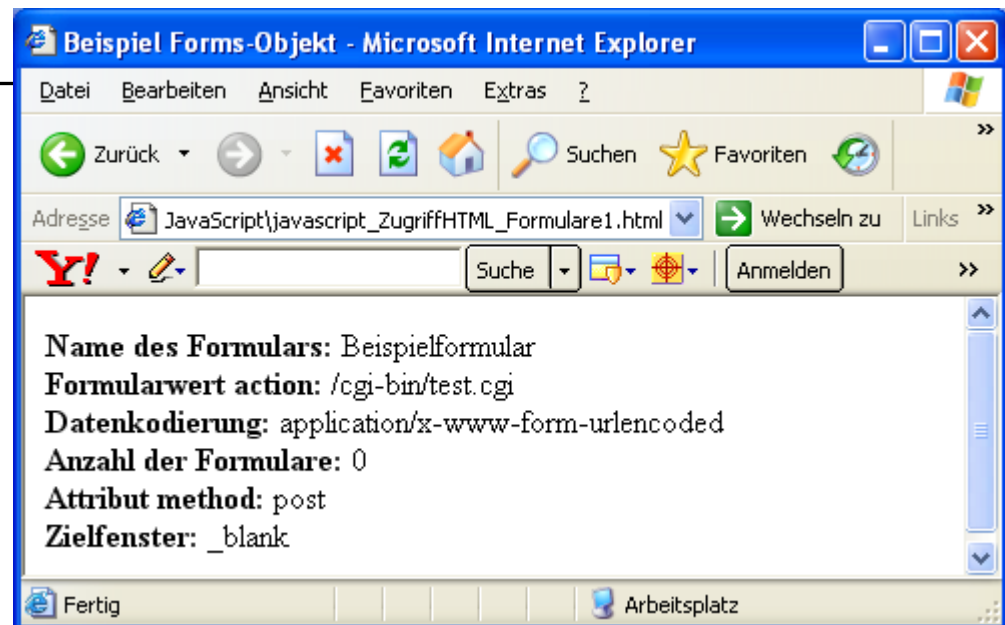
<i>action</i>	enthält Formularwert „ <i>action</i> “ z.B. URL des Scripts, welches die Daten verarbeiten soll
<i>encoding</i>	Ermitteln oder Setzen der Datenkodierung (z.B. „/text/plain“)
<i>length</i>	Anzahl der Elemente im Formular
<i>method</i>	enthält Wert des method - Attributs
<i>name</i>	enthält Name des Formulars
<i>target</i>	enthält Zielfenster für Rückmeldungen aus dem Serverskript; entspricht dem HTML-Attribut: <i><form target = “ “></i>

Methoden:

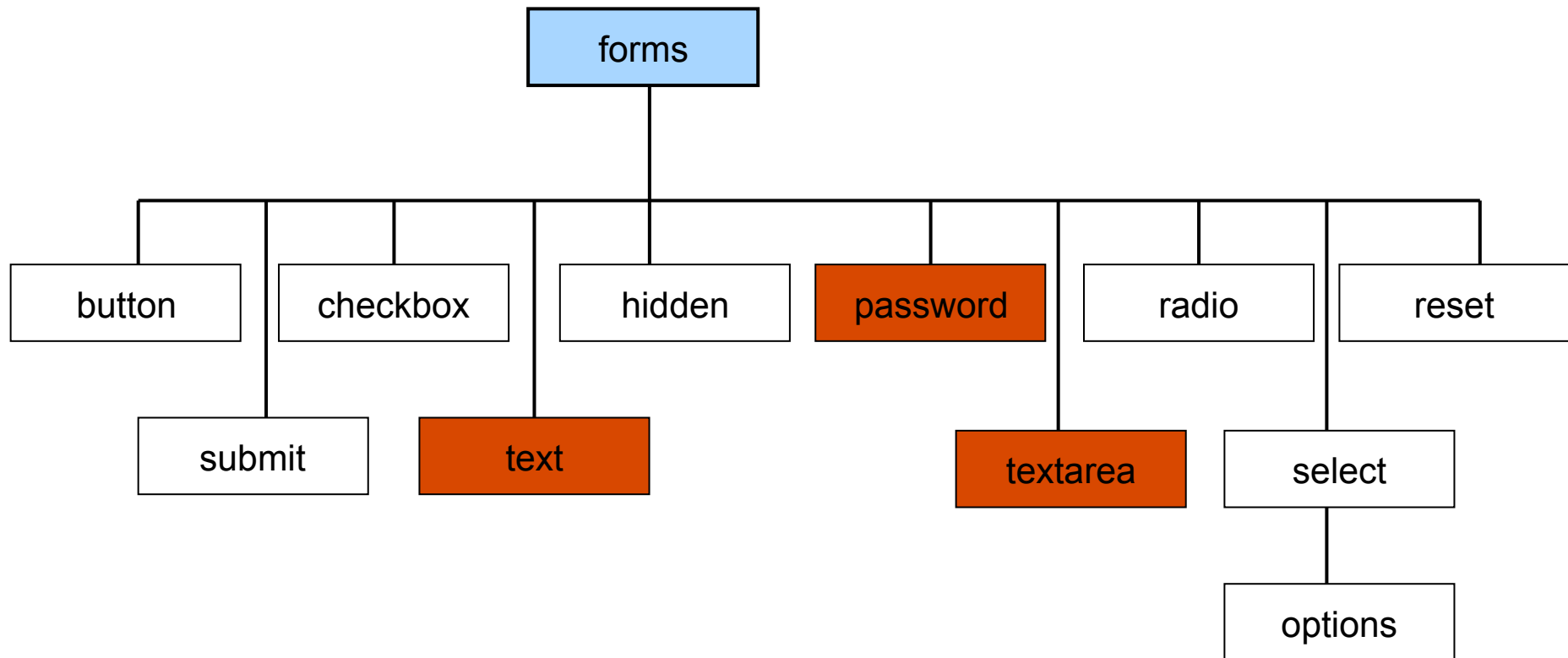
<i>reset()</i>	Zurücksetzen aller Formuleinträge auf den Anfangswert
<i>submit()</i>	Absenden der Formulardaten an das Ziel wie im attribut <i>action</i> festgelegt.

Unterobjekt *forms*: Beispiel zu *forms*-Objekt

```
<html>
<form name = "Beispielformular" action="/cgi-bin/test.cgi" method="post" encoding="text/plain" target="_blank">
<script type="text/javascript">
  <!--
    document.write("<b>Name des Formulars: </b> " + document.forms[0].name + "<br>");
    document.write("<b>Formularwert action: </b> " + document.forms[0].action + "<br>");
    document.write("<b>Datenkodierung: </b> " + document.forms[0].encoding + "<br>");
    document.write("<b>Anzahl der Formulare: </b>" + document.forms[0].length + "<br>");
    document.write("<b>Attribut method: </b> " + document.forms[0].method + "<br>");
    document.write("<b>Zielfenster: </b> " + document.forms[0].target + "<br>");
  //-->
</html>
```



Unterobjekt *forms*



Unterobjekt *forms*: Eingabefelder

Html-Tags zur Definition von Eingabefeldern:

1. `<input type="text">` - einzeliges Feld
2. `<input type="textarea">` - mehrzeiliges Feld
3. `<input type="password">` - nicht lesbare Passworteingabe

JavaScript-Objekt:

text
textarea
password

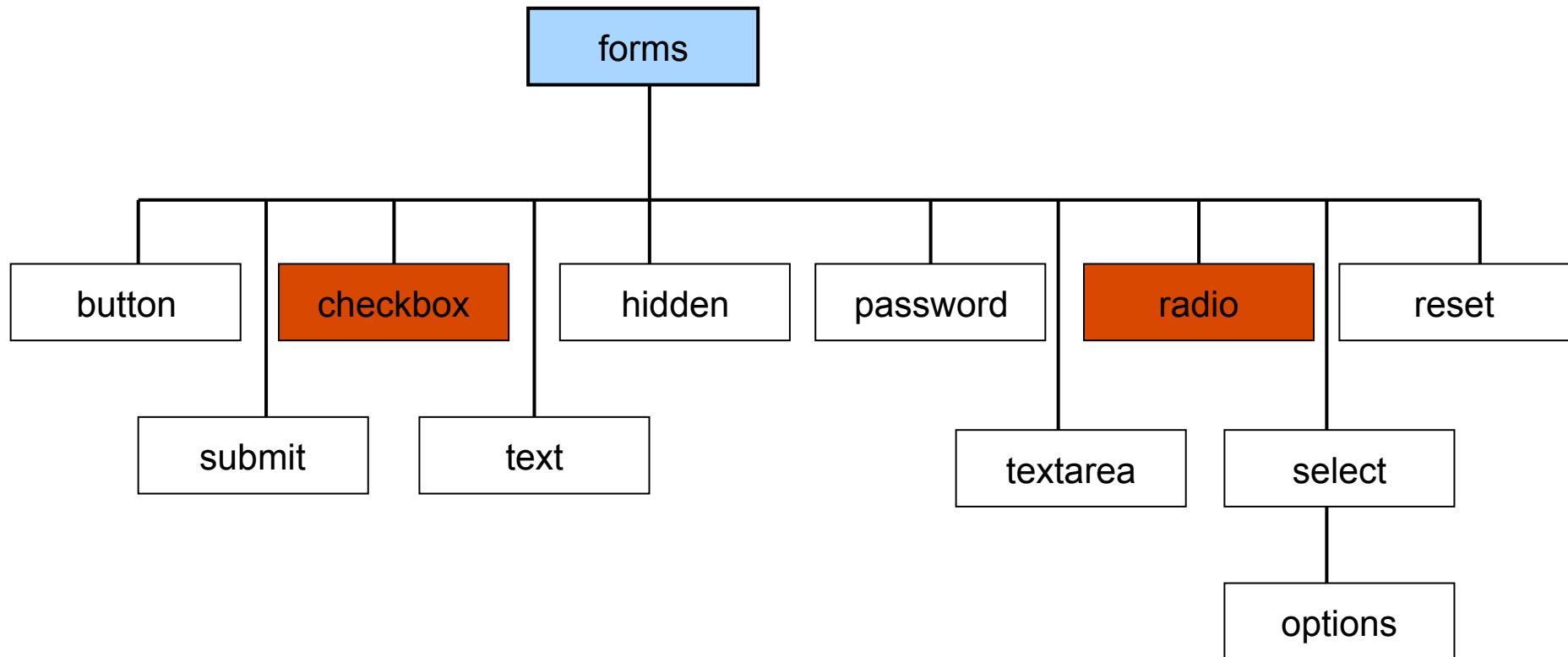
Eigenschaften:

<i>defaultname</i>	enthält Formularwert „ <i>value</i> “ aus HTML-Quelltext
<i>form</i>	Zugriff auf Formularnamen, zu dem das Element (text, passw,...)gehört
<i>name</i>	liefert Feldnamen des Objekts
<i>type</i>	liefert Objekttyp (<i>text</i> , <i>textarea</i> , <i>password</i>)
<i>value</i>	Zugriff auf Inhalt des Eingabefeldes

Methoden:

<i>blur()</i>	Verlassen des Feldes
<i>focus()</i>	Setzen des Mouse-Cursors in das entsprechende Eingabefeld
<i>select()</i>	Auswahl des Feldinhalts

Unterobjekt *forms*



Unterobjekt *forms*:

Kontroll- und Optionsfelder

Html-Tags zur Definition von Kontrollfeldern:

JavaScript-Objekt:

- | | | |
|---|--|------------------------|
| 1. <code><input type="radio"></code> - | Kontrollfeld, nur eins pro Gruppe kann markiert werden | <i>radio</i> |
| 2. <code><input type="checkbox"></code> - | Optionsfeld, mehrere können markiert werden | <i>checkbox</i> |

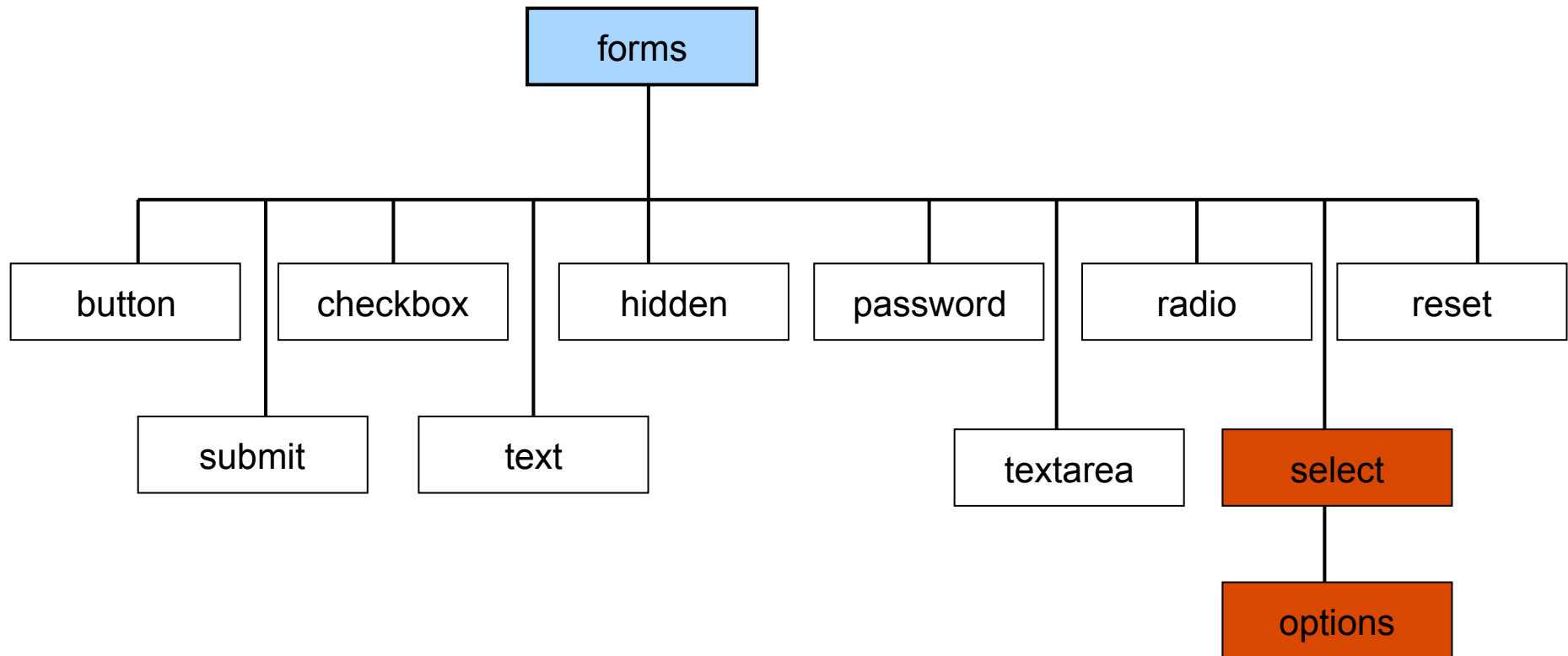
Eigenschaften:

<i>checked</i>	Wert: true => markiert, false => nicht markiert
<i>defaultChecked</i>	Wert: true => Element beim Laden aktiviert, false=> nicht aktiviert
<i>form</i>	enthält Namen des Formulars
<i>length</i>	Anzahl der Optionsfelder innerhalb einer Optionsgruppe
<i>name</i>	Name des Kontroll- oder Optionsfeldes wie im HTML-Dokument
<i>type</i>	enthält Typ des Objekts (<i>checkbox</i> , <i>radio</i>)
<i>value</i>	enthält Übergabewert

Methoden:

<i>blur()</i>	Verlassen des Feldes
<i>click()</i>	Funktion, welche bei Auswahl des Feldes ausgeführt werden soll
<i>focus()</i>	Setzen des Mause-Cursors in das entsprechende Kontroll- oder Eingabefeld

Unterobjekt *forms*



Unterobjekt *forms*:

Auswahllisten (*select*-Objekt)

Html-Tags zur Definition von Auswahllisten:

<select multiple> - Selection einer oder mehrerer
Einträge möglich

JavaScript-Objekt:

select

HTML - Beispiel:

```
<select multiple Name="Item" Size = "3">  
    <option>&Auml;pfel</option>  
    <option>Birnen</option>  
    <option>Quitten</option>  
</select>
```

Unterobjekt *forms*: *Eigenschaften und Methoden des select-Objekts*

Eigenschaften:

<i>form</i>	enthält Formularnamen
<i>length</i>	Anzahl der Optionen in der Auswahlliste
<i>name</i>	enthält Namen der Auswahlliste, wie <i>Name</i> aus HTML-Tag
<i>options[]</i>	ermöglicht Zugriff auf die Einträge
<i>selectedIndex</i>	enthält Index der ausgewählten Option, bei Mehrfachauswahl den Index des ersten ausgewählten Eintrags
<i>type</i>	enthält Typ des Objekts: <i>“select-one”</i> => Einfachauswahl <i>“select-multiple”</i> => Mehrfachauswahl
<i>value</i>	enthält Übergabewert

Methoden:

<i>blur()</i>	Verlassen der Auswahlliste
<i>focus()</i>	Setzen des Mause-Cursors in die entsprechende Auswahlliste

Unterobjekt *forms*: *options* – Array für Auswahllisten

<u>Html-Tags zur Def. der Einträge von Auswahllisten:</u>	<u>JavaScript-Objekt:</u>
<code><option></code> - Einträge der Auswahlliste	<i>options[]</i>

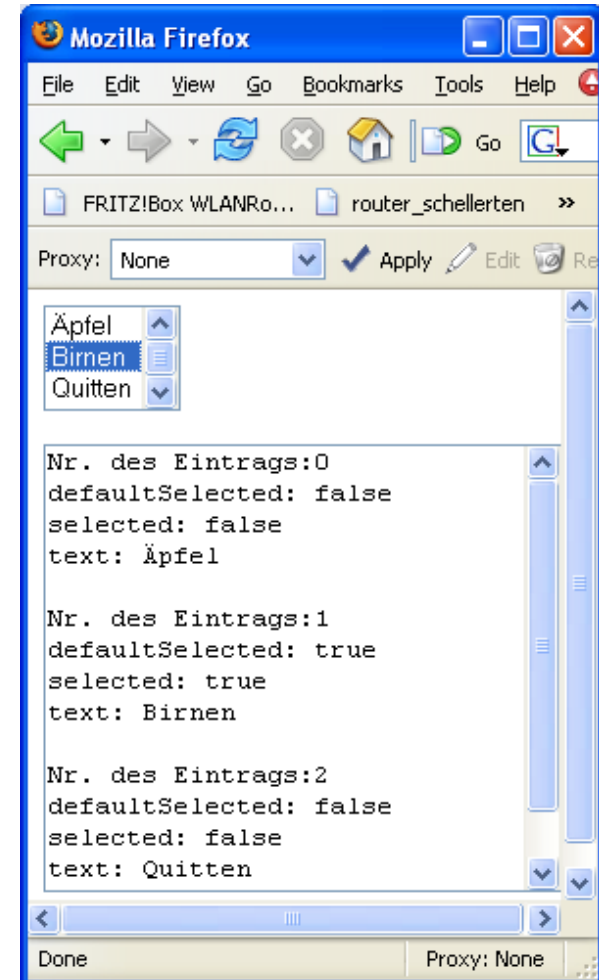
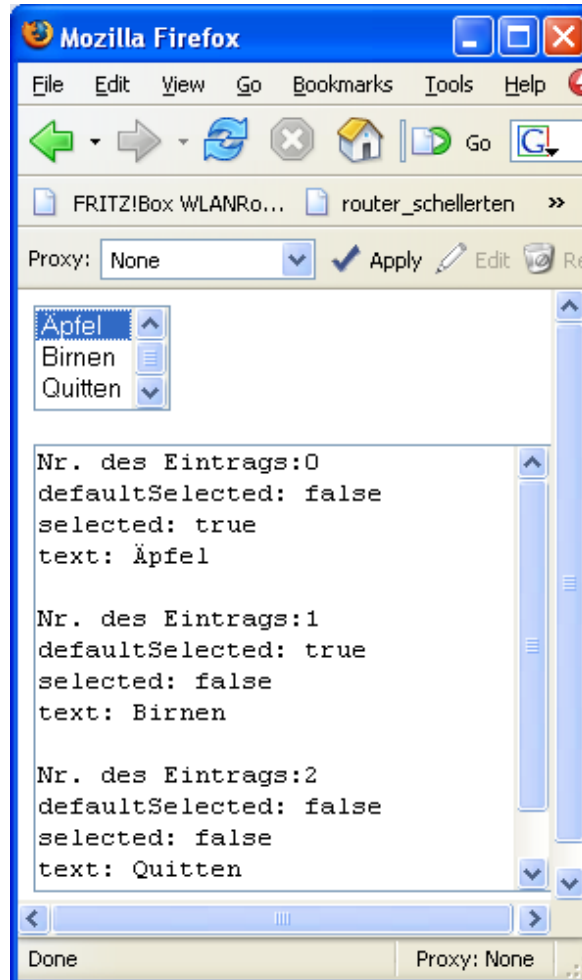
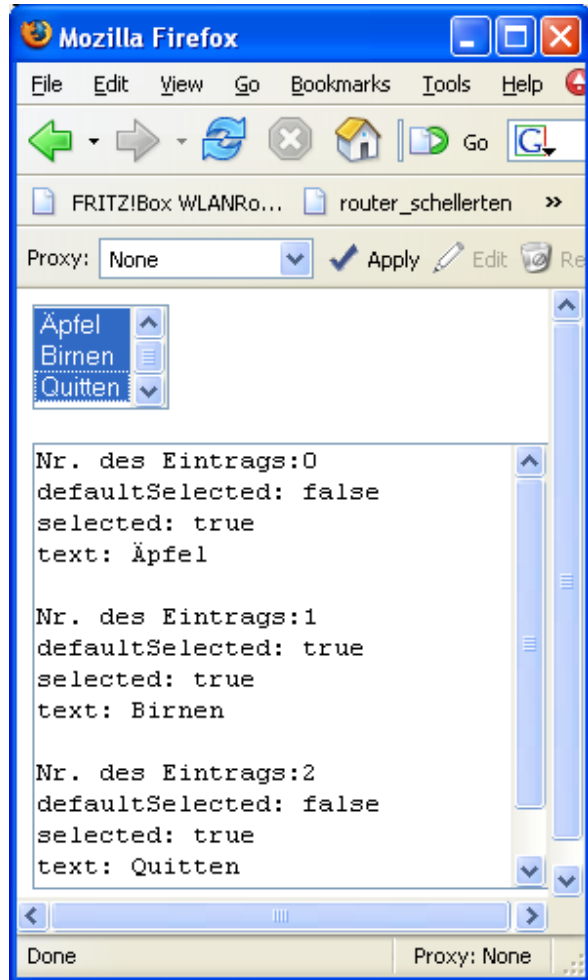
Eigenschaften:

<i>defaultSelected</i>	Wert = true => Auswahl voreingestellt, = false => nicht voreingestellt
<i>index</i>	Nummer des Eintrags
<i>length</i>	Anzahl der Listenelemente
<i>selected</i>	enthält Auswahl durch Benutzer
<i>text</i>	enthält Ausgabertext für die entsprechende Auswahloption
<i>value</i>	enthält zusätzlichen Wert für die entsprechende Auswahl, falls diese im HTML-Quelltext angegeben wurde (optional)

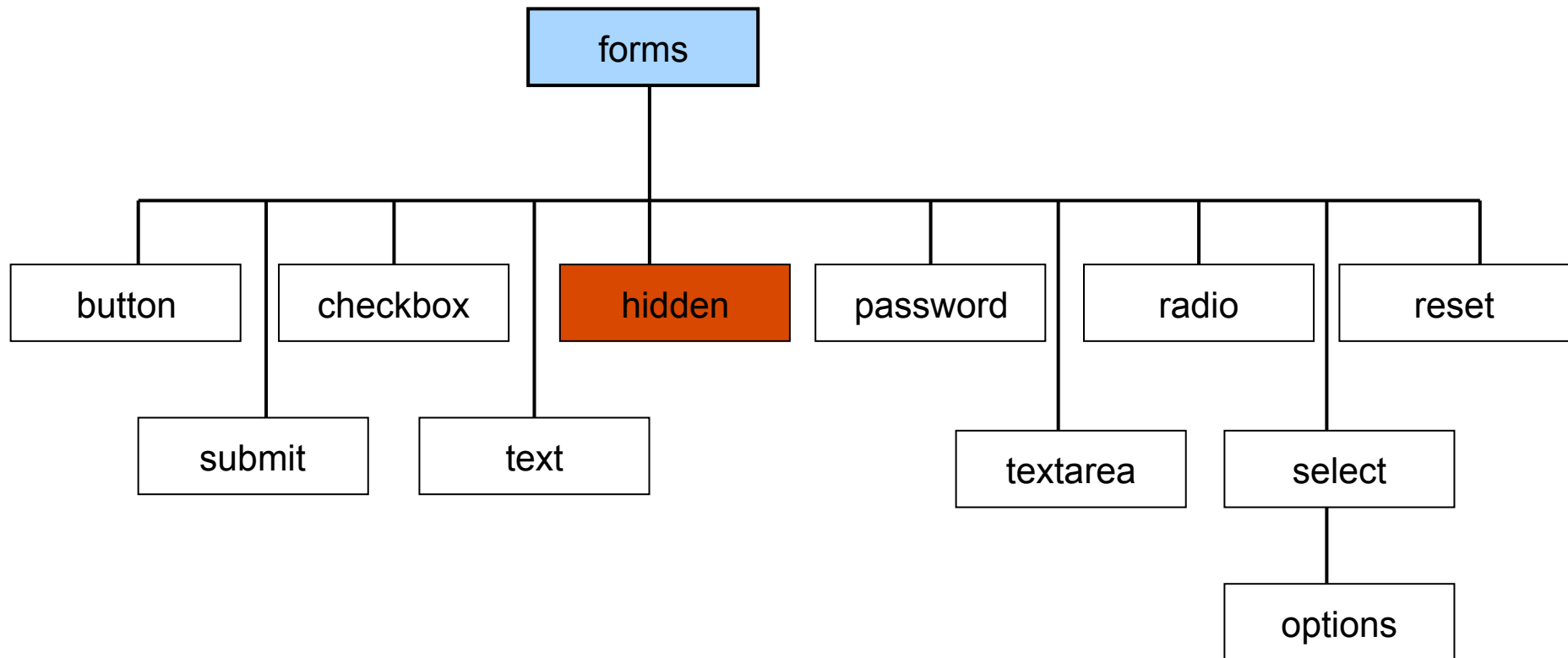
Unterojekt *forms: options* – Objekt Beispiel

```
<html>
<script type="text/javascript">
  function durchlauf() <!-- zeigt Eigenschaften des option-Elements an ->
  { var text = "";
    for (i = 0; i < document.formular.Item.length; i++)
      {text = text + 'Nr. des Eintrags:' + document.formular.Item.options[i].index + '\n'
        + 'defaultSelected: ' + document.formular.Item.options[i].defaultSelected + '\n'
        + 'selected: ' + document.formular.Item.options[i].selected + '\n'
        + 'text: ' + document.formular.Item.options[i].text + '\n\n';}
    document.formular.ausgabe.value = text;} <!-- Zuweisen des Strings an Option value von ausgabe ->
  }
</script>
<body onLoad="durchlauf();"> <!-- fkt. durchlauf() wird nach Laden der Webseite initial aufgerufen ->
  <form name="formular">
    <select name="Item" size="3" multiple onChange="durchlauf();"> <!-- Aktualisierung der Anzeige ->
      <option>&Auml;l;pfel</option>
      <option selected>Birnen</option>
      <option>Quitten</option>
    </select></p>
    <textarea cols="30" rows="13" name="ausgabe"></textarea> <!-- mehrzeiliges Textfeld ->
  </form>
</body>
</html>
```


Unterobjekt *forms: options* – Objekt Beispiel



Unterobjekt *forms*



Das hidden-Objekt: versteckte Felder

Html-Tag für verstecktes Feld:

`<input type = "hidden" ...> -`

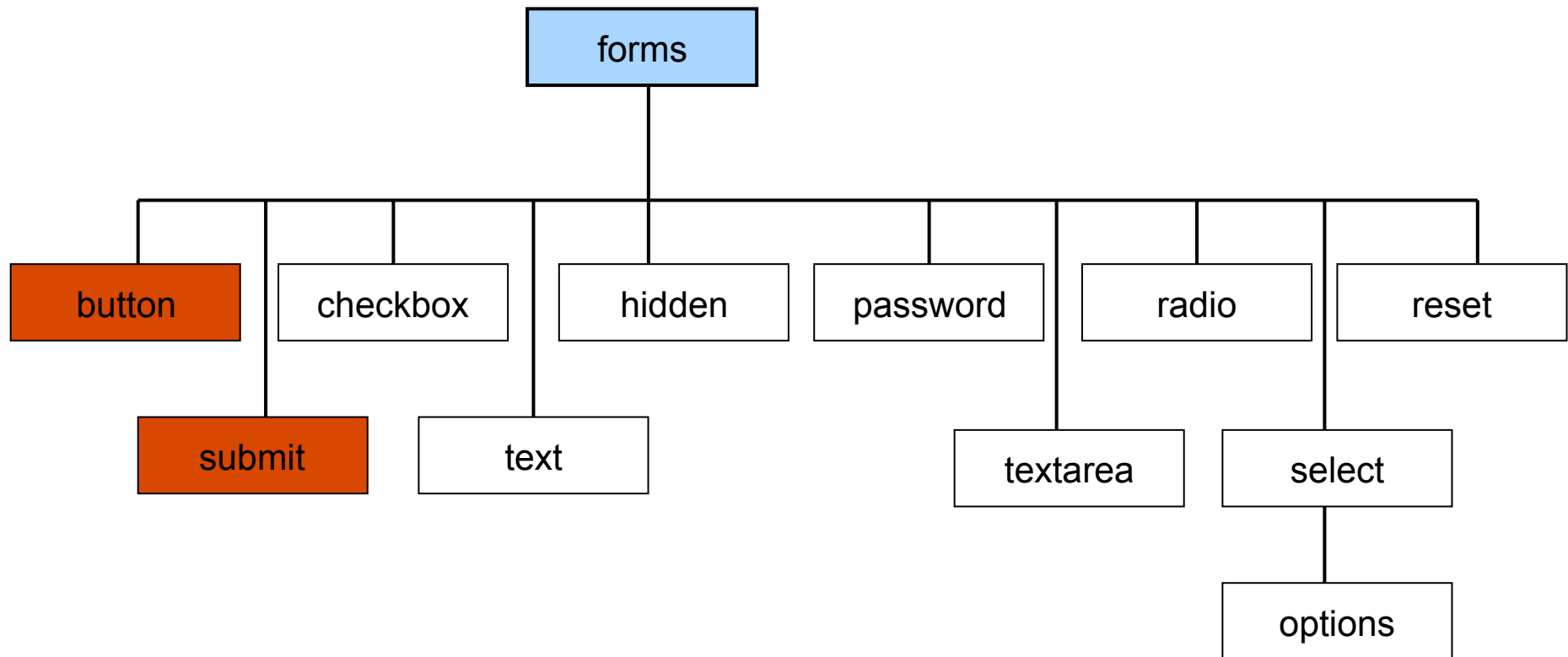
JavaScript-Objekt:

hidden

Eigenschaften des hidden - Objekts:

<i>form</i>	enthält Name des Formulars
<i>name</i>	enthält Name des Objekts
<i>type</i>	enthält Typ des Objekts (<i>hidden</i>)
<i>value</i>	enthält Wert des versteckten Feldes

Unterobjekt *forms*



Das button-Objekt: Schaltflächen

Html-Tag für Schaltflächen:

JavaScript-Objekt:

`<input type = "...“ name = "... “...>` - Zugriff über `document.“form-name“.“input-name“`

Eigenschaften des *button* – Objekts :

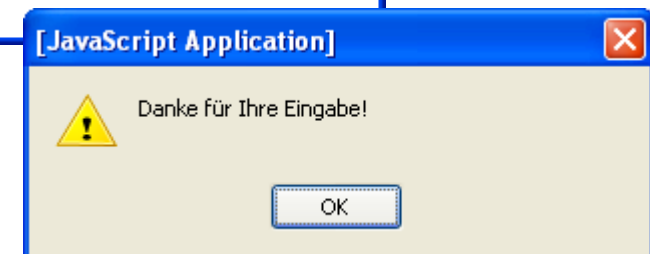
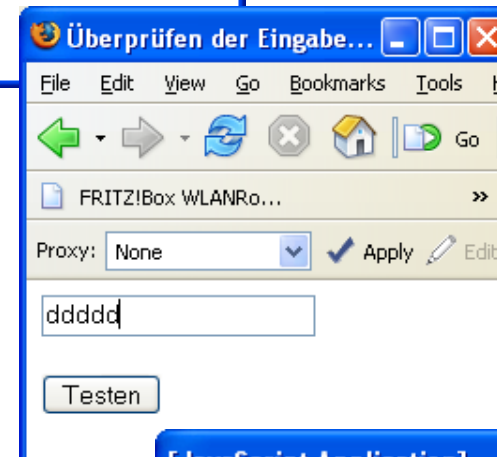
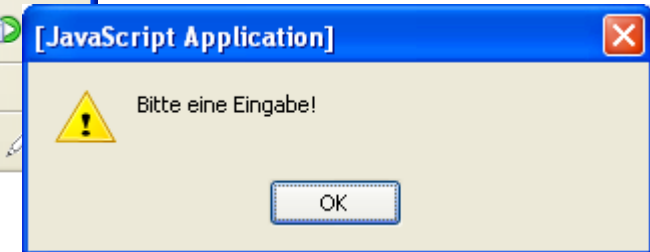
<i>form</i>	enthält Name des Formulars
<i>name</i>	enthält Name der Schaltfläche
<i>type</i>	enthält Typ der Schaltfläche
<i>value</i>	enthält Beschriftung der Schaltfläche

Methode des *button* – Objekts:

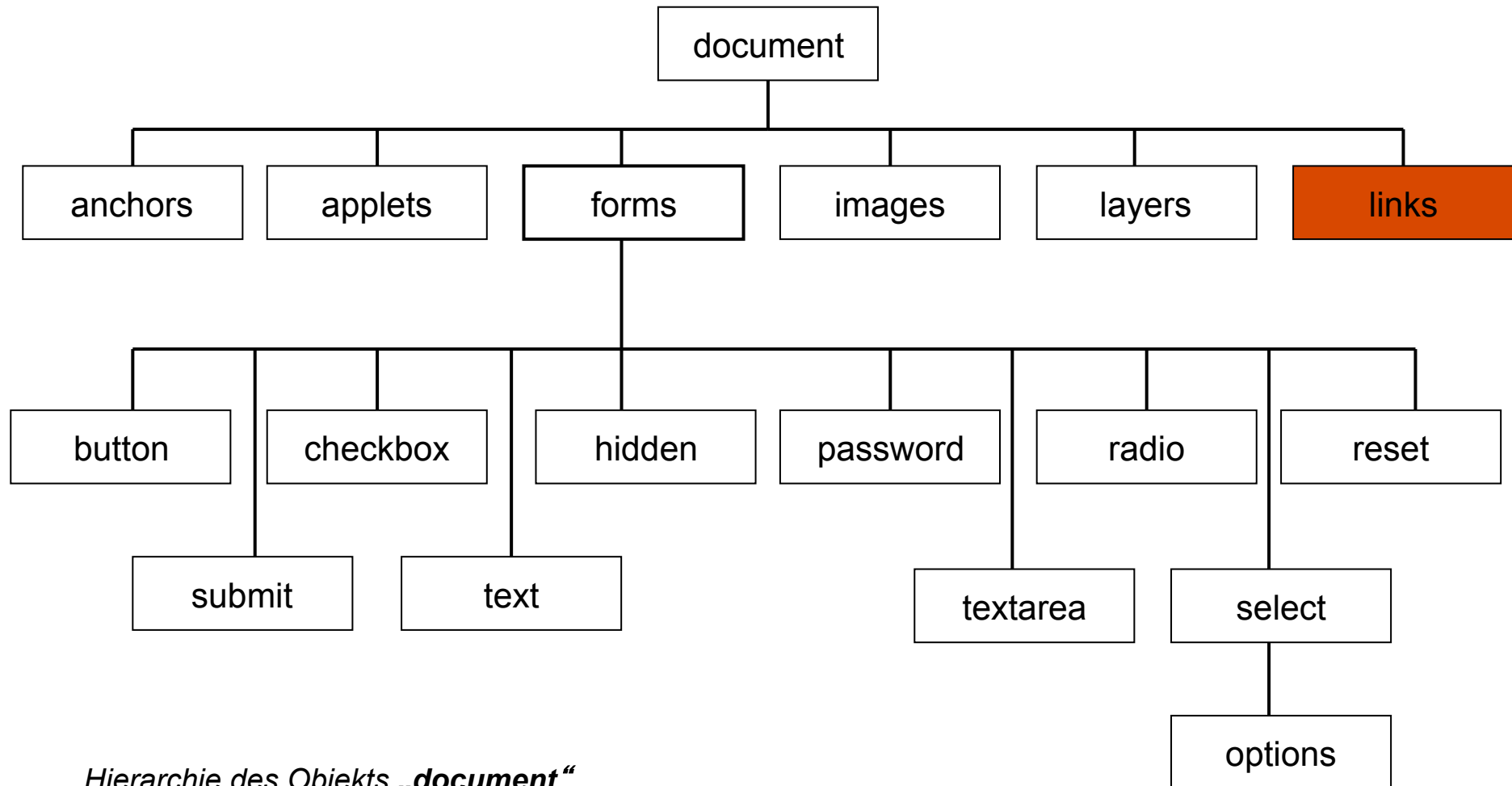
<i>click()</i>	Funktion, welche bei Auswahl des Feldes ausgeführt werden soll
-----------------------	--

Anwendung des *form* – Objekts: Überprüfen der Eingabefelder

```
<html>
<head>
  <title>Überprüfen der Eingabefelder</title>
  <script type="text/javascript">
    <!--
      function testen()
      {eingabe = document.Eingabe.Feld.value;
        if (eingabe == "")
        { alert("Bitte eine Eingabe!")
          document.Eingabe.Feld.focus();
          return false;}
        else
        {alert("Danke für Ihre Eingabe!");
          return true;}}
    //-->
  </script>
</head>
<body>
  <form action="" name="Eingabe" onSubmit="return testen();">
    <input type="Text" name="Feld"> <br> <br>
    <input type="submit" name="Testsubmit" value="Testen">
  </form>
</body>
</html>
```



Zugriff auf HTML - Dokumente



Hierarchie des Objekts „document“

Unterobjekt *links*

Html-Tags zur Def. von Verweisen:

<a href = "Verweisziel" target=
"Fenstername" name = "link1 ">
(z.B. link1) Verweistext ****

JavaScript-Objekt:

document.links[index]
document.links.name

Eigenschaften des Unterobjekts *links*:

length

Anzahl der Verweise in einer Webseite

name

enthält Namen des Verweises (z.B. "link1")

target

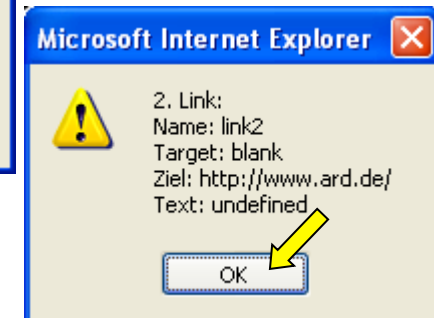
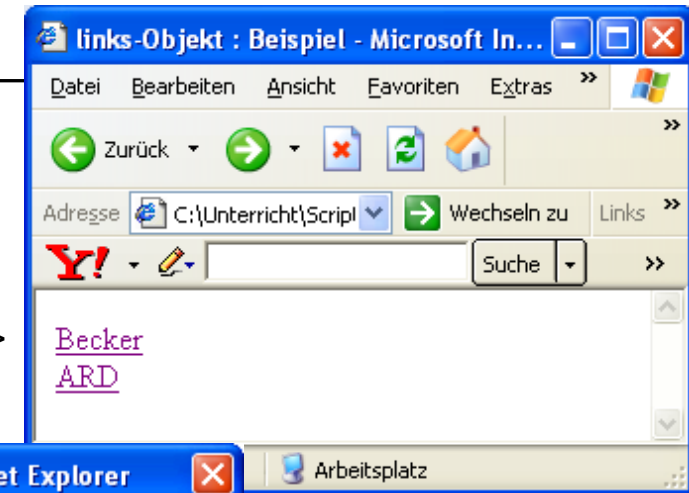
enthält Zielfenster für die Darstellung des Verweises

text

enthält Text des Verweises (nicht bei MSIE)

Unterobjekt *links* : Beispiel

```
<html>
<head>
  <title>links-Objekt : Beispiel</title>
</head>
<body>
  <a href="http://www.beckerclub.de/" name="link1">Becker</a> <br>
  <a href="http://www.ard.de" target="blank" name="link2">ARD</a>
  <script type="text/javascript">
    <!--
    for(i = 0; i < document.links.length; i++)
    {
      alert((i + 1) + ". Link:" + "\n" +
        "Name: " + document.links[i].name + "\n" +
        "Target: " + document.links[i].target + "\n" +
        "Ziel: " + document.links[i].href + "\n" +
        "Text: " + document.links[i].text + "\n");
    }
    //-->
  </script>
</body>
</html>
```



Clientseitige Implementierungstechnologien (I): Javascript

1. Scriptsprachen
2. Einführung in JavaScript
3. Technisches Umfeld
4. Grundlagen : Auswahl
5. Objektmodell
6. Zugriff auf HTML-Dokumente
- 7. Event-Handler**

Event Handler : Grundlagen

Interaktive Webseiten ermöglichen z.B.:

- Abfangen fehlerhafter Eingaben bei Formularen (weniger Datentransfer, Entlasten des Webserver)
- Mausbewegungen und Mausklicks werden ausgewertet
- Auslösen von Operationen wie:
 - Starten von Countdowns
 - Öffnen neuer Fenster

Event – Handler werden als Attribut im HTML-Tag definiert. Sie definieren Ereignisse, auf welche der Browser mit dem Ausführen einer JavaScript-Funktion reagiert.

Diese **JavaScript – Funktionen** werden meist im Kopf des HTML-Dokuments definiert.

Event Handler : Reaktion auf Ereignisse

Ausführen von JavaScript – Anweisungen:

- **JavaScript – Anweisungen** werden sofort nach Laden der Webseite vom Browser ausgeführt
- Anweisungen aus **JavaScript – Funktionen** werden erst nach Funktionsaufruf ausgeführt ; Funktionsaufrufe können an Ereignisse gekoppelt sein



Event Handler : Ereignisse I

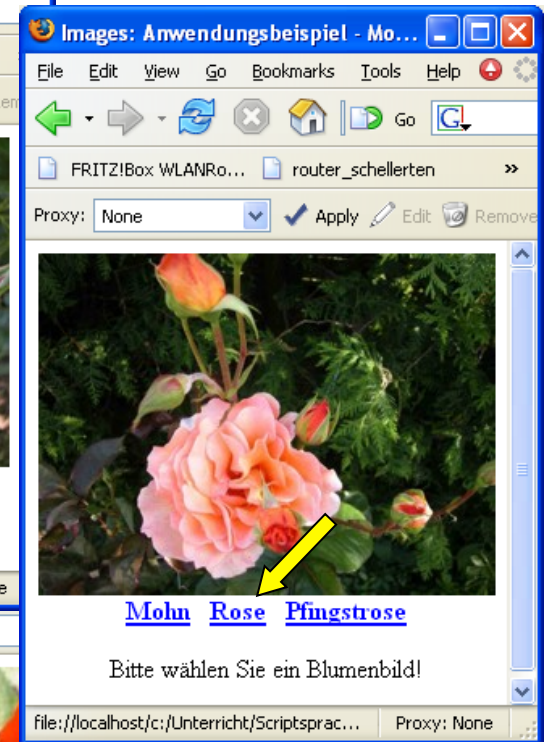
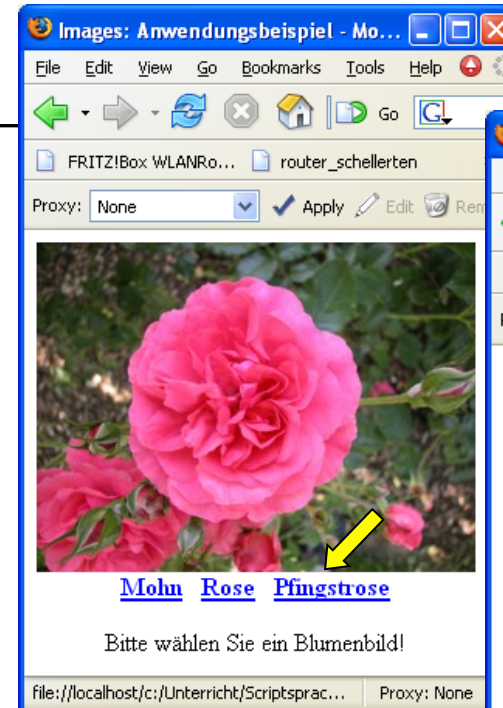
Ereignis	Auslösegrund	HTML-Tags mit dem Ereignis (Beispiele)
<i>onClick</i>	Klicken eines Elements mit der Maus	<a> <address> <area> <body> <button> <center> <col> <div> <form> <h1> <h2> ... <i> <input> <label> <link> <option> <p> <select> <strike> <table> <textarea> ...
<i>onAbort</i>	Abbruch des Ladens einer Webseite	
<i>onBlur</i>	Verlassen eines Elements	<a> <area> <button> <input> <label> <select> <textarea>
<i>onChange</i>	Änderungen von Angaben	<input> <select> <textarea>
<i>onDbClick</i>	Doppeltes Anklicken	fast alle HTML- Tags (wie <i>onClick</i>)
<i>onError</i>	Fehlerfall (z.B. falsche Bildangabe)	
<i>onFocus</i>	Aktivieren eines selektierbaren Elements	<a> <area> <button> <input> <label> <select> <textarea>

Event Handler : Ereignisse II

Ereignis	Auslösegrund	HTML-Tags mit dem Ereignis (Beispiele)
<i>onKeyDown</i>	Betätigen einer Taste	fast alle HTML- Tags
<i>onKeyPress</i>	Betätigen einer Taste	fast alle HTML- Tags
<i>onKeyUp</i>	Loslassen einer Taste	fast alle HTML- Tags
<i>onLoad</i>	Laden einer Webseite	<frameset> <body>
<i>onMouseDown</i>	Betätigen der Maustaste	fast alle HTML-Tags
<i>onMouseOut</i>	Verlassen eines Elements mit der Maustaste	fast alle HTML-Tags
<i>onMouseOver</i>	Überfahren eines Elements mit der Maus	fast alle HTML-Tags
<i>onMouseUp</i>	Loslassen der Mausetaste	fast alle HTML-Tags
<i>onReset</i>	Zurücksetzen eines Formulars	<form>
<i>onSelect</i>	Selektieren von Text in Eingabefeldern	<input> <textarea>
<i>onSubmit</i>	Absenden von Formulardaten	<form>
<i>onUnload</i>	Verlassen einer Webseite	<frameset> <body>

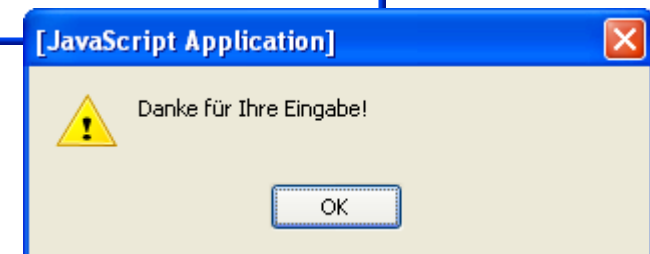
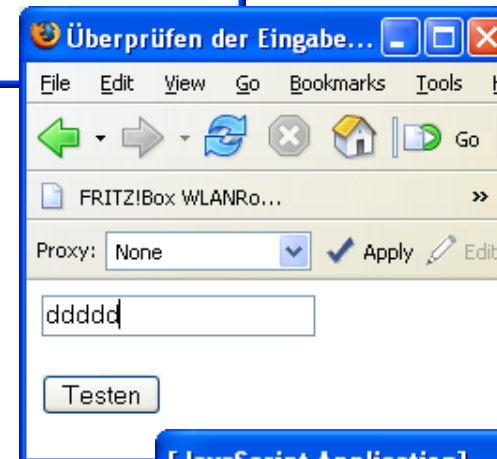
Beispiel für *onMouseOver*, *OnMouseOut*

```
<html>
<head>
  <title>Images: Anwendungsbeispiel</title>
  <script type="text/javascript">
    <!--
      function zurueck()
      {document.bild.src = "mohn.jpg";}
    //-->
  </script>
</head>
<body>
  <div align="center">
    <br>
    <a href="#" onMouseOver="document.bild.src='mohn.jpg';"
onMouseOut="zurueck();"><b>Mohn</b></a> &nbsp;
    <a href="#" onMouseOver="document.bild.src='rose.jpg';"
onMouseOut="zurueck();"><b>Rose</b></a> &nbsp;
    <a href="#" onMouseOver="document.bild.src='Pfingsrose.jpg';"
onMouseOut="zurueck();"><b>Pfingstrose</b></a>
    <p>Bitte wählen Sie ein Blumenbild!</p>
  </div>
</body>
</html>
```



Event – Handler: Beispiel für *onSubmit*

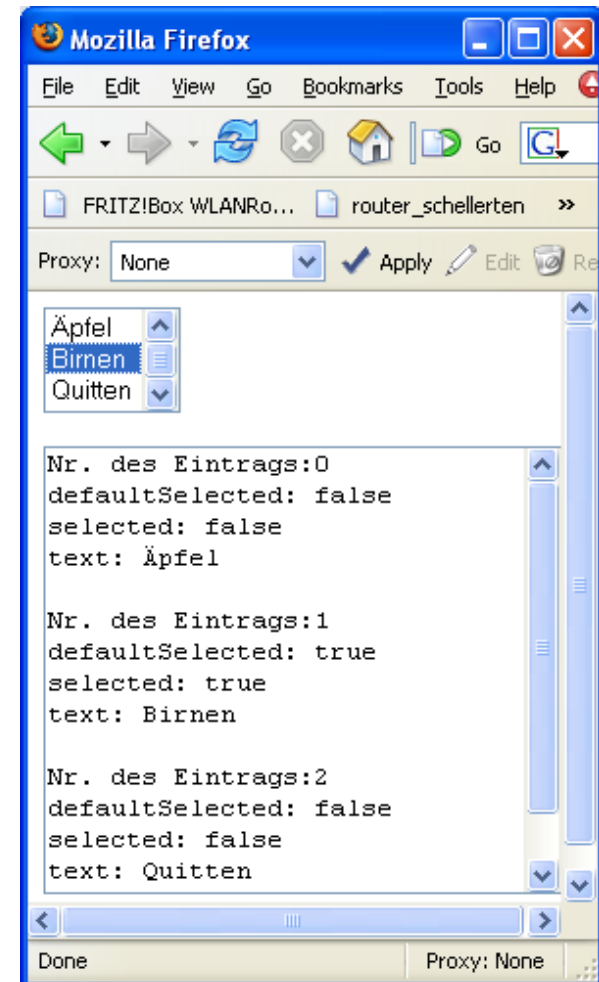
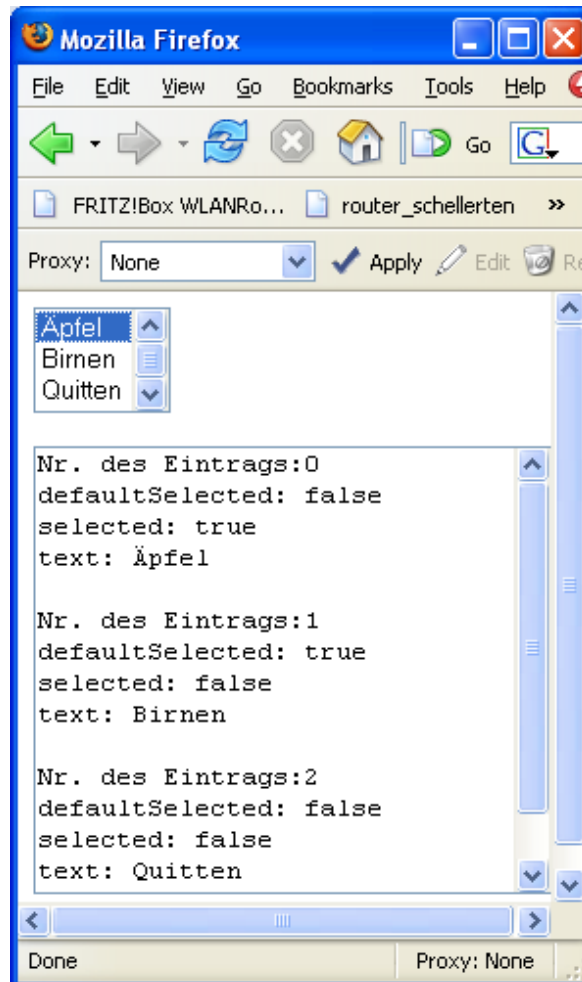
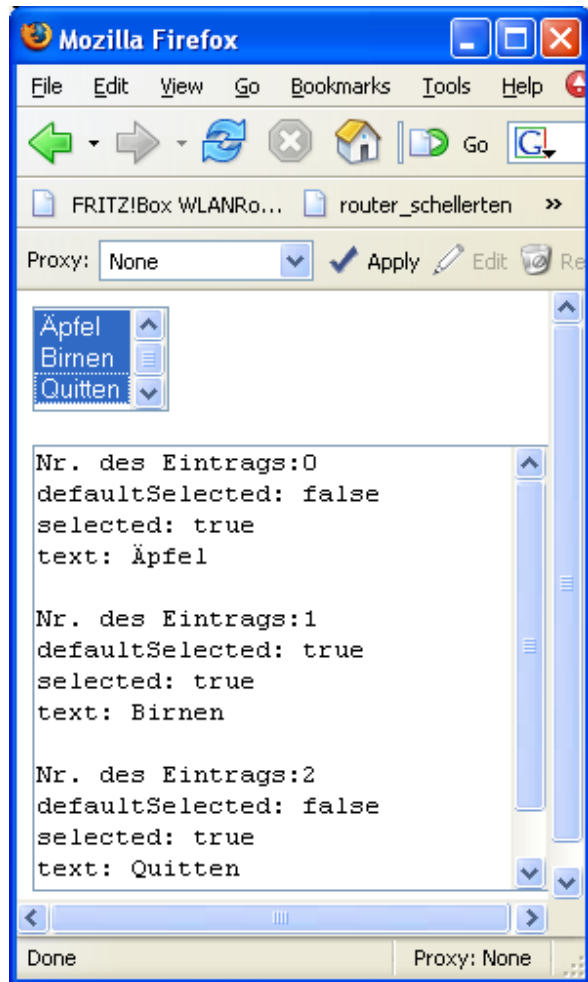
```
<html>
<head>
  <title>Überprüfen der Eingabefelder</title>
  <script type="text/javascript">
    <!--
      function testen()
      {eingabe = document.Eingabe.Feld.value;
        if (eingabe == "")
        { alert("Bitte eine Eingabe!")
          document.Eingabe.Feld.focus();
          return false;}
        else
        {alert("Danke für Ihre Eingabe!");
          return true;}}
    //-->
  </script>
</head>
<body>
  <form action="" name="Eingabe" onSubmit="return testen();">
    <input type="Text" name="Feld"> <br> <br>
    <input type="submit" name="Testsubmit" value="Testen">
  </form>
</body>
</html>
```



Event – Handler: Beispiel für *onLoad*, *onChange*

```
<html>
<script type="text/javascript">
  function durchlauf() <!-- zeigt Eigenschaften des option-Elements an ->
  { var text = "";
    for (i = 0; i < document.formular.Item.length; i++)
      {text = text + 'Nr. des Eintrags:' + document.formular.Item.options[i].index + '\n'
        + 'defaultSelected: ' + document.formular.Item.options[i].defaultSelected + '\n'
        + 'selected: ' + document.formular.Item.options[i].selected + '\n'
        + 'text: ' + document.formular.Item.options[i].text + '\n\n';}
    document.formular.ausgabe.value = text;} <!-- Zuweisen des Strings an Option value von ausgabe ->
  }
</script>
<body onLoad="durchlauf();" <!-- fkt. durchlauf() wird nach Laden der Webseite initial aufgerufen ->
  <form name="formular">
    <select name="Item" size="3" multiple onChange="durchlauf();" <!-- Aktualisierung der Anzeige ->
      <option>&Auml;pfel</option>
      <option selected>Birnen</option>
      <option>Quitten</option>
    </select></p>
    <textarea cols="30" rows="13" name="ausgabe"></textarea> <!-- mehrzeiliges Textfeld ->
  </form>
</body>
</html>
```

Event – Handler: Beispiel für *onLoad*, *onChange*



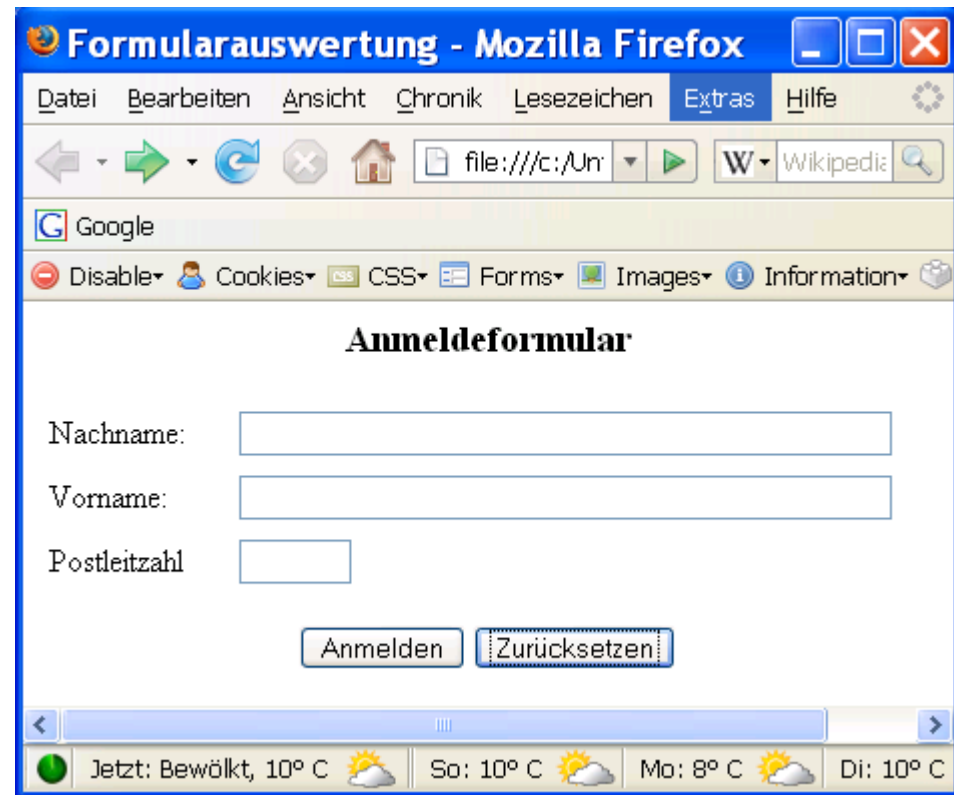
Event – Handler: Beispiel zu Formulareingaben testen

- *Häufige Anwendung: Formulareingaben auf Fehleingaben testen, bevor sie zum Server versendet werden.*
- *Nutzen des Eventhandlers „onSubmit“*

Beispiel

Es sollen die Eingabefelder eines Eingabeformulars überprüft werden:

- Nachname Zeichen
- Vorname Zeichen
- PLZ genau 5 Zahlen



The screenshot shows a Mozilla Firefox browser window with the title 'Formularauswertung - Mozilla Firefox'. The address bar shows a local file path. The page content includes a login form titled 'Anmeldeformular' with three input fields: 'Nachname:', 'Vorname:', and 'Postleitzahl:'. Below the fields are two buttons: 'Anmelden' and 'Zurücksetzen'. The status bar at the bottom shows the current weather as 'Jetzt: Bewölkt, 10° C' and forecasts for the next few days.

Event – Handler: Beispiel zu Formulareingaben testen

```
<h3 align="center">Anmeldeformular</h3>
<div align="center">
  <form name="anmeldeform" action="http://www.irgendeineUrl.de" method="get"
    onSubmit="return checkform(this)"
  <table width="500" cellspacing="0" cellpadding="5" border="0">
    <tr>
      <td>Nachname:</td>
      <td><input name="name" size="50" maxlength="70" type="text"></td>
    </tr>
    <tr>
      <td>Vorname:</td>
      <td><input name="vorname" size="50" maxlength="70" type="text"></td>
    </tr>
    <tr>
      <td>Postleitzahl</td>
      <td><input name="plz" size="5" maxlength="5" type="text" </td>
    </tr>
  </table>
  <p align="center">
    <input type="submit" value="Anmelden">
    <input type="reset" value="Zurücksetzen">
  </p>
</form>
</div>
```

Formulareingaben testen

```
function checkform(myform)
{
  if (myform.name.value == "")
  {
    alert ("Bitte geben Sie Ihren Nachnamen ein!");
    return false;
  }
  else if (myform.vorname.value == "")
  {
    alert ("Bitte geben Sie Ihren Vornamen ein!");
    return false;
  }
  else if (myform.plz.value == "")
  {
    alert("Bitte geben sie Ihre Postleitzahl an!");
    return false;
  }
  else if (!testeZeichen (myform.plz.value, "1234567890"))
  {
    alert ("Geben Sie für die Postleitzahl bitte nur Zahlen ein!");
    return false;
  }
  else if (myform.plz.value.length != 5)
  {
    alert ("Die Postleitzahl sollte fünf Stellen lang sein!");
    return false;
  }
  return confirm("Überprüfung abgeschlossen, alle Eingaben sind in Ordnung.
  \nMöchten Sie die Daten jetzt absenden?");
}
```

```
function testeZeichen (testString, erlaubteZeichen)
{
  var allezeichenok = true;
  for (var i = 0; i < testString.length ; i++)
    if (erlaubteZeichen.indexOf(testString.charAt(i)) == -1)
      allezeichenok = false;
  return allezeichenok;
}
...
```

Event – Handler setzen 1/2

- **Häufige Anwendung:** Setzen von Events (Tastaturklick oder Mouseclick) in grafischen Browserflächen

```
// Definition einer Hilfsfunktion für Eventhandler

Function attach(element, type, fn)
{
    if(element.addEventListener)
    {
        element.addEventListener(type, fn, false);
    }
    else if(element.attachEvent)
    {
        element.attachEvent('on' + type, fn);
    }
}
```

- *element* – Zeigerelement aus DOM z.B. *document*
- *type* – Eventbeschreibung, z.B. *load*
- *fn* – aufzurufende Funktion

Event – Handler setzen 2/2

- *Häufige Anwendung: setzen von Events (Tastaturklick oder Mouseclick) in grafischen Browserflächen*

```
// Tastaturabfragen für Eventhandler

attach(document, 'keypress', function(e) {
    switch(e.keyCode)
    {
        case 32:
            alert (',Leertaste');
            break;
        default:
            alert(e.keyCode);
            break;
    }
})
```

- 'keypress' – event wird im gesamten Dokument abgefragt
- alert-Meldung bei Leertaste
- Abfrage aller anderen Tasten (Hilfe für Aufgabe im Übungsblatt)

Javascript-Code auslagern

Javascript-Code kann in **separate Dateien** ausgelagert werden. Dabei handelt es sich um **Textdateien mit der Endung *.js**, welche **ausschließlich Javascript-Code** enthalten.

Syntax zum Einbinden in HTML:

```
<script src=„datei.js“ type="text/javascript">
```

- Attribut src (*src = source = Quelle*) definiert Quelldatei als URI
- Dateieigenschaften:
 - reine ASCII-Datei
 - Dateinamenerweiterung .js
 - darf nichts anderes als JavaScript-Code enthalten.

HTML5: Das Canvas-Tag

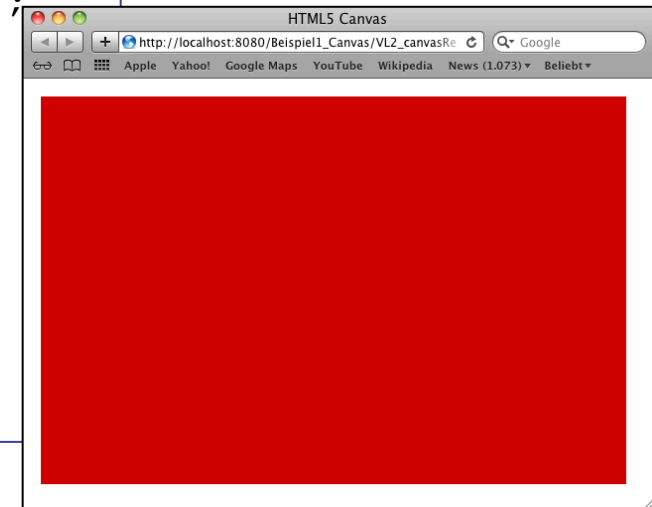
Das Canvas-Element ermöglicht die Darstellung einer graphischen Benutzeroberfläche.

```
<html>
  <head>
    <title>HTML5 Canvas</title>
  </head>
  <body>
    <canvas id="stage" width="600" height="400"></canvas>
  </body>
</html>
```

- erzeugt eine leere graphische Oberfläche im Browser mit Breite 600 Pixel und Höhe 400 Pixel
- Zugriff über id (z.B. „stage“)

HTML5: Das Canvas-Tag: Rechteck

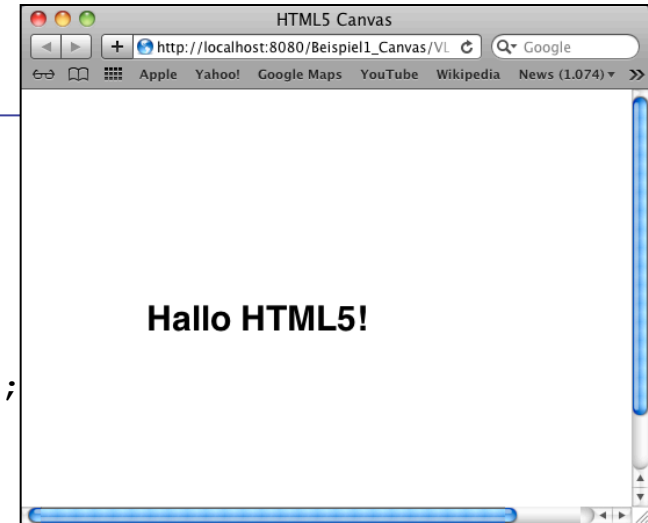
```
<html>
  <head>
    <title>HTML5 Canvas</title>
    <script type="text/javascript">
      function zeichne() {
        var canvas = document.getElementById('stage');
        if (canvas.getContext) {
          var stageContext = canvas.getContext('2d');
          stageContext.fillStyle = "rgb(200,0,0)";
          stageContext.fillRect (10, 10, 590, 390);
        }
      }
    </script>
  </head>
  <body onload="zeichne();">
    <canvas id="stage" width="600" height="400"></
canvas>
  </body>
</html>
```



- erzeugt ein rotes ausgefülltes Rechteck in der graphischen Oberfläche im Browser mit Breite 580 Pixel und Höhe 380 Pixel (Rand zur „stage“ je 10 Pixel)

HTML5: Das Canvas-Tag: Text

```
<html>
<head>
  <title>HTML5 Canvas</title>
  <script type="text/javascript">
    function zeichne(){
      var canvas = document.getElementById('stage');
      if (canvas.getContext){
        var stageContext = canvas.getContext('2d');
        stageContext.font='bold 30 px sans-serif';
        stageContext.fillText('Hallo HTML5!',100, 200);
      }
    }
  </script>
</head>
<body onload="zeichne();">
  <canvas id="stage" width="600" height="400"></canvas>
</body>
</html>
```



- erzeugt einen Text in der graphischen Oberfläche

HTML5: Das Canvas-Tag: Bildausgabe

```
<html>
<head>
  <title>HTML5 Canvas</title>
  <script type="text/javascript">
    function zeichne(){
      var canvas = document.getElementById('stage');
      if (canvas.getContext){
        var stageContext = canvas.getContext('2d');
        var imgBuffer = new Image();
        imgBuffer.src = 'scl_logo.jpg';
        imgBuffer.onload = function(){
          stageContext.drawImage(imgBuffer,50,50,180,120);
        }
      }
    }
  </script>
</head>
<body onload="zeichne();">
  <canvas id="stage" width="600" height="400"></canvas>
</body>
</html>
```



Zusammenfassung

- Einführung in JavaScript mit ausgewählten Grundlagen
- Objektmodell von JavaScript
- Zugriffstechniken auf HTML – Elemente
 - den HTML Tags sind jeweils entsprechende JavaScript – Objekte mit ihren Eigenschaften und Methoden zugeordnet
 - Beispielanwendungen für Zugriff auf HTML – Elemente mit JavaScript
- Event – Handler :
 - verschiedene Ereignisse und bei welchen HTML- Tags sie auftreten können
 - Beispielanwendungen für Zugriff auf HTML – Elemente mit JavaScript + Ereignisbehandlung (z.B. Formularauswertung)
- HTML5: Canvas

Literatur für JavaScript

- Wolfgang Dehnhardt: „Scriptsprachen für dynamische Webauftritte“, Carl Hanser Verlag München Wien, 2001
- JavaScript – Eine Einführung, RRZN Hannover, Skript
- JavaScript kurz und gut, David Flanagan, O ‘Reilly
- JavaScript Kochbuch für Web- Anwendungen, Jerry Bradenbaugh, O ‘Reilly
- JavaScript. Einführung, Programmierung und Referenz, Stefan Koch, dpunkt Verlag
- <http://de.selfhtml.org/javascript/sprache/index.htm>
- <http://www.javascript-workshop.de/buch/03.html>
- <http://www.peterkropff.de/site/javascript/prototypen.htm>