

E3. Statische und dynamische Semantik am Beispiel PHP

E3.1 Statische Bindung von Namen

Namen im Programmtext werden an **Objekte der Programmausführung** gebunden:

Variablenname an Variable mit Speicherstelle
 Funktionsname an definierte Funktion

Gültigkeitsbereich einer Bindung:

der Bereich des Programms, in dem ein Name *n* an das Programmobjekt *o* gebunden ist.

Außerhalb des Gültigkeitsbereiches ist *n* nicht an *o* sondern an ein anderes Programmobjekt oder gar nicht gebunden.

Sprachen haben unterschiedliche **Gültigkeitsregeln** (engl.: **scope rules**).

Eine **Bindung** entsteht

- **explizit durch eine Definition**, z. B. in PHP: Funktionen und formale Parameter; in C, C++, Java, u.v.a.m.: Variablen, Funktionen, Typen, Parameter, Marken etc.
- **implizit durch Benutzung des Namens**: z. B. in PHP: Variable

Explizite Definitionen ordnen dem Programmobjekt **statische Eigenschaften** zu: z. B. in PHP:

Funktion: Anzahl und Übergabeart der Parameter, Funktionsrumpf
 Parameter: Übergabeart
 in statisch typisierten Sprachen:
 Variable: Typ ihrer Werte

```
{ float x; int i; x = 3.1; i = 0; }
```

Gültigkeitsbereiche in PHP-Programmen

Gültigkeitsregeln in PHP: Art des Namens	Gültigkeitsbereich
Funktion	ganzes Programm
Parameter	sein Funktionsrumpf
lokale Variable	ihr Funktionsrumpf
globale Variable <code>\$x</code>	ganzes Programm außer Funktionsrumpfe ohne <code>global \$x</code>

Beispiel für Gültigkeitsbereiche:

```
function namesOut ($v) {
    global $out, $lineCnt;
    $lg = strlen ($v);
    fputs($out, $v);
    $lineCnt++;
}
$out = fopen ("names", "w");
$lineCnt = 0;
$sum = 0;
$lg = 5;
while ( ... ) {
    namesOut (...);
}
print $lineCnt;
fclose ($out);
```



E3.2 Lebensdauer von Variablen

Lebensdauer: Begriff der **dynamischen Semantik**
 Zeit, während der eine **Variable im Speicher existiert**;
 sie wird ausgedrückt in Bezug auf die **Ausführung von Programmabschnitten**

Art von PHP-Variablen

globale Variable
 lokale Variable und Parameter einer Funktion

Lebensdauer

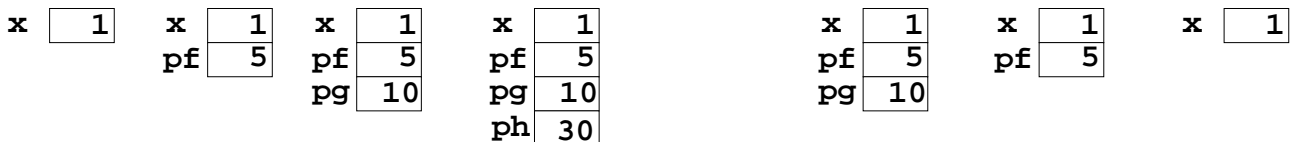
gesamte Ausführung des Programms
 Ausführung eines Aufrufes der Funktion

Beispielprogramm:

```
function ff ($pf) {gg(2*$pf); print $pf . "\n";}
function gg ($pg) {hh(3*$pg); print $pg . "\n";}
function hh ($ph) {print $ph . "\n";}
$x = 1; ff (5); print $x . "\n";
```

ausgeführte Aufrufe:

```
ff(5);    gg(10);    hh(30); print $ph;    print $pg;    print $pf; print $x;
```



Variablen im Speicher und ihre Lebensdauer

Rekursive Funktionsaufrufe

Rekursiv: auf sich selbst bezogen

Rekursive Funktion F:

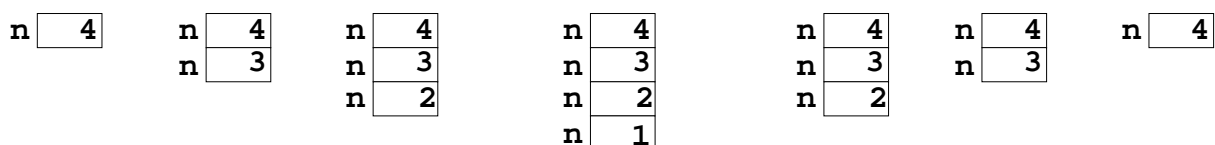
Der Rumpf von *F* enthält einen Aufruf von *F* oder von einer anderen Funktion, die *F* direkt oder indirekt aufruft.

Beispiel:

```
function fak ($n) {
    if ($n <= 1) return 1;
    else return $n * fak ($n - 1);
}
print fak (4). "\n";
```

ausgeführte Aufrufe:

```
fak(4)    fak(3)    fak(2)    fak(1) return 1;    return 2;    return 6;    return 24;
```



Variablen im Speicher und ihre Lebensdauer

E3.3 Dynamische Semantik von Aufrufen

Der **Aufruf einer Funktion** führt die definierte Berechnung aus mit den (aktuellen) **Parameterwerten**, die beim Aufruf angegeben sind. (siehe EWS-4.23)

Aufrufe haben die **Form**: `FuncExpr ([Parameters])`

Ein **Aufruf wird in folgenden Schritten ausgeführt**:

0. **FuncExpr auswerten** (ist meist nur ein Name), liefert eine Funktion.
1. **Aktuelle Parameter** an der Aufrufstelle **auswerten**.
2. **Speicher** für formale Parameter und lokale Variablen **anlegen**.
3. Die **Werte** der **aktuellen** Parameter an die **formalen** Parameter **zuweisen** (bei Übergabeart *call-by-value*).
4. **Anweisungen aus dem Funktionsrumpf** im Speicher des Aufrufes ausführen.
5. **Speicher** (aus Schritt 2) **freigeben**; **Ergebnis** an die Aufrufstelle **zurückgeben**.

Verschieden **Arten der Parameterübergabe**: (siehe EWS-4.38)

Call-by-value:	Der formale Parameter ist eine Variable, die mit dem Wert des aktuellen Parameters initialisiert wird. in PHP, C, C++, Java, Ada, Pascal, u.v.a.m.
Call-by-reference:	in PHP, C++, Pascal
Call-by-value-and-result:	in Ada

S4 JavaScript

Skriptsprache, 1995 bei Netscape als *LiveScript* entwickelt dann in *JavaScript* (unpassend) umbenannt. Standard ECMA 262 (1996) fasst JavaScript (Netscape) und JScript (Microsoft) zusammen

- abgeleitet von Perl; Notation wie C, C++, Java, PHP; sonst kein Bezug zu Java
- **interpretiert, dynamisch typisiert**
- spezielle **objektorientierte** Eigenschaften
- eingebettet in HTML
- Interpretierer **in Web-Browser integriert** (Netscape, Internet Explorer)
- Zugriff auf Elemente des dargestellten Dokumentes (DOM)

Anwendungszwecke:

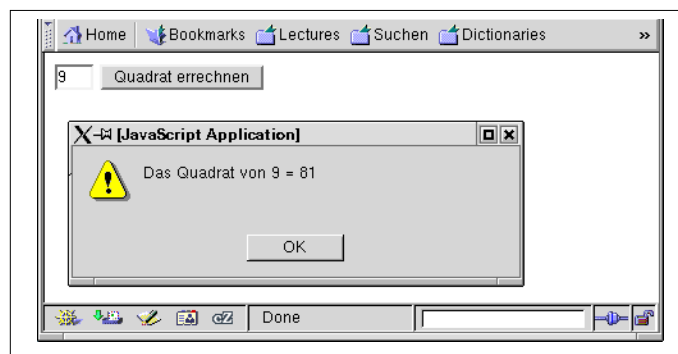
- Programme, die im Web-Browser des **Client** ausgeführt werden
- **Bedienoberflächen** in dynamischen Web-Seiten
- **Reaktionen auf Ereignisse** bei der Interaktion mit Web-Seiten
- Formular-Elemente **dynamisch erzeugen**, Eingabe **prüfen**
- **Animationseffekte**

Ein erstes Beispiel in JavaScript

```
<html><head>
  <title>Quadrat</title>
<script type="text/javascript">
  function Quadrat() {
    var Zahl = document.Formular.Eingabe.value;
    var Ergebnis = Zahl * Zahl;
    alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
  }
</script>
</head><body>
  <form name="Formular" action="">
    <input type="text" name="Eingabe" size="3">
    <input type="button" value="Quadrat errechnen"
      onClick="Quadrat()">
  </form>
</body></html>
```

Funktionsdefinition
eingebettet mit `script`-
Tags

Aufruf der
Funktion bei
`onClick`-Ereignis



Anzeige im
Browser

S3.1 Einbettung in HTML `script`-Tags

Programmfragmente in JavaScript können auf unterschiedliche Arten in HTML-Dateien eingebettet werden. Wir betrachten hier 3 von 4 Arten:

1. Mit `script`-Tags geklammerte Programmfragmente:

```
<script type="text/javascript">
function makeTextElem (name) {
  return "<input type=\"text\" \" +
    \" name=\"\" + name +
    \"\" size=10>";
}
</script>
```

```
<script type="text/javascript">
document.writeln
  (makeTextElem ("Zuname"));
</script>
```

Solche Programmfragmente werden **ausgeführt und die Ausgabe**, die sie erzeugen wird an ihrer Stelle in die HTML-Datei **eingesetzt** und angezeigt (wie in PHP).

Funktionsdefinitionen erzeugen keine Ausgabe;
man bettet sie sinnvoll in dem `head`-Teil ein.
(siehe vorige Folie)

Einbettung: Werte spezieller HTML-Attribute

2. Einige **HTML-Attribute benennen Ereignisse**, die der Bediener auslösen kann, z. B. einen Knopf betätigen.

Als **Wert des Attributes** wird eine **Anweisungsfolge** angegeben. Sie wird ausgeführt, wenn das Ereignis eintritt.

```
<input type="button" value="Quadrat errechnen"
      onClick="Quadrat()">
```

3. In einem **Anker-Element** kann man statt einer URL auch eine **Anweisungsfolge** mit vorangestelltem **javascript:** angeben. Beim Klicken darauf wird sie ausgeführt.

```
<a href="javascript: alert ('habt Geduld');">Musterlösung</a>
```

Meist werden an solchen Stellen anderweitig definierte **Funktionen aufgerufen**.



S3.2 Spracheigenschaften Notation

Die Notation stimmt in Vielem mit der von PHP, vom Kern von C, C++ und Java überein.

einige wichtige **Unterschiede**:

Bezeichner, Wortsymbole:

Groß- und Kleinschreibung ist unterscheidend (case-sensitive):

headOut verschieden von headout

true und false aber nicht True oder False

Bezeichner:

einheitliche Schreibweise für alle Arten von Bezeichnern:

(Buchstabe | \$ | _) (Buchstabe | \$ | _ | Ziffer)*

Anweisungen:

abschließendes ; kann am Zeilenende entfallen

mit ;

```
lineCount = 1;
sum = 0;
while (lineCount < 100) {
    ...
}
```

ohne ;

```
lineCount = 1
sum = 0
while (lineCount < 100) {
    ...
}
```

Variable und Zuweisungen

Variable und Zuweisungen haben die **gleiche Bedeutung wie in PHP** (siehe EWS-4.5).

- **Variablenamen** brauchen nicht durch ein $\$$ -Zeichen gekennzeichnet zu werden.
- Eine Variable kann (wie in PHP) **Werte beliebiger Typen** annehmen.
- Es wird (wie in PHP) unterschieden zwischen
globalen Variable x : gilt im ganzen Programm, außer in Funktionen mit einem lokalen x
lokale Variable: gilt in der Funktion, in der sie **definiert** ist
- Es gibt **Definitionen für Variable**.
Lokale Variable muss man, **globale kann** man definieren.
Wird eine Variable in einer Funktion benutzt aber nicht definiert, so ist sie global.

In einer **Variablendefinition** können eine oder mehrere Variable definiert werden, sie können auch durch **Zuweisung eines Anfangswertes initialisiert** werden, er muss durch ein **Literal oder eine andere Variable** angegeben werden.

```
var line;
var sum, result;

var col, count = 3, row;
var minimum = count,
    maximum = 999;
```

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

n lokal
sum lokal
col global

Datentypen

number:

numerische Werte, ganze Zahlen und Gleitpunktzahlen zusammengefasst,
spezieller Wert für undefinierte Werte **NaN** (not a number)

Literale wie in PHP

arithmetische Operatoren wie in PHP

string:

Zeichenreihen

Literale: Klammerung mit " oder ' ist gleichbedeutend

'Er sagt "Hallo!'" "Sag's auch!"

Umschreibungen nicht-druckbarer Zeichen wie in PHP \n, \t, usw.

In Zeichenreihen werden **Variablenwerte nicht eingesetzt** (anders als in PHP)

Operatoren: Kontatation +: "Er heißt " + name

string-Funktionen in Objekt-Notation (wird nicht hier erklärt)

boolean:

Wahrheitswerte

Literale: false, true

Operatoren: Konjunktion &&, Disjunktion ||, Negation ! (wie in PHP)

Undefined: einziger Wert **undefined**, Ergebnis bei Zugriff auf nicht-zugewiesene Variable

Objekte inklusive Arrays

Funktionen als Werte

Konversion

Alle Konversionen implizit (coercion), so wie der Kontext es erfordert:

in **boolean**:

von **number**: 0, NaN -> **false**, sonst -> **true**

von **string**: "" -> **false**, sonst -> **true**

von **undefined**: **false**

in **number**:

von **boolean**: **false** -> 0, **true** -> 1

von **string**: ganze Zeichenreihe (ohne Leerzeichen am Anfang und Ende) wie ein numerisches Literal (ggf. mit Vorzeichen) lesen und konvertieren;

leere Zeichenreihe (oder nur Leerzeichen) -> 0;

sonst **NaN**

von **undefined**: **NaN**

in **string**:

von **boolean**: **false** -> "**false**", **true** -> "**true**"

von **number**: Zahlwert als Zeichenreihe, wie Literal (ggf. mit Vorzeichen)

von **undefined**: "**undefined**"

Operatoren

Präzedenz (steigend)	Stelligkeit	Assoziativität	Operatoren	Erklärung
1	2	rechts	= += -= usw.	Zuweisungsoperatoren
2	3	links	?:	bedingter Ausdruck
3	2	links		log. Disjunktion (oder)
4	2	links	&&	log. Konjunktion (und)
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 (log., arithm., bitw.)
	1		++ --	Inkrement, Dekrement
	1		typeof void	Typ?; nach undefined konv.
14	1		() [] .	Aufruf, Index, Objektzugriff

Ablaufstrukturen

Anweisungsfolge: wie in PHP

```
{ st = st + "*"; i = i + 1; }
{ var k = 42; document.writeln (5*k); }
```

Durch `var` eingeführte Bindung gilt nicht nur in der Anweisungsfolge, sondern in umgebender Funktion bzw. im umgebenden Programm.

Bedingte Anweisung: wie in PHP

```
if (a < 0) {a = b;}      if (a < b) {min = a;} else {min = b;}
Bei einzelnen Anweisungen sind die {}-Klammern nicht nötig aber sinnvoll.
```

while-Schleife: wie in PHP

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

for-Schleife: wie in PHP

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

Funktionsaufrufe: wie in PHP, aber nur call-by-value als Parameterübergabe

```
headOut ("Test")      document.write ("*")
```

return-Anweisung: wie in PHP

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

Funktionen

Funktionsdefinitionen: wie in PHP

```
function Ueberschrift (grad, text) {
  var marke = "h"+grad;
  document.writeln ("<"+marke+">"+text+"</"+marke+">");
}
```

`grad` und `text` sind formale Parameter, `grad`, `text` und `marke` sind lokale Variable.

```
function fak (n) {
  if (n<=1) return 1; else return n * fak (n-1);
}
```

Funktionen können im `head`- oder im `body`-Teil der HTML-Datei definiert werden. Aufrufe können in jedem JavaScript-Fragment stehen.

Funktionen als Werte:

Funktionen kann man als Werte notieren. **Literal** für eine Funktion ohne Namen:

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

Solche **Funktionsliterals** kann man in Ausdrücken verwenden, z. B. einer **Variablen zuweisen**

```
var add = function (a, b) { return a + b; };
```

und den Wert der Variablen (die Funktion) aufrufen: `x = add (42*k, 3);`

Arrays

Ein **Array** ist eine **Abbildung** von Indizes (oder Schlüsseln) auf Werte (wie in PHP):
 Jedes **Element eines Arrays** ist ein **Paar aus Index** und zugeordnetem **Wert**;
 erste Komponente der Paare: **numerischer Index oder ein string als Schlüssel**.
 In JavaScript sind **Arrays spezielle Objekte** (wird nicht hier erklärt).

Arrays kann man auf verschiedene Weise **erzeugen**:

Liste von Werten:	<code>monatsName = new Array("", "Jan", ..., "Dez");</code> indiziert von 0 an, erster Wert ist hier irrelevant
leeres Array erweitern:	<code>monatsName = new Array();</code> <code>monatsName[1]= "Jan"; monatsName[2]= "Feb";..</code>
auch	<code>monatsNr = new Array();</code> <code>monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; ...</code>

Zugriff auf die Werte von Array-Elementen durch

Indizierung wie in PHP:	<code>monatsName[4]</code> oder <code>monatsNr["Apr"]</code>
Objektselektion:	<code>monatsNr.Apr</code>

Schleife zum Aufzählen aller Elemente eines Arrays (ähnlich wie in PHP):

```
for (key in arr) { ... }    arr muss ein Array sein;
mit key wird im Rumpf auf den Schlüssel eines Elementes zugegriffen

for (mname in monatsNr)
{ document.writeln (mname + "=>" + monatsNr[mname]); }
```

S3.3 Objekte

Ein Grundkonzept von JavaScript ist der **Objektbegriff**. Er wird hier nur soweit eingeführt, dass die notwendigen Notationen verstanden werden.

Das aktuelle **Fenster** und das **Dokument** sind auch als Objekte verfügbar.

Ein Objekt wird **im Speicher** erzeugt und durch seine **Speicherstelle** eindeutig **identifiziert**.

```
student = {name:"E. Mustermann", matrNr:999999};
erzeugt ein Objekt und weist seine Speicherstelle der Variablen zu.
```

Ein Objekt **besteht aus Komponenten**.

Sie haben jeweils einen **Namen** und einen **Wert** - wie Variable.

Das obige Objekt hat Komponenten mit Namen `name`, `matrNr`, usw.

Objektcomponenten werden durch **Objekt-Ausdruck.Name** zugegriffen.

```
student.name liefert "E. Mustermann"
```

Arrays und **Zeichenreihen** sind auch Objekte.

Array-Objekte haben auch eine Komponente `length`:

```
monatsName.length liefert 13 (den größten numerischen Schlüssel + 1, also 12+1)
```

Einige der **Komponenten können auch Funktionen** sein; sie heißen dann **Methoden**.

Ihre Aufrufe können die übrigen **Komponenten des Objektes lesen oder verändern**:

```
monatsName.reverse()    monatsName.sort()    document.writeln()
```

Funktionen auf Zeichenreihen-Objekten

Zeichenreihen sind Objekte in JavaScript. **Aufrufe** von Funktionen (Methoden) auf Zeichenreihen werden in **Objekt-Notation** geschrieben, z. B.

```
var Aussage = "Der Mensch ist des Menschen Feind";
var Suche = Aussage.indexOf("Mensch");
```

Die Funktion **indexOf** wird für die Zeichenreihe der Variablen **Aussage** mit dem Parameter **"Mensch"** aufgerufen. In PHP hätten wir in **Funktions-Notation** stattdessen geschrieben:

```
$Suche = strpos ($Aussage, "Mensch");
```

Diese Aufrufe **ändern die Zeichenreihe nicht**, auf die sie angewandt werden:

```
Aussage.toLowerCase();
```

liefert eine neue Zeichenreihe: alle Großbuchstaben durch Kleinbuchstaben ersetzt.

Weitere **string-Funktionen**:

charAt(i)	Zeichen an Position i liefern
replace(r, s)	Auftreten des regulären Ausdruck r suchen und ersetzen durch s
search(r)	suchen mit regulärem Ausdruck r
substr(p,l)	Teilzeichenreihe ab Position p der Länge l liefern

Einige **string-Funktionen** erleichtern die **Auszeichnung in HTML**:

```
Inh = "Inhalt"; document.write(Inh.anchor("IH"));
```

gibt einen Anker in HTML aus: `Inhalt`

Zugriff auf Elemente des Dokumentes

Aus dem JavaScript-Programm kann man auf **Elemente des Dokumentes zugreifen**, das der Browser anzeigt. Das ist meist die HTML-Datei, in die das Programm eingebettet ist. Damit kann man z. B. den **Inhalt von Formular-Elementen** prüfen:

```
<script type="text/javascript">
  function Kontrolle() {
    var Zahl = document.QuadratForm.Eingabe.value;
    alert ("Eingabe war " + Zahl);
  }
</script>
<form name="QuadratForm" action="">
  <input type="text" name="Eingabe" size="3">
  <input type="button" value="Quadrat errechnen"
    onClick="Kontrolle()">
</form>
```

Hier wird die Zugriffsstruktur angewandt:

```
document.FormularName.EingabeElementName.AttributName
```

Alternativ kann man die Formulare im Dokument und ihre Elemente jeweils indizieren:

```
document.forms[i].elements[j].AttributName
```

also für obiges Beispiel:

```
var zahl = document.forms[0].elements[0].value;
```

Document Object Model (DOM)

Das **Document Object Model (DOM)** regelt, welche Informationen ein Browser zur Client-seitigen Programmierung, z. B. in JavaScript, verfügbar macht:

- Eigenschaften des gerade **angezeigten Dokumentes**, wie

<code>document.title</code>	Titel-string
<code>document.forms[i]</code>	Formulare, durchnummeriert von 0 an
<code>document.images[i]</code>	Bilder, durchnummeriert von 0 an
- Methoden für das gerade **angezeigte Dokument**, wie

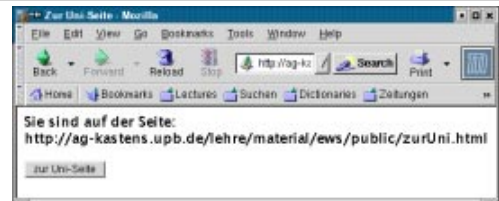
<code>document.write(string, ...)</code>	Ausgabe von Zeichenreihen
<code>document.writeln(string, ...)</code>	ebenso mit abschließendem Zeilenwechsel
- Eigenschaften der vom Browser **angezeigten URL**, wie

<code>location.href</code>	die gesamte URL
----------------------------	-----------------
- Methoden für die vom Browser **angezeigte URL**, wie

<code>location.reload()</code>	erneut laden
<code>location.replace(url)</code>	ein anderes Dokument anzeigen

Beispiel: Seitenwechsel

```
<p>Sie sind auf der Seite:<br>
<script type="text/javascript">
  document.write(location.href);
</script>
<form name="UniURL" action="">
<input type="button" value="zur Uni-Seite"
  onClick='location.replace("http://www.upb.de")'>
</form>
```



S3.4 Ereignisbehandlung

Ereignis in der Informatik (engl. event): Wahrnehmung einer Zustandsänderung.

Ereignis-getriebene Programmierung: Ereignissen werden Operationen zugeordnet (Ereignisbehandler); sie werden **bei Eintreten des Ereignisses ausgelöst**; typisch für Regelung und Steuerung realer Prozesse, Programmierung von Bedienoberflächen, z. B.

Ereignis	Ereignisbehandlung
Temperatur überschreitet 90 C	Kühlung anschalten
Temperatur unterschreitet 80 C	Kühlung ausschalten
Knopf wird gedrückt	Formular abschicken
Mauszeiger ist über dem Bild	Bildüberschrift blinken lassen

Behandlung des Ereignisses **Click** als Attribut von Formular-Elementen:

```
<form name="testForm">
  <input type="button" value="ping"
    onclick='alert("ping!");'>
  <input type="button" name="oKnopf" value="pong">
</form>
<script type="text/javascript">
  document.testForm.oKnopf.onclick=
    function(){alert("pong!");};
</script>
```

Operation als Attributwert im HTML-Tag zugeordnet

Funktion als Attributwert zugewiesen

Wichtige Ereignisse

Ereignisbehandler	HTML-Elemente	Bedeutung
onclick	Knopf, Checkbox, Anker	Element wird angeklickt
onchange	Textfeld, Textbereich, Auswahl	Wert wird geändert
onkeydown onkeyup onkeypress	Dokument, Bild, Anker, Textfeld	Taste gedrückt, losgelassen
onmousedown onmouseup	Dokument, Knopf, Anker	Maustaste gedrückt, losgelassen
onmouseout	Bereiche, Anker	Mauszeiger verlässt einen Bereich
onmouseover	Anker	Mauszeiger über Anker
onreset, onsubmit	Formular	Reset, Submit für ein Formular
onselect	Textfeld, Textbereich	Element wird ausgewählt
onfocus onblur	Fenster, alle Formular-Elemente	Eingabefokus wird dem Element gegeben, entzogen

Beispiel: Bildtausch

Für ein `img`-Element werden die Ereignisse `onmouseover` und `onmouseout` benutzt, um das Bild auszutauschen:

```
<html><head>
  <title>Bildtausch</title>
  <script type="text/javascript">
    function enter () { document.ews.src="ewsEin.jpg"; }
    function leave () { document.ews.src="ewsAus.jpg"; }
  </script>
</head><body>
  
</body></html>
```

Bild wechselt zwischen



und



Unterschiede: Netscape Navigator und Internet Explorer

Leider sind Eigenschaften des Ereignismodells im Netscape Navigator und Internet Explorer unterschiedlich realisiert. Man muss auf sie in **separaten Programmzweigen** zugreifen.

```

<html><head>
  <title>Navigator vs. Explorer</title>
  <script type="text/javascript">
    function coord(e) {
      var isNavigator =
        navigator.appName.indexOf("Netscape") != -1;
      var x = isNavigator ? e.pageX : event.clientX;
      var y = isNavigator ? e.pageY : event.clientY;
      alert("Coordinate = (" + x + ", " + y + ")");
    }
  </script>
</head>
<body>
  <a name="hier">
  <a href="#hier" onclick="coord(event);">
    
  </a>
</body>
</html>

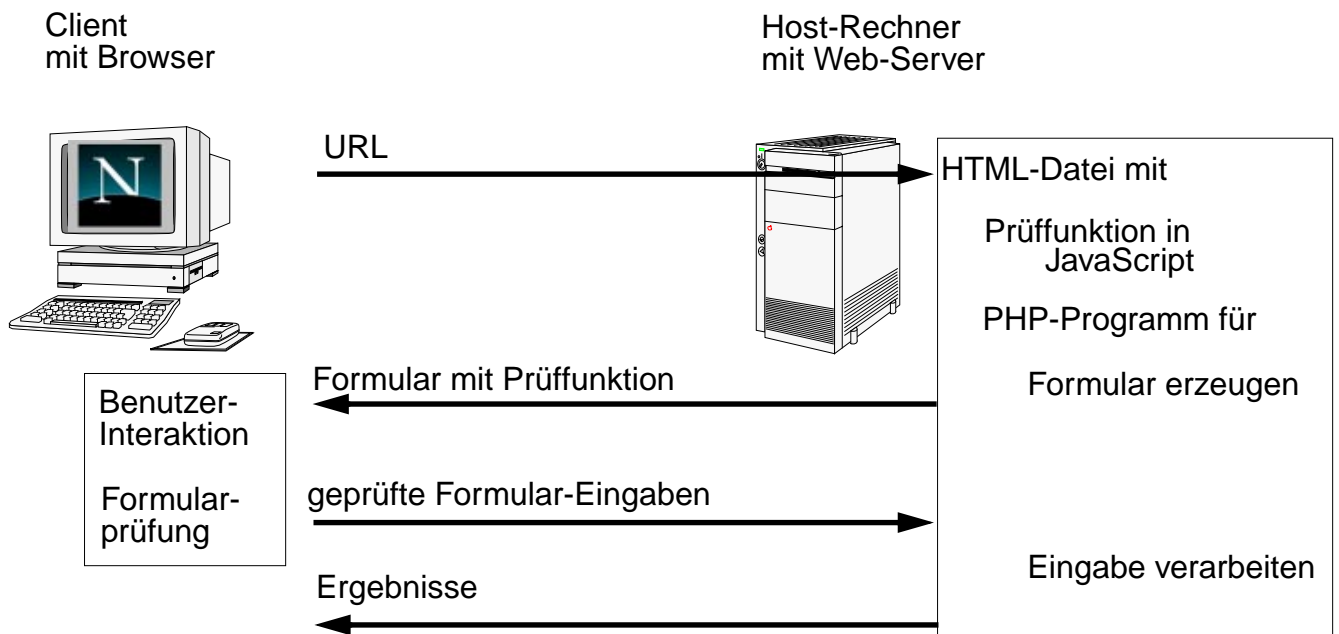
```

Unterscheidung der Browser

gerade eingetretenes Ereignis in der Variablen **event**:
Netscape: lokal im Ereignisbehandler
Explorer: global

Koordinaten der Stelle, wo ein Ereignis e eingetreten ist:
Netscape: e.pageX, e.pageY
Explorer: e.clientX, e.clientY

Interaktion zwischen Client und Server



Beispiel mit Formularprüfung

```
<html><head><title>Geprüfte Formular-Eingabe</title>
<script type="text/javascript">
  function check () {
    if (document.SpendenForm.Zuname.value.length < 2)
      { alert ("Zuname zu kurz!"); return false; }
    var betrag = document.SpendenForm.Spende.value;
    if (betrag <= 0) { alert ("Betrag angeben!"); return false; }
    if (betrag > 1000) { return confirm ("Höhe der Spende: " + betrag); }
    return true;
  }
</script>
</head><body>
<?php
  if (!isset($_REQUEST['submit'])) {
echo <<<FORMULARANZEIGE
  <form name="SpendenForm"
    action="http://ag-kastens.upb.de/..." method="POST">
    <p>Zuname:<br><input type="text" name="Zuname" size="10"></p>
    <p>Höhe der Spende:<br>
      <input type="text" name="Spende" size="10"></p>
    <input type="reset" value="löschen"><br>
    <input type="submit" value="abschicken" name="submit"
      onclick="return check();"><br>
  </form>
FORMULARANZEIGE;
  } else {
    echo "<h4>Vielen Dank für Ihre Spende:</h4><p>\n<pre>";
    foreach ($_REQUEST as $name => $value) { echo "$name => $value\n";}
    echo "</pre>";
  }
?></body></html>
```