



JavaScript

Grundlagen der Programmierung



Cegos Group

inspire
qualify
change



Inhalts- verzeichnis



Übersicht



Weiterführende Themen



Grundlegende Sprachelemente



ECMA2016



Objektorientierte Programmierung



UI-Programmierung






Beispielanwendungen



Übersicht



-  Technologien für Browser-Anwendungen
-  Einordnung von JavaScript
-  Runtime



Technologien für Browser- Anwendungen



Einleitung

- Was Sie lernen werden:
 - Sämtliche grundlegenden Techniken des Arbeitens mit JavaScript
 - IDE
 - Debugger
 - Die Beispiele haben einen möglichst großen Praxisbezug, so dass diese auch sofort in eigenen Anwendungen umgesetzt werden können
 - Kompatibilitätsprobleme und deren Lösung
- Was Sie nicht lernen werden
 - Komplexe AJAX-Anwendungen
 - Die Benutzung reichhaltiger Bibliotheken wie YUI, DOJO oder jQuery



Vom statischen Web zu Rich Internet Applications

- Die ersten Browser stellten Informationen rein statisch dar
 - Rendern von HTML-Seiten
 - Verlinkung
- Der „klassische“ Browser bietet eine „Thin Client“-Plattform:
 - HTML-Formulare zur Erfassung von Daten
 - Die Gestaltung der Seiten erfolgt mit Cascading Style Sheets



Vom statischen Web zu Rich Internet Applications II

- Diese Ausprägung war jedoch für viele Client-Anwendungen zu beschränkt
- Deshalb wurden für den Browser verschiedene Ansätze für die Integration von echter Programmlogik entwickelt:
 - Java Applets,
 - Adobe Flash,
 - Microsoft Silverlight,
 - JavaFX
 - JavaScript
 - ...
- Damit wird der Browser zu einer echten Rich Client Platform für Rich Internet Applications.
- Wie das folgende Kapitel zeigt ist von den oben gegebenen Varianten JavaScript zum de facto Standard geworden.



Sind Client-seitige Skripte notwendig?

- Vorteile:
 - Der Server wird entlastet, weil Berechnungen, Modifikationen der Web-Seite etc. auf dem Client ausgeführt werden
 - Die Ausführung der Programme erfolgt schneller, da keine Daten zum Server und wieder zurück übertragen werden müssen
 - Es muss während der Ausführung des Programms keine Verbindung zum Internet bestehen
 - Einfacher zu Testen für den Entwickler



Einordnung von JavaScript



Geschichte von JavaScript

- Erste Veröffentlichung Ende 1995 von Netscape unter dem Namen LiveScript
 - Erweiterung von HTML um den Befehl `<script> ... </script>`
- JavaScript wurde in rascher Folge um neue Funktionalitäten erweitert
 - Insbesondere wurde ein Event-Modell eingeführt
- Die weiteren Versionen 1.3, 1.4 und 1.5 brachten nur kleine Neuerungen, die Details wurden geringfügig verbessert.
- Einführung der ECMAScript-Spezifikation



ECMA-Script

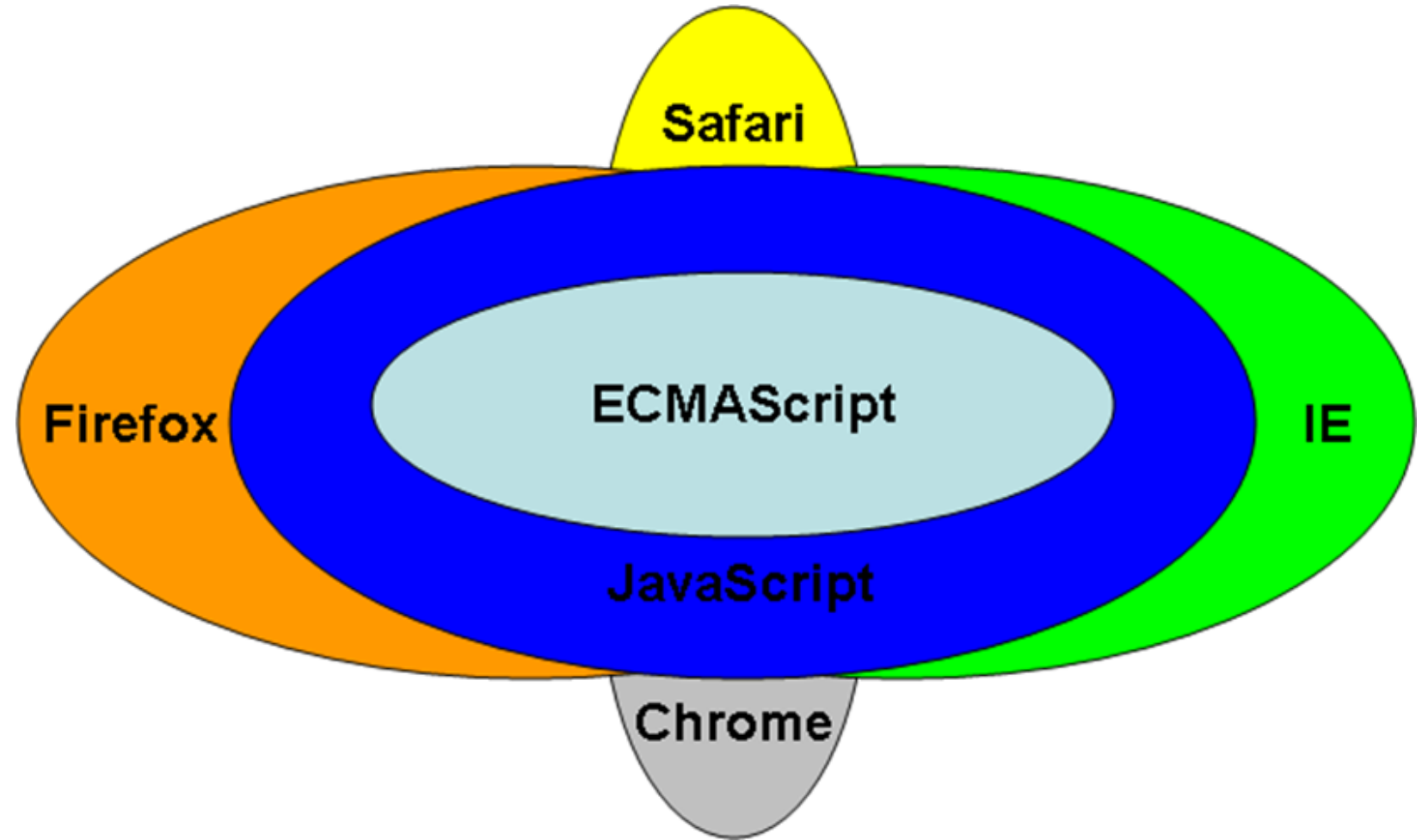
- JavaScript ist unter dem Namen ECMA-Script 262 seit Dezember 1999 in der Version 3.0 standardisiert worden. Sie können sich entweder auf der Webseite <http://www.ecma.ch/> über den aktuellen Stand informieren oder per Mail die entsprechende CD anfordern.
- Die ECMA versucht die Kernelemente von JavaScript zu standardisieren, so dass die z. Z. beide großen Browserhersteller vermelden, konform zum ECMA-Standard zu sein.
- Die aktuelle Diskussion können Sie unter <http://www.webstandards.org/> verfolgen.



Runtime



Umfang



Browser- Unterstützung (aus wikipedia.org)

Version	Release date	Equivalent to	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	03 1996			3.0			
1.1	08 1996						
1.2	06 1997						
1.3	10 1998	ECMA-262 1 st edition / ECMA-262 2 nd edition		4.0			
1.4							
1.5	11 2000	ECMA-262 3 rd edition	1.0	5.5=JScript 5.5, 6 = JScript 5.6, 7 = JScript 5.7, 8 =JScript 6	6.0, 7.0, 8.0, 9.0		

Browser- Unterstützung (aus wikipedia.org)

Version	Release date	Equivalent to	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.6	11 2005	1.5 + Array extras + Array and String generics + E4X	1.5			3.0, 3.1	
1.7	10 2006	1.6 + Pythonic generators + Iterators + let	2.0			3.2, 4.0	1.0
1.8	06 2008	1.7 + Generator expressions + Expression closures	3.0				

Browser- Unterstützung (aus wikipedia.org)

Version	Release date	Equivalent to	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.8.1		1.8 + Native JSON support + Minor Updates	3.5				
1.8.2		1.8.1 + Minor updates	3.6				
1.9		1.8.1 + <u>ECMAScript</u> 5 Compliance	4				



Javascript und andere Programmiersprachen

- Häufig wird JavaScript auf Grund der Ähnlichkeit des Namens mit der Programmiersprache Java verglichen
- Dieser Vergleich bringt allerdings im Grunde genommen mehr Probleme mit sich als sich daraus Lern-Vorteile ergeben
 - Es entsteht sogar manchmal der Eindruck, dass eingefleischte Java-Programmierer mehr Schwierigkeiten haben, JavaScript zu verinnerlichen als „neutrale“ Einsteiger
- JavaScript hat keinerlei besondere Ähnlichkeiten oder Analogien zur Programmiersprache Java!



Grundlegende Eigenschaften: Sprache

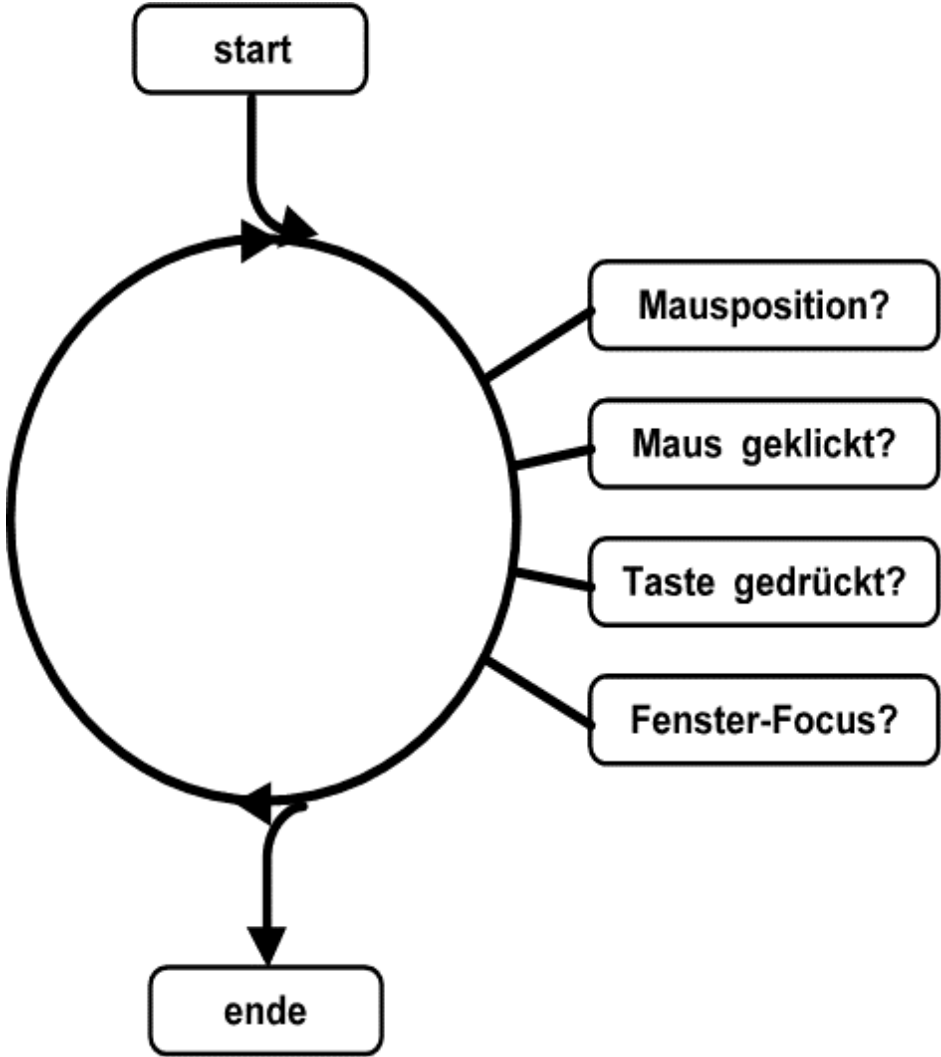
- JavaScript ist als Skript-Sprache relativ einfach zu erlernen.
- Die Laufzeitumgebung für JavaScript ist meistens der Browser
- JavaScript-Programme werden vom Browser als lesbare Textdatei geladen und liegen deshalb auf den Client-Maschinen im Klartext vor
- JavaScript hat aus Sicherheitsgründen keinerlei Zugriff auf Ressourcen außerhalb des Browsers.
 - Insbesondere kein Dateizugriff oder Öffnen von Socket-Verbindungen
- JavaScript ist eine typisierte Programmiersprache
 - Die Typprüfung erfolgt jedoch nur während der Ausführung
- JavaScript ist Objekt-orientiert
 - Objekte haben Attributen und Funktionen



Grundlegende Eigenschaften: Event-Loop

- Der Aufruf von JavaScript-Programmen erfolgt durch den Browser als Reaktion auf ein Ereignis
 - Das Laden der Seite
 - Benutzer-Interaktionen
- Es gibt somit keinen zentralen Start-Einstiegspunkt in das Programm
 - Keine „main“-Funktion
 - Statt dessen eine Reihe von vordefinierten Event-Handler-Funktionen, in denen die Programmlogik hinterlegt werden wird

Grundlegende Eigenschaften: Event-Loop





Grundlegende Eigenschaften: Implizite Objekte

- Jedes JavaScript-Programm hat jederzeit Zugriff auf implizite Objekte:
 - Das aktuell dargestellte HTML-Dokument
 - Das aktuelle Browser-Fenster
 - Grundlegende Browser-Funktionen
 - Statuszeile
 - Historie
 - Öffnen von Seiten



Werkzeuge

- Um in JavaScript Programme zu schreiben, benötigt man lediglich einen einfachen ASCII-Editor
 - Sinnvoll ist jedoch ein Syntax-Highlighting sowie ein Code-Assist
 - Eclipse hat beispielsweise einen hervorragenden JavaScript-Editor.
- Zum Testen der Programme wird ein Webbrowser benötigt, um Funktionstests ausreichend zu durchlaufen
 - Debugger und weitere Hilfswerkzeuge sind mittlerweile in allen modernen Browsern integriert



Übersicht der Beispielprogramme

- Die Beispiele aus den ersten Kapiteln sind mit möglichst wenig HTML- und Design-Aufwand realisiert, um den Fokus auf JavaScript zu legen
 - Für das Erlernen einer Programmiersprache müssen erst einmal die grundsätzlichen Sprachelemente vermittelt werden!
- Daran anschließend werden aber selbstverständlich auch Programme gezeigt, in denen JavaScript direkt auf die Elemente einer HTML-Seite zugreifen wird
 - Effekte oder die Formular-Validierung sind Beispiele dafür
 - Grundkenntnisse in HTML und CSS erleichtern hier das Verstehen



Grundlegende Sprachelemente



JavaScript und HTML



Funktionen



Erste Befehle



Variable



Kontrollstrukturen



Operatoren



JavaScript und HTML



Einbindung von JavaScript: HTML

- Am einfachsten wird JavaScript zwischen den `<script....></script>`-Tags eingebunden.

```
<script language="JavaScript">  
    Anweisungen  
</script>
```
- Die standardisierte Art der JavaScript-Einbindung nach dem W3C lautet:

```
<script type="text/javascript">  
    Anweisungen  
</script>
```
- Der Script-Tag kann sowohl im Head wie auch im Body stehen
 - Meistens findet man ihn, zwecks Übersichtlichkeit, im Head



Einbindung von JavaScript: Externes Skript

- Auslagern in eine externe Datei
 - Mit dem Attribut `src` wird auf die Datei, in der die Kommandos stehen, verwiesen



Aufrufen von JavaScript: Event- Handler

- Zuständig für den Aufruf sind so genannte Event-Handler
 - Der entsprechende JavaScript-Code für den Event-Handler wird in jenem HTML-Tag deklariert, dessen Bildschirmdarstellung auf die Benutzeraktivität reagieren soll
- Beispiel: Ein Hyperlink, der beim Überfahren des Hyperlinks mit dem Mauszeiger in einer Fensterbox eine Meldung ausgibt:

```
<a href="#" onmouseover="alert ('Hello,  
JavaScript' )" >einlink</a>
```

- In dem Event-Handler kann beliebiger JavaScript-Code stehen, der beim Eintreten des betreffenden Ereignisses ausgeführt wird
- HTML-Elemente bieten eine Vielzahl von Event-Handlern an
 - onclick
 - onmouseover
 - onkeyup
 - ...



Aufrufen von JavaScript: Links

- JavaScript-Code kann in einen Link eingefügt werden

```
<a href="javascript:alert('Hello,  
Javascript');">ein Link</a>
```

- Beim Klicken auf den Link wird der Javascript-Code ausgeführt



Erste Befehle



Kommentare in JavaScript-Code

- Der Erhöhung der Übersichtlichkeit dient es auch, wenn Sie Ihren JavaScript-Code kommentieren
- Folgende Kommentarmöglichkeiten sind möglich:
 - `//`Einzeiliger Kommentar
 - `/*`
Mehrzeiliger Kommentar
`*/`
- Vorsicht: Kommentare werden als Bestandteil des Skripts in lesbarer Form zum Browser gesendet!
 - Es gibt jedoch auch Filter, die JavaScript-Kommentare beim Senden des Skripts automatisch entfernen



Anweisungen

- Eine Anweisung in JavaScript besteht immer aus einem Befehl, der mit einem Strichpunkt ";" oder einem Zeilenumbruch abgeschlossen wird
- Eine Anweisung ist zum Beispiel:
 - wenn einer Variablen ein Wert zugewiesen wird
 - wenn mit Variablen oder Werten eine Operation durchgeführt wird
 - wenn bedingte Anweisungen ausgeführt werden
 - wenn eine selbst definierte Funktion oder eine Objektmethode aufgerufen wird
- Ein Anweisungsblock besteht aus einer oder mehreren Anweisungen, die innerhalb einer übergeordneten Anweisung oder innerhalb einer Funktion stehen
- Ein Anweisungsblock wird durch eine öffnende geschweifte Klammer { begonnen und durch eine schließende geschweifte Klammer } beendet



Ausdrücke

- Ein Ausdruck ist etwas, das einen Wert und einen Datentyp besitzt. Beispiele für Ausdrücke sind:
 - Zusammensetzungen aus Zahlenwerten und Rechenoperatoren, aber auch einfache Zahlenkonstanten wie z. B. 139 (sog. Literale)
 - Die Verkettung von Texten (String)
 - Funktionsaufrufe, denn eine Funktion liefert in JavaScript (zumindest formal) immer ein Ergebnis
 - Aus einem Ausdruck kann man leicht eine Anweisung machen, indem man einen Strichpunkt ";" anfügt. Der Inhalt dieser sogenannten Ausdrucksanweisung besteht dann darin, den Wert des genannten Ausdrucks auszurechnen



Selbstvergebene Namen

- Variablen, Funktionen und Objekte werden über einen Namen identifiziert
- Bei selbstvergebenen Namen gelten folgende Regeln:
 - sie dürfen keine Leerzeichen enthalten
 - sie dürfen nur aus Buchstaben und Ziffern bestehen, wobei das erste Zeichen keine Ziffer sein darf; es sind Groß- und Kleinbuchstaben erlaubt. Groß- und Kleinschreibung wird streng unterschieden!
 - sie dürfen keine Umlaute, scharfes S oder sonstige Sonderzeichen enthalten
 - sie dürfen als einziges Sonderzeichen den Unterstrich "_" enthalten, der auch am Anfang des Namens erlaubt ist
 - sie dürfen nicht mit einem reservierten Wort identisch sein
 - Dies sind die Schlüsselwörter von JavaScript wie `if`, `for` ...



Einfache Literale

- JavaScript kennt Literale für:
 - Zeichenketten
 - "45678"
 - Ganzzahlen
 - 42
 - Kommazahlen
 - 47.11
 - Logische Werte
 - `true` bzw. `false`
 - Reguläre Ausdrücke
 - `/[a..z]/`



Komplexe Literale

- Listen bzw. Arrays:
 - `[elem1, elem2, elem3]`
- Eigene Datentypen, die aus benannten Eigenschaften, den Attributen bestehen. Dies sind Objekte
 - `{attribut1: wert1, attribut2: wert2}`
- Auch Funktionen werden über ein Literal definiert
 - `function(params){impl}`



Steuerzeichen bei Zeichenketten

Steuerzeichen	Funktion
<code>\n</code>	Zeilenumbruch
<code>\b</code>	Backspace
<code>\r</code>	Wagenrücklauf
<code>\t</code>	Tabulator
<code>\"</code>	Anführungszeichen
<code>\'</code>	Hochkomma
<code>\\</code>	Backslash



Variable



Variablen und Werte

- Variablen werden in JavaScript üblicherweise über

```
var meintest;
```

bzw.

```
var meintest = Wert;
```

deklariert.

- Variablen sind Speicherbereiche, in denen Sie Daten, die Sie im Laufe Ihrer Programmprozeduren benötigen, speichern können
- Der Inhalt, der in einer Variablen gespeichert ist, wird als "Wert" bezeichnet
 - Sie können den Wert einer Variablen jederzeit ändern



Notation numerischer Werte

- Numerische Werte sind Ganzzahlig oder Komma-Zahlen
- Sie können Zahlen wie gewohnt notieren
 - Beachten Sie dabei nur, dass bei Kommazahlen anstelle eines Kommas ein Punkt verwendet werden muss
- Mit e oder E können Sie einen Exponenten angeben



Gültigkeitsbereich

- Es gibt globale Variablen und lokale Variablen.
 - Eine lokale Variable erhalten Sie durch die Deklaration der Variablen mit `var` innerhalb einer Funktion.
- Diese Variablen sind deshalb nur innerhalb dieser Funktion gültig
 - Man spricht in diesem Zusammenhang auch von der "Lebensdauer" von Variablen.
- Parameter, die einer Funktion übergeben werden, werden ebenfalls als lokale Variablen behandelt
- Eine globale Variable ist im gesamten Dokument gültig und steht jederzeit zur Verfügung
- Wenn Sie innerhalb von Funktionen Variablen ohne das Schlüsselwort `var` deklarieren, dann sind diese Variablen global
 - Genaueres hierzu später!



Kontrollstrukturen



Typisierung

- Welchen Typ eine Variable besitzt, entscheidet sich bei der ersten Wertzuweisung:

```
var x = "45678";    //String
var y = 45678;      //Zahl
var z = 13.2345;    //Zahl
```
- Hier enthält die Variable `x` den String (Buchstaben- und Ziffernfolge) 45678, während die Variable `y` die Zahl 45678 und die Variable `z` die Zahl 13,2345 enthält.
- Variable haben einen Typ, der zur Laufzeit festgelegt wird
 - `var x = "45678"; //String`
 - `var x = 45678; //Zahl`



Abfragen und Schleifen

- Bestandteil jedes Programms sind bedingte Anweisungen:
 - Wenn eine bestimmte Bedingung erfüllt ist, wird ein entsprechender Programmcode ausgeführt, wenn nicht, wird ein anderer Programmcode ausgeführt
- Hiermit sind Programme in der Lage, in Abhängigkeit von den Benutzereingaben zu unterschiedlichen Ergebnissen zu gelangen



Die if-Anweisung

- Die if-Anweisung fragt einmalig eine Bedingung ab und verzweigt dann je nach Ergebnis zu einem bestimmten Programmcode.

```
if (Bedingung)
{
  Anweisung1;
  Anweisung2;
  ...
}
else
{
  Anweisung7;
  Anweisung8;
  ...
}
```



Verkürzte Form

- Es gibt außerdem eine verkürzte Variante der if-else Anweisung:

`(Bedingung) ? Wert1 : Wert2`

- Ist die Bedingung erfüllt, ihr Wert also true, so wird Wert1 zurückgegeben, ansonsten Wert2.

- Beispiel:

```
var tageszeit= (Essen ==  
"Pizza") ? "Mittag" : "Abend";
```

- Ist der Wert für Essen „Pizza“, dann wird der Variablen tageszeit der Wert Mittag zugewiesen, ansonsten der Wert Abend.



switch-case- Anweisung

- Die switch-case-Anweisung prüft bei switch den Inhalt eines Wertes
- Jeder case-Block muss mit break abgeschlossen werden
- Am Ende steht die default- Anweisung

```
switch (Ausdruck) {  
  case FallA: {  
    Anweisung1; break;  
  }  
  case FallB: {  
    Anweisung2; break;  
  }  
  default: {  
    Anweisung4;  
  }  
}
```



Schleifen: while-Anweisung

- Die while-Anweisung führt solange eine Schleife aus, bis eine bestimmte Bedingung erfüllt ist:

```
while (Bedingung)
{
  Anweisung;
  Anweisung;
}
```




Schleifen: do-while-Anweisung

- In der do-while Schleife wird der Code der Schleife einmal ausgeführt und erst danach die Bedingung geprüft
- Ist die Prüfung positiv wird die Schleife erneut ausgeführt

```
do  
{  
  Anweisung;  
}  
while (Bedingung)
```



Schleifen: for-Anweisung

- Die for-Anweisung führt eine Anweisung solange durch, bis ein gesetzter Zähler den vorgegebenen Zählerstand erreicht hat:

```
for(Initialisierung;Bedingung;Zähleranweisung)
{
Code, der ausgeführt werden soll;
}
```



Schleifensteuerung: `break` und `continue`

- `break`
 - Veranlasst das sofortige Verlassen der aktuellen Schleife
 - Der Interpreter fährt also hinter dem aktuellen Anweisungsblock fort
- `continue`
 - Springt zum nächsten Schleifendurchlauf
 - Der Rest des Schleifenkörpers wird für diesen Durchlauf nicht ausgeführt



For-in-Schleife

- Mit dieser Schleife kann man durch alle Eigenschaften eines Objektes und alle Elemente eines Arrays laufen.

```
for (a in object | array) {  
    //Anweisungen  
}
```



For-each-Schleife

- Mit dieser Schleife kann man durch alle Eigenschaften eines Objektes und alle Elemente eines Arrays laufen
 - Diese Variante ist nicht ECMA-Standard

```
for each (a in object | array) {  
    //Anweisungen  
}
```



Operatoren



Wichtige Rechenoperatoren

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Teilungsrest bei einer Division)
++	um 1 erhöhen (Inkrement)
--	um 1 vermindern (Dekrement)



Vergleichsoperatoren

<code>==, ===</code>	Gleichheit, mit oder ohne Konvertierung
<code>></code>	Größer
<code><</code>	Kleiner
<code>>=</code>	Größer gleich
<code><=</code>	Kleiner gleich
<code>!=, !==</code>	ungleich



Logische Operatoren

 	OR
&&	AND
!	Negation



Funktionen



Funktionen

- Die allgemeine Form einer selbstdeklarierten Funktion in JavaScript lautet:

```
function name([var1][,var2][,var3])  
{  
    Anweisung1;  
    Anweisung2;  
    ...  
    [return ausdruck;]  
}
```

- Dabei bedeuten:
 - name() der Name der Funktion
 - var1, var2, var3 ... sind Namen von Parametern
 - Diese Parameter sind innerhalb der Funktion als lokale Variable bekannt.
 - return ausdruck; benennt einen optionalen Rückgabewert der Funktion



Funktionen

- Die Parameter-Anzahl in der Funktions-Definition und im Funktions-Aufruf ist vollkommen gleichgültig:
 - Überschüssige Parameter werden ignoriert.
 - Ein „überladen“ von Funktionen, das heißt eine Unterscheidung gleich benannter Funktionen über eine Parameter-Liste ist in JavaScript nicht möglich
 - Die jeweils letzte Funktion wird effektiv benutzt und ersetzt alle vorherigen Definitionen.
- Nicht vorhandene Parameter bekommen die JavaScript-Ausdruck den internen Wert `undefined` zugeordnet
- Alle Parameter stehen innerhalb der Funktion in dem Array `arguments` zur Verfügung



Objektorientierte Programmierung



Speicherverwaltung und Referenzen



Von Datenstrukturen zu Objekten



Exkurs: Konstruktoren und Klassen



Spezielle Referenzen



Expertenwissen: Das prototype-Objekt



Funktionen im Detail



Fehlerbehandlung



JavaScript Standard Objekte



Speicherverwaltung und Referenzen



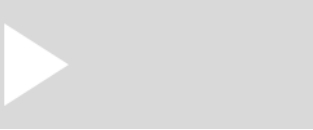
Arrays

- Arrays sind Variablencontainer, die mehrere Variablen enthalten können
- Auf einzelne Variable greift man über den Variablennamen und eine Nummer zu
- Als Literal-Kennzeichen für Arrays wird eine eckige Klammer benutzt, die Werte sind durch Kommas getrennt:

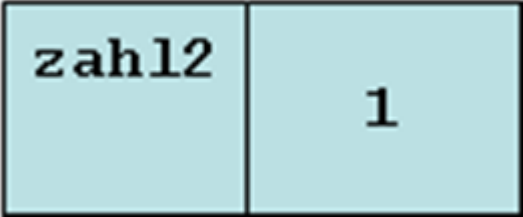
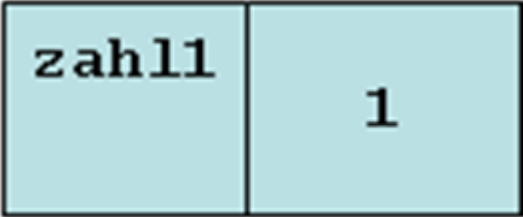
```
var theBeatles = ["John", "Paul", "George",  
"Ringo"];
```
- Der Index, über den auf ein Array-Element zugegriffen wird, steht in eckigen Klammern:

```
var john = theBeatles[0];
```
- Ein Array kann jederzeit mit weiteren Werten befüllt werden bzw. die vorhandenen Einträge geändert werden
 - Der Zugriff erfolgt ebenfalls über den Index:

```
theBeatles[4] = "George Martin";
```



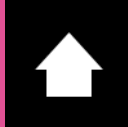
Speicherverwaltung: Einfache Datentypen





Speicherverwaltung: Einfache Datentypen

- Wird beispielsweise `zahl2` inkrementiert, so hat diese selbstverständlich nur Auswirkungen auf einen Speicherbereich:
`zahl2++;`



Speicherverwaltung: Einfache Datentypen

- Das ist sicherlich vollkommen klar und verständlich
- Was passiert aber nun bei einer Zuweisung an eine weitere Variable:
`var zahl3 = zahl1;`
- Das ist nun nicht automatisch klar, denn es gibt prinzipiell zwei Möglichkeiten



Speicherverwaltung: Einfache Datentypen

- Der Speicherbereich bekommt einen weiteren Namen:



Speicherverwaltung: Einfache Datentypen

- Der Wert wird in einen neuen Speicherbereich kopiert:

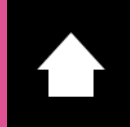


Speicherverwaltung: Einfache Datentypen

- Welcher der beiden Varianten korrekt ist bestimmt ein einfacher Test:
- Wir erhöhen die Variable zahl3 und schauen nach, was in zahl1 steht:

```
zahl3++;  
alert(zahl1)
```

- Die Ausgabe ist „1“ und somit ist klar, dass bei der Zuweisung an Variablen der Wert an einen neuen Speicherbereich kopiert wird

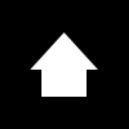


Speicherverwaltung: Parameterübergabe

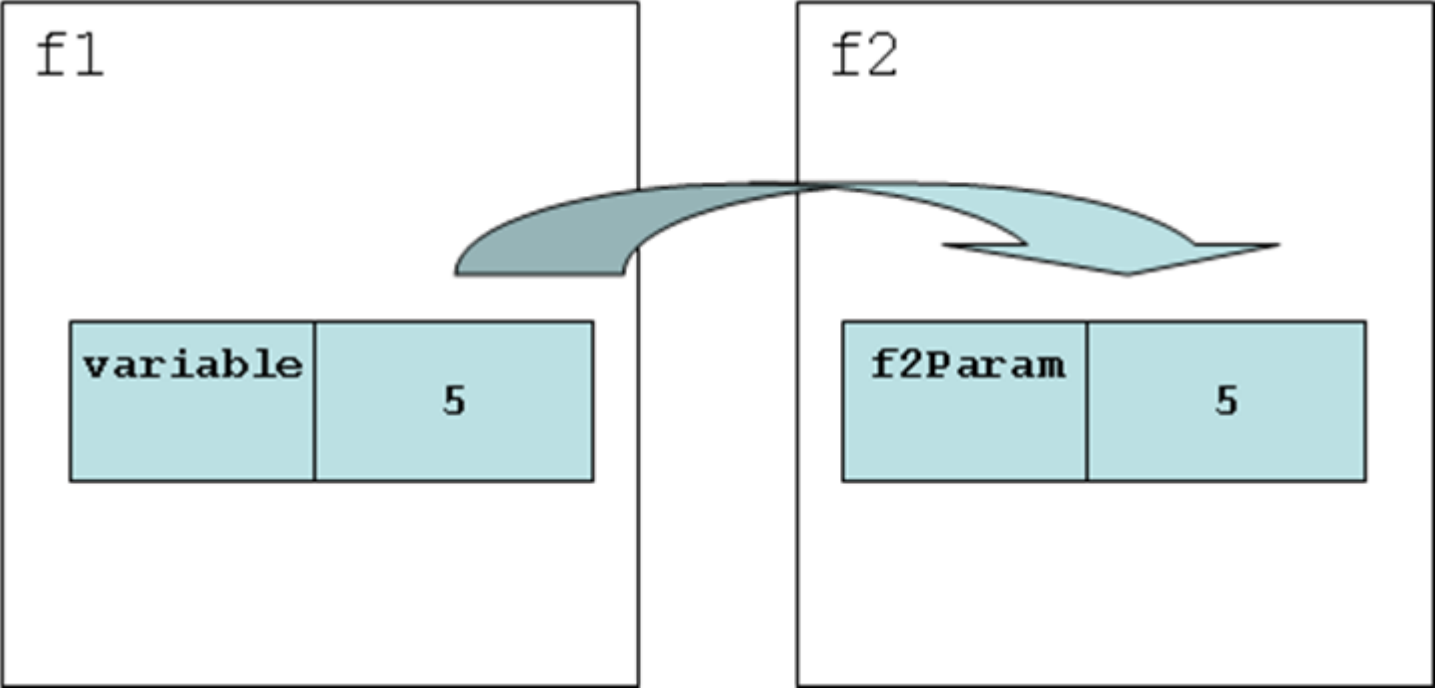
- Was hier am Beispiel von Variablen gezeigt wurde ist auch bei einem Funktionsaufruf relevant:
- Auch hier wird die übergebene Variable in den benannten Funktionsparameter kopiert:

```
function f1(){  
  var variable = 5;  
  f2(variable);  
}  
function f2(f2Param){  
  //aktionen mit f2Param durchführen  
  f2Param = 7;  
}
```

- Sämtliche Änderungen, die innerhalb von f2 mit f2Param passieren, sind lokal innerhalb von f2 und haben keine Auswirkung auf die Variable variable in f1.



Speicherverwaltung: Parameterübergabe





Arrays

- Nun gehen wir aber zurück auf das Array-Beispiel. Dies verhält sich nämlich grundsätzlich anders:

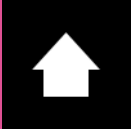
```
var theBeatles = ["John", "Paul", "George",  
"Ringo"];  
  
var theBeatles2 = theBeatles;  
theBeatles[4] = "George Martin";  
alert(theBeatles2[4])
```

- Hier wird nämlich sehr wohl der „5. Beatle“ George Martin ausgegeben!

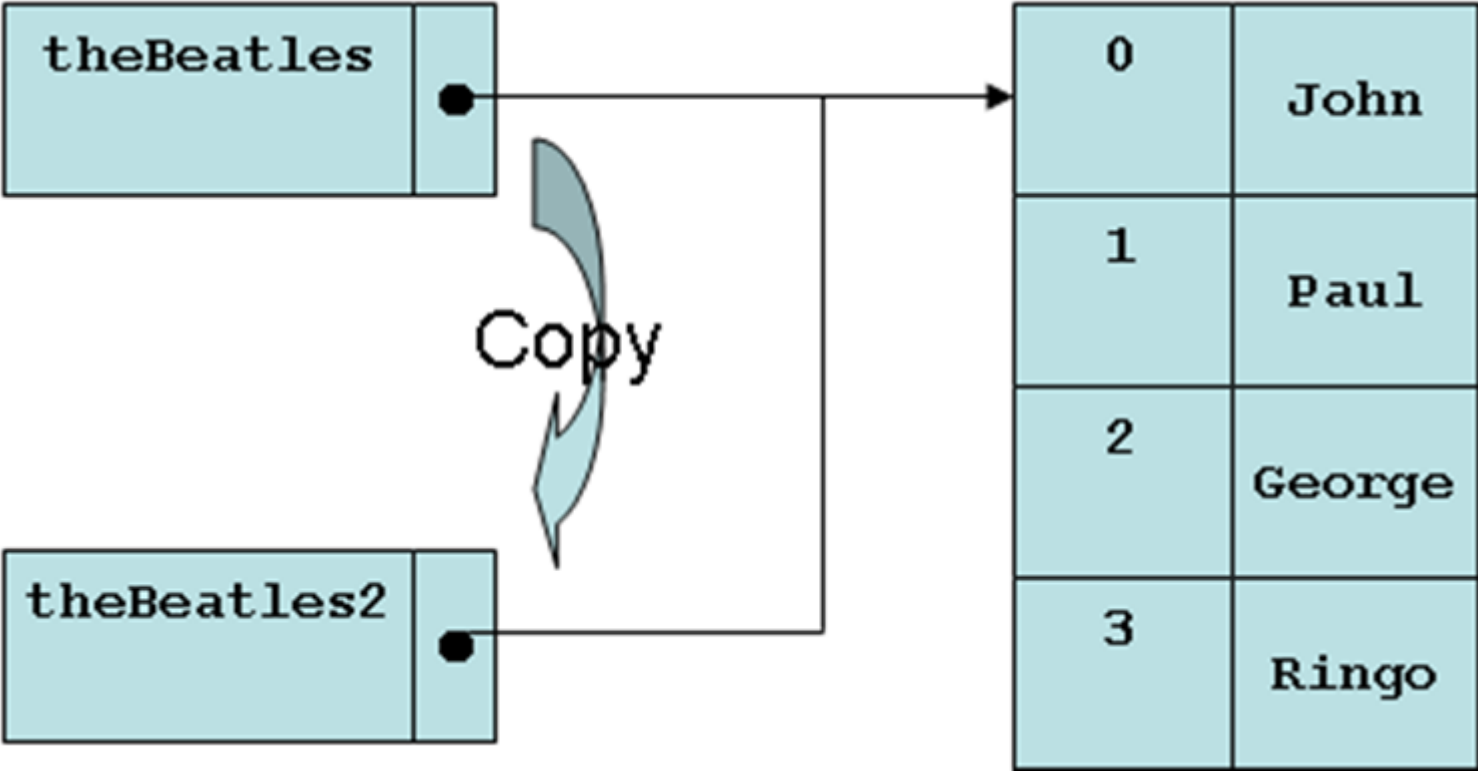


Arrays

- Eine grafische Darstellung arbeitet am Besten mit einem Pfeilsymbol, den sogenannten Referenzen:
 - Technisch gesehen ist eine Referenz eine interne Speicheradresse.
- Auch hier wird bei der Zuweisung bzw. bei einer Parameterübergabe an eine Funktion ein Wert kopiert: Es ist hier allerdings der Wert der Referenz!
- Wird somit ein Array einer Funktion als Parameter übergeben, so schlagen Änderungen, die innerhalb der aufgerufenen Funktion durchgeführt werden, sehr wohl auf die Variablen der aufrufenden Funktion durch
 - Ein fundamentaler und sehr wichtiger Unterschied im Vergleich zu den bisher benutzten Variablen!



Arrays





Von Datenstrukturen zu Objekten



Hashes

- Eine Datenstruktur kann auch mit Schlüssel-Werte-Paaren definiert werden:

```
var theBeatlesGroup = {leadGuitar: "George",  
  rhythmGuitar: "John", bass: "Paul", drums:  
  "Ringo"};
```

- Im Unterschied zu den Array verwenden wir hier ein paar geschweifter Klammern
 - Weiterhin müssen wir natürlich die Schlüssel-Namen bei der Definition des Hashes mit angegeben werden
- Der Zugriff erfolgt nun über den Schlüssel, ebenfalls angegeben in eckigen Klammern:

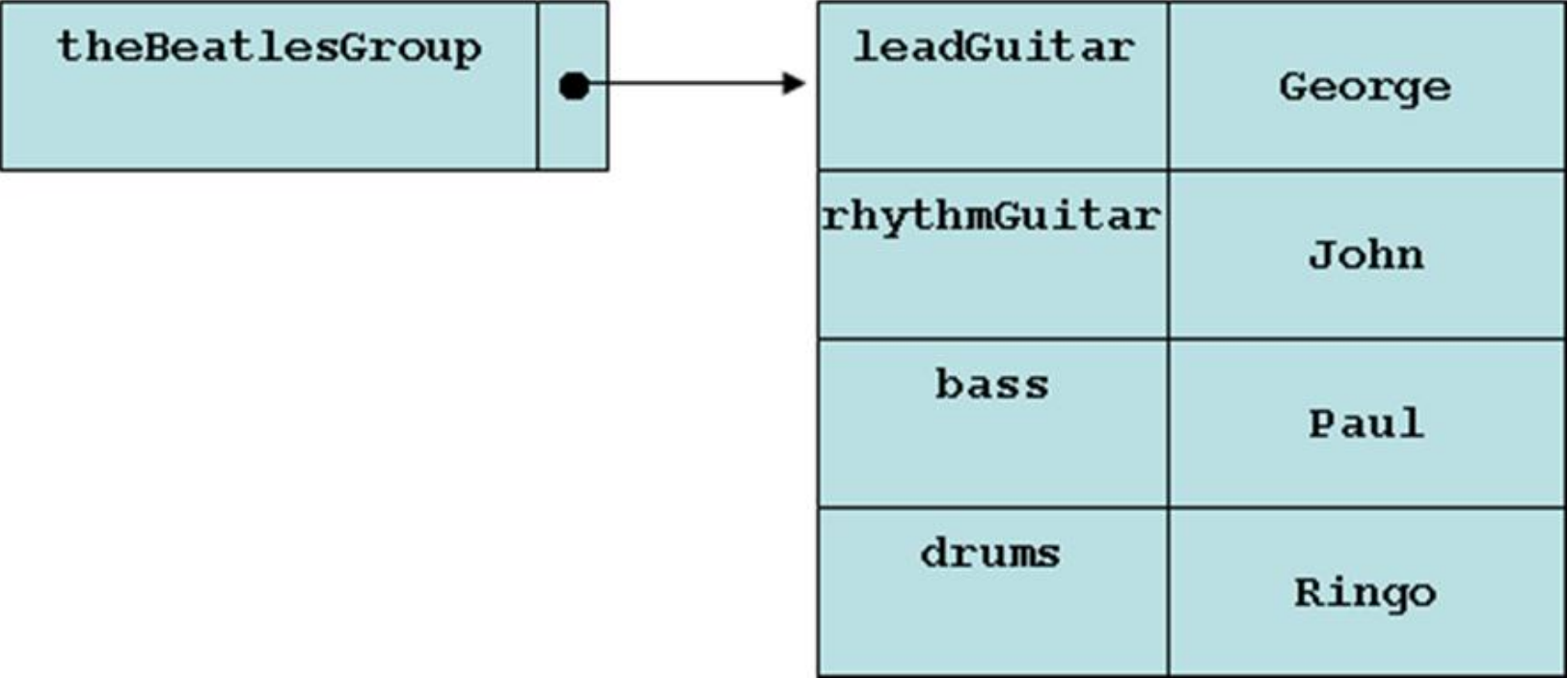
```
alert(theBeatlesGroup["drums"])
```

liefert uns „Ringo“.

- Auch Hashes werden über Referenzen angesprochen



Hashes





Vom Hash zum Objekt

- Statt der eckigen Klammern kann für den Zugriff auch der Punkt-Operator benutzt werden:
`theBeatlesGroup.drums`
- Die Bedeutung dieses Punkts ist für den JavaScript-Laufzeitumgebung einfach zu interpretieren: Der Punkt-Operator heißt: „Verfolge die Referenz“
- Diese erst einmal einfach scheinende Änderung hat begrifflich weit reichende Folgen: Statt eines „Assoziativen Arrays“ mit Schlüsseln sprechen wir von einem Objekt mit Eigenschaften
 - Rein technisch gesehen unterscheidet JavaScript nicht zwischen Hashes und Objekten
 - oder noch genauer: (Fast) alles in JavaScript sind Objekte



Objekte

- Was nun extrem wichtig für den Objekt-Begriff ist, ist die Tatsache, dass eine Eigenschaft eines Objekts auch eine Funktion sein kann!
- Oder anders formuliert:
 - „Ein Objekt hält Daten-Informationen (die Eigenschaften oder Attribute) und Verhalten (die Funktionen oder Methoden)“
 - Dies ist eine grundlegende Definition innerhalb der Objektorientierten Programmierung
- JavaScript interpretiert den Objekt-Begriff dynamisch:
 - Jedem Objekt können zur Laufzeit beliebige Eigenschaften neu zugeordnet werden



Objekte

- Simple Zuweisung:

```
var theBeatles = {leadGuitar: "George",  
rhythmGuitar: "John", bass: "Paul", drums:  
"Ringo"};  
theBeatles.home = "Liverpool";
```

- Als Eigenschaften können natürlich auch wieder komplexe Referenz-Typen benutzt werden, also zum Beispiel ein Array:

```
theBeatles.albums = ["Please please me",  
/*usw*/, "Let it be"];
```

- ein anderes Objekt:

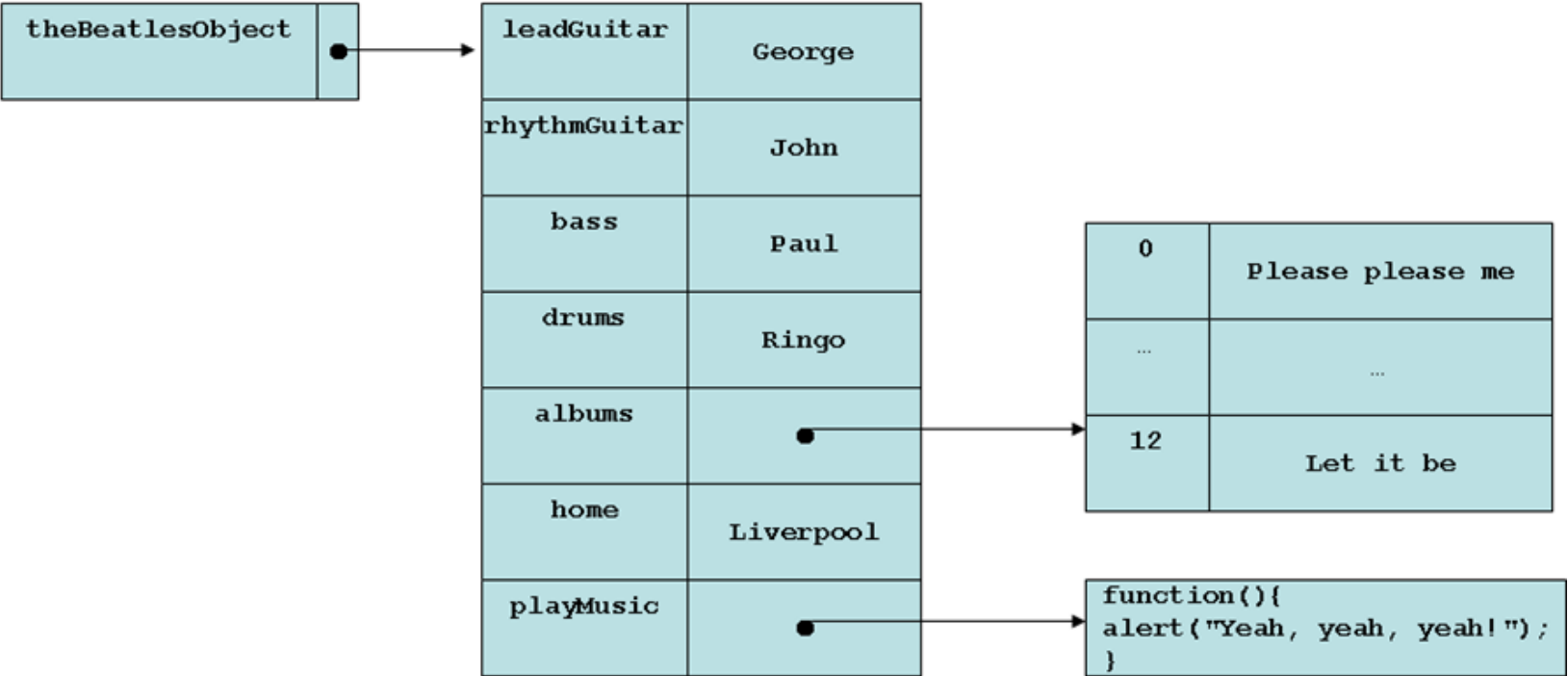
```
theBeatles.lifetime = {start: 1962, end: 1970};
```

- oder eben eine Funktion:

```
theBeatles.playMusic = function() {alert("Yeah,  
yeah, yeah!");};
```

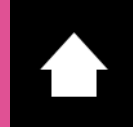



Ein Blick in den Speicher





Exkurs: Konstruktoren und Klassen



Konstruktor-Funktionen

- Statt Objekte über ein {}-Literal zu definieren kann auch eine Konstruktor-Funktion benutzt werden
- Der Konstruktor bekommt für jede Eigenschaft einen Parameter
- Innerhalb der Konstruktor-Funktion werden diese Parameter den Eigenschaften des neuen Objekts zugewiesen

```
function Person(lastname, firstname){  
  this.lastname = lastname;  
  this.firstname = firtsname;  
}
```

- Das Erzeugen von Instanzen kann nun an anderer Stelle im JavaScript-Programm mit dem Operator `new` geschehen

```
var person = new Person('Lennon', 'John');
```

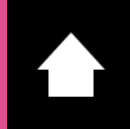


JavaScript und Klassen

- Die Konstruktor-Funktionen und der `new`-Operator sind in JavaScript notwendig, da es keine Klassen-Definitionen gibt
 - Eine Klasse ist ein abstraktes Template, aus dem Objekte erzeugt, besser: instanziiert werden
 - Jede Instanz einer Klasse hat damit einen durch die Klassen-Definition Satz von Eigenschaften
- Klassen sind in anderen Programmiersprachen wie Java und C# weit verbreitet
 - und sind bei Entwicklern sehr beliebt
- JavaScript und Klassen:
 - Workarounds sind möglich
 - Das "Module-Pattern" ist ein Beispiel hierfür
 - Die Sprache Typescript führt Klassen ein
 - Aus Typescript wird JavaScript generiert
 - ES2015 führt Klassen ein



Spezielle Referenzen



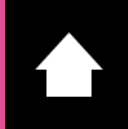
this

- Das eben eingeführte `this`-Schlüsselwort in JavaScript muss noch genauer beschrieben werden
- `this` ist die Referenz auf den aktuellen Aufruf-Kontext
 - Ohne weitere Aktionen ist dies das „Globale Objekt“, siehe unten, das für jedes JavaScript-Programm definiert ist



Der new-Operator

- Jeder normale Funktionsaufruf findet im diesem Kontext statt. Wird jedoch der Aufruf mit dem Schlüsselwort new durchgeführt passiert folgendes:
 - Es wird ein neues Objekt im Speicher erzeugt
 - Die this-Referenz deutet auf dieses neu erzeugte Objekt
 - Nicht mehr auf das globale Objekt!
 - Die constructor-Eigenschaft dieses neuen Objekts wird mit der aufgerufenen Funktion belegt
- Sämtliche Aktionen innerhalb der Konstruktor-Funktion beziehen sich somit auf das neu erzeugte Objekt



Undefined und null

- `undefined` ist etwas, was noch nicht definiert ist
 - Wird beispielsweise auf eine Eigenschaft eines Objektes verwiesen, die nicht existiert, so wird der Wert `undefined` benutzt.
- `null` hingegen ist zwar vorhanden, soll aber aus Sicht der Anwendung „nichts“ beinhalten
- Hinweis:
 - In Abfragen werden implizit beide Objekt-Referenzen als `false` evaluiert



Garbage Collection

- Objekte werden, wie wir gesehen haben, mit Hilfe von `new` erzeugt
- Können Objekte auch durch eine Anweisung gelöscht werden?
 - Prinzipiell „Ja“, dafür gibt es die Anweisung `delete`
 - Allerdings ist die Anwendung dieser Anweisung optional,
 - statt dessen kontrolliert die JavaScript-Laufzeitumgebung selber, welche Objekte vom Programm noch benutzt werden können
 - Alle anderen werden automatisch entfernt
- Dieser ständig laufende Hintergrundprozess wird „Garbage Collection“ genannt
- Damit muss sich der Programmierer nicht um die Speicherbereinigung kümmern

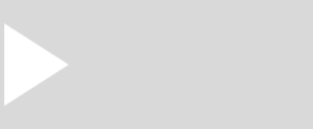


Objekt-Funktionen

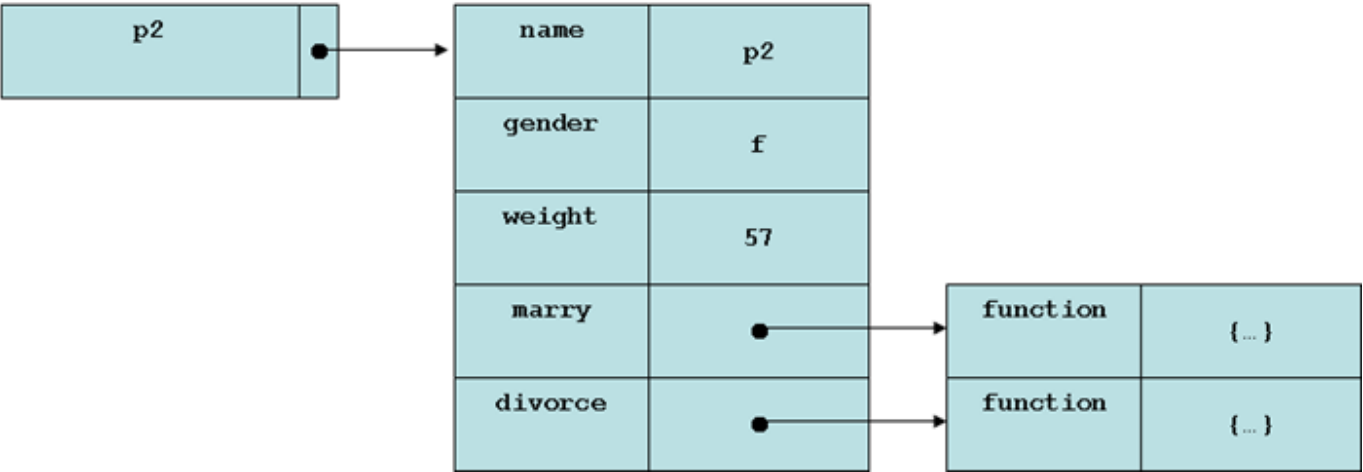
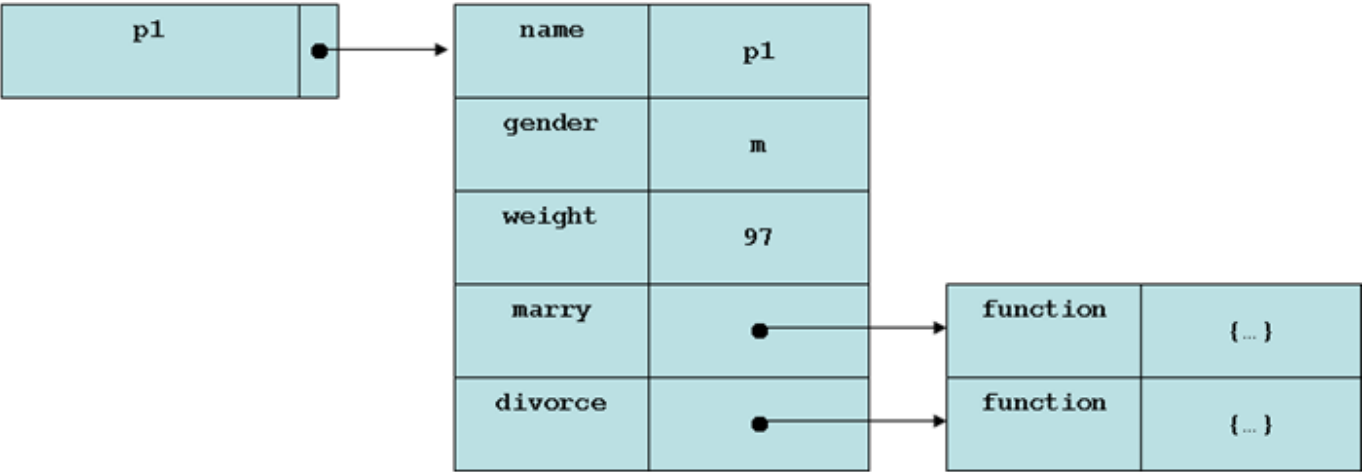
- Werden in einer Konstruktor-Funktion dem Objekt Funktionen zugewiesen, so wird für jedes Objekt ein eigenes Funktionsobjekt neu erzeugt

```
function Person(lastname, firstname, age){  
  this.lastname = lastname;  
  this.firstname = firtsname;  
  this.age = age;  
  this.marry = function(){//...}  
  this.divorce = function(){//...}  
  
}
```

```
var p1 = new Person();  
var p2 = new Person();
```

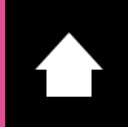


Objekt-Funktionen



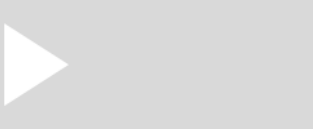


Expertenwissen: Das prototype-Objekt

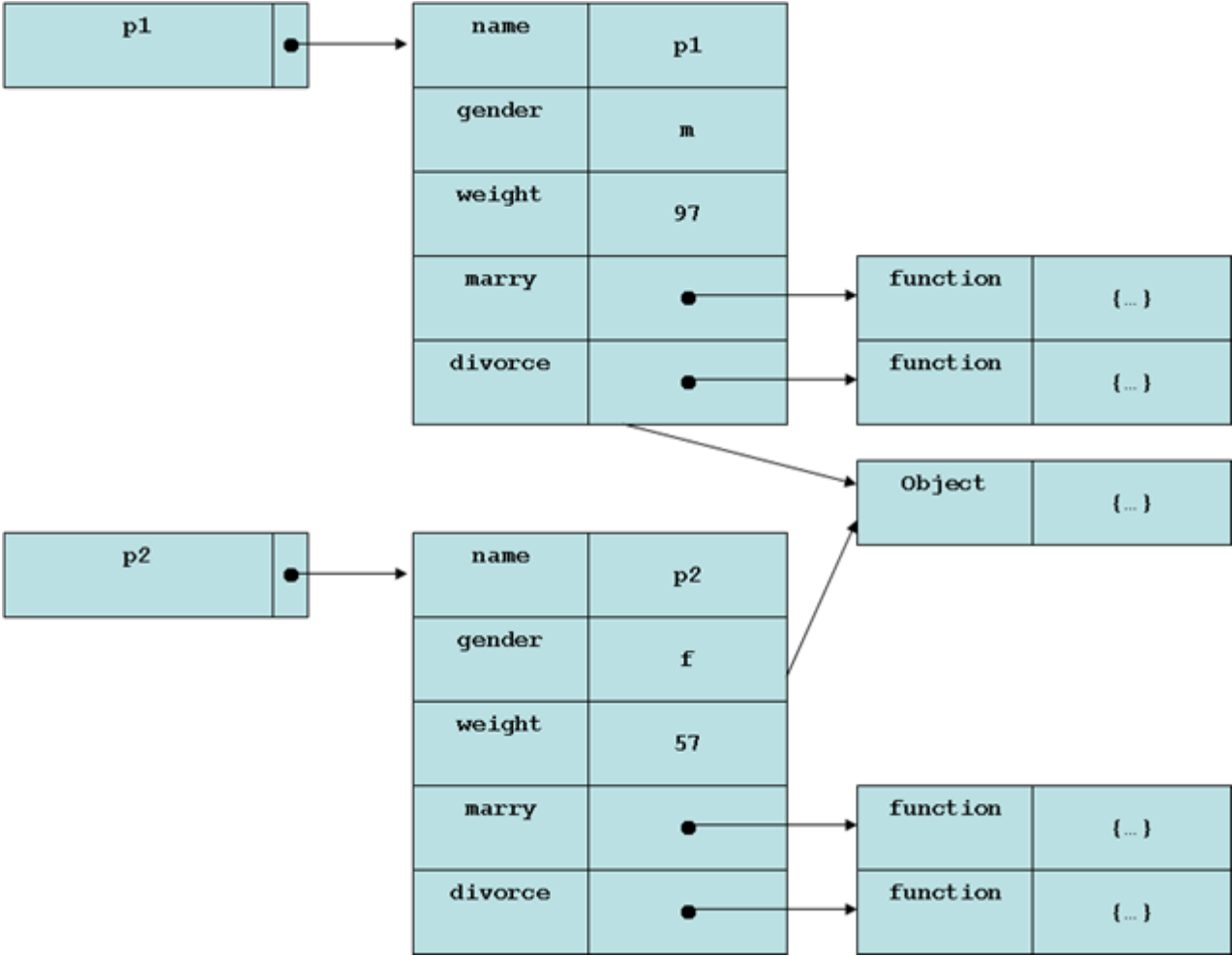


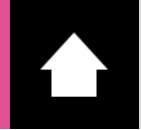
Das prototype-Objekt

- Diese Verschwendung können wir unterbinden:
 - Jedes Objekt besitzt eine interne weitere Referenz auf sein Prototyp-Objekt
 - Dieses ist ohne weitere Aktionen nichts anderes als ein normales Object ohne besondere Eigenschaften und Methoden.



Referenz auf das prototype-Objekt



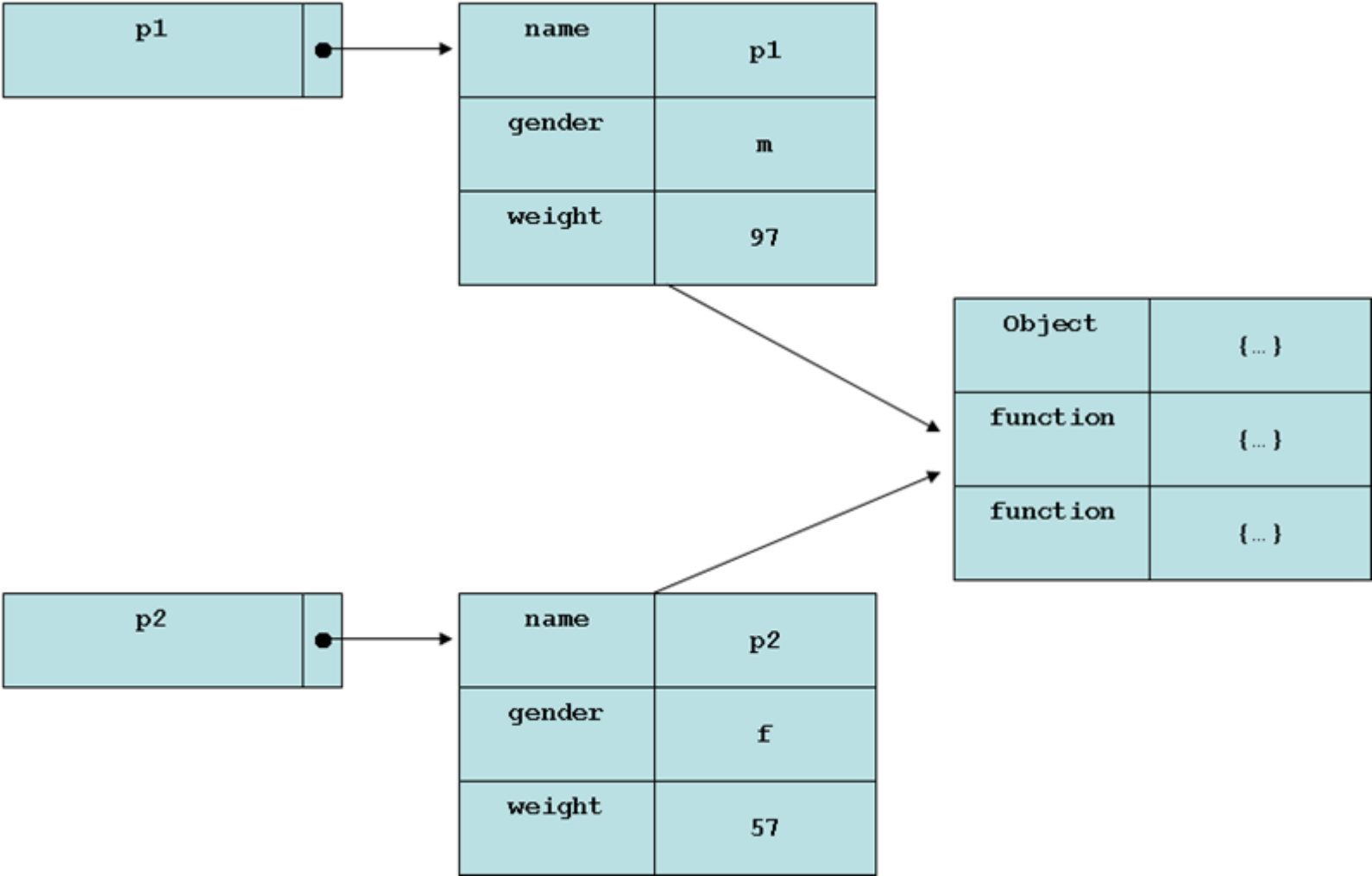


Das prototype-Objekt

- Der JavaScript-Interpreter berücksichtigt bei der Suche nach Funktionen jedoch das Prototype-Objekt
 - Der Aufruf `object.myFunction()` ist also erfolgreich wenn
 - entweder `object` unter dem Namen `myFunction` ein Funktionsobjekt besitzt oder
 - `object.prototype` unter dem Namen `myFunction` ein Funktionsobjekt besitzt



Arbeitsweise des prototype-Objekts



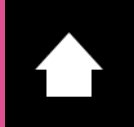


Funktionen im Detail



Das globale Objekt

- Jedem JavaScript-Programm wird eine globale Objekt-Instanz zur Verfügung gestellt
 - Dieses ist abhängig von der Laufzeitumgebung, in der das Skript aufgerufen wird
 - In der Praxis ist diese Umgebung in den allermeisten Fällen der Browser
 - Das globale Objekt ist das aktuelle Browser-Fenster repräsentiert durch das `window`-Objekt.



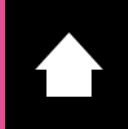
Funktionen des globale Objekts

<code>decodeURI ()</code>	Kodierten URI dekodieren
<code>decodeURIComponent ()</code>	Kodierten URI dekodieren - II
<code>encodeURI ()</code>	URI kodieren
<code>encodeURIComponent ()</code>	URI kodieren - II
<code>eval ()</code>	JavaScript-Code interpretieren
<code>escape ()</code>	Zeichen in Zahlencodes umwandeln
<code>isFinite ()</code>	Auf numerischen Wertebereich prüfen
<code>isNaN ()</code>	Auf numerischen Wert prüfen
<code>parseFloat ()</code>	Text in Kommazahl umwandeln
<code>parseInt ()</code>	Text in Ganzzahl umwandeln
<code>Number ()</code>	auf numerischen Wert prüfen
<code>String ()</code>	in Zeichenkette umwandeln
<code>unescape ()</code>	Zahlencodes in Text umwandeln



Funktionen und das globale Objekt

- Eine Funktions-Definition der Form
 - `function myFunction() {...}`
- ist in Wahrheit völlig identisch zu:
 - `window.myFunction = function() {...}`
- Funktionen werden so dem globalen Objekt zugeordnet.
 - Einem Aufruf einer Funktion wird somit implizit die `window`-Referenz vorangestellt.
- Dieses Prinzip ist auch für Variablen gültig:
 - Ohne `var` wird implizit bei der Deklaration ein `window` vorangestellt
- Können Funktionen auch anderen Objekten zugeordnet werden?
 - Selbstverständlich
 - genau das passiert ja bei der Zuweisung einer Funktion an ein anderes Objekt im Rahmen der Objekt-orientierten Programmierung.



Das Funktions- Objekt: Details

- Funktionen sind auch wiederum Objekte, die so wie alle anderen Objekte auch als Variable oder als Parameter benutzt werden können
- Zum Aufruf eines Funktions-Objekts existieren die beiden Methoden `apply` und `call`
 - Beide Funktionen bekommen als ersten Parameter die Referenz, die innerhalb der Methode als `this` benutzt werden soll
 - Die Funktionen unterscheiden sich nur in den weiteren Parametern
 - `apply` verlangt ein Array der Aufrufparameter
 - `call` benutzt eine beliebig lange Parameterliste



Closures

- Funktionen können auch innerhalb anderer Funktionen definiert werden
 - Dies haben wir auch bereits bei den Konstruktor-Funktionen gesehen
- Variablen der umhüllenden Funktion bleiben so lange gültig wie die innere Funktion
 - Dies nennt man allgemein „Closures“



Closures

- Ein potenziell gefährlicher Effekt von Closures darf nicht verschwiegen werden:
 - Durch die Verlängerung des Gültigkeitsbereiches kann die Garbage Collection die Objekte nicht mehr am Ende des Aufrufs sofort bereinigen
 - Insgesamt wird somit mehr Speicher benötigt
 - durch fehlerhafte Verwendung von Closures kann es sogar dazu führen, dass der Browser seine Speicher-Ressourcen erschöpft
 - es kommt zu einem Speicherleck

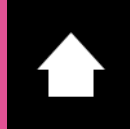


Fehlerbehandlung



Fehlerbehandlung und Error-Objekte

- Jeder Aufruf von Programmlogik kann potenziell zu Fehlersituationen führen
- Die Auswertung einer Fehlersituation wird durch einen `try-catch`-Block erfolgen
 - Im `try`-Block steht die normale Anwendungs-Sequenz
 - Im `catch`-Block erfolgt die Fehlerbehandlung mit Zugriff auf ein Fehler-Objekt
 - Das Fehler-Objekt hat die Eigenschaften `message` und `name`, die eine genauere Analyse des Fehlers ermöglichen
- Durch den `try-catch`-Block wird eine saubere Unterteilung zwischen erwartetem Programmablauf und Fehler-Behandlung erreicht.



throw

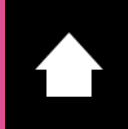
- Fehler können an allen Stellen des Programms auftreten.
 - So kann beispielsweise eine Division durch 0 erfolgen oder aber ein Objekt benutzt werden, das noch gar nicht initialisiert worden ist.
 - Solche Fehler sind in der Regel jedoch Programmierfehler und werden intern von der JavaScript-Engine des Browsers erzeugt.
- Möchte eine eigene Anwendung sich ebenfalls in diesen Fehler-Mechanismus einklinken, so erfolgt dies mit Hilfe der Erzeugung eines Error-Objekts sowie dem Werfen dieses Objekts mit `throw`



finally

- Sollen bestimmte Aufgaben sowohl für die normale Ausführung als auch für die Fehlerbehandlung ausgeführt werden kann der **try-catch-Block** durch einen **finally-Block** ergänzt werden

```
function testExceptions() {
  try {
    throwException();
  } catch (error) {
    window.alert("Caught error: message=" + error.message
+ ", name="
+ error.name);
  }
  finally{
    window.alert("Always");
  }
}
```



Ein komplettes Beispiel

```
function testExceptions() {  
    try {  
        throw new Error();  
    } catch (error) {  
        window.alert("Caught error: message=" +  
            error.message + ", name=" +  
                error.name);  
    }  
    finally{  
        window.alert("Always");  
    }  
}
```



JavaScript Standard Objekte



Der JavaScript- Objektkatalog

- Array
- Date
- Math
- Reguläre Ausdrücke
- ...
- Seit ES2015 auch Listen und Maps (Dictionaries)



Error-Typen

- Die JavaScript-Bibliothek enthält einen überschaubaren Satz von Fehlern, die im Wesentlichen von der JavaScript-Engine benutzt werden.
- Diese Laufzeitfehler sind:
 - `EvalError`
 - `ReferenceError`
 - `RaiseError`
 - `RangeError`
 - `SyntaxError`
 - `TypeError`
 - `URIError`



Dokumentation der Standard-Objekte

- Einfach Online zu finden
 - HTML-basiert
- Beschrieben werden alle Funktionen und andere Eigenschaften
- Beispielprogramme, Code-Fragmente und Tutorials stehen ebenfalls zur Verfügung
 - Natürlich auch die Beispiele der elektronischen Musterlösung



Beispiel: String

JavaScript Strings

A JavaScript string stores a series of characters like "John Doe".

A string can be any text inside double or single quotes:

```
var carname = "Volvo XC60";  
var carname = 'Volvo XC60';
```

String indexes are zero-based: The first character is in position 0, the second in 1, and so on.

For a tutorial about Strings, read our [JavaScript String Tutorial](#).

String Properties and Methods

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Properties

Property	Description
constructor	Returns the string's constructor function
length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

String Methods

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a new joined strings



Beispiel: Math

Math Object

The Math object allows you to perform mathematical tasks.

Math is not a constructor. All properties/methods of Math can be called by using Math as an object, without creating it.

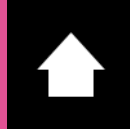
Syntax

```
var x = Math.PI;           // Returns PI
var y = Math.sqrt(16);     // Returns the square root of 16
```

For a tutorial about the Math object, read our [JavaScript Math Tutorial](#).

Math Object Properties

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)



Code- Vervollständigung im Browser

- JavaScript-Editoren können zumindest eine rudimentäre Code-Vervollständigung anbieten
 - Aber kein Vergleich zu statisch typisierten Sprachen wie Java oder C#
 - Der Editor kann den konkreten Datentyp und damit die anzuzeigenden Möglichkeiten nur "erraten"



Beispiel: Code- Vervollständigung

```
var message = "Hello World!";  
message.
```

- charAt(Number position) : String - String
- charCodeAt(Number position) : Number - String
- concat(String value) : String - String
- fromCharCode(Number charCode) : String - String
- hasOwnProperty(String name) : Boolean - Object
- indexOf(String searchString, Number startPosition) : Number - String
- isPrototypeOf(Object o) : Boolean - Object
- lastIndexOf(String searchString, Number startPosition) : Number - String
- localeCompare(String otherString) : Number - String
- match(RegExp regexp) : any[] - String
- propertyIsEnumerable(Object name) : Boolean - Object

Press 'Ctrl+Space' to show Template Proposals



UI-Programmierung



Browser-Objekte



Das Event-Modell



Browser-Objekte

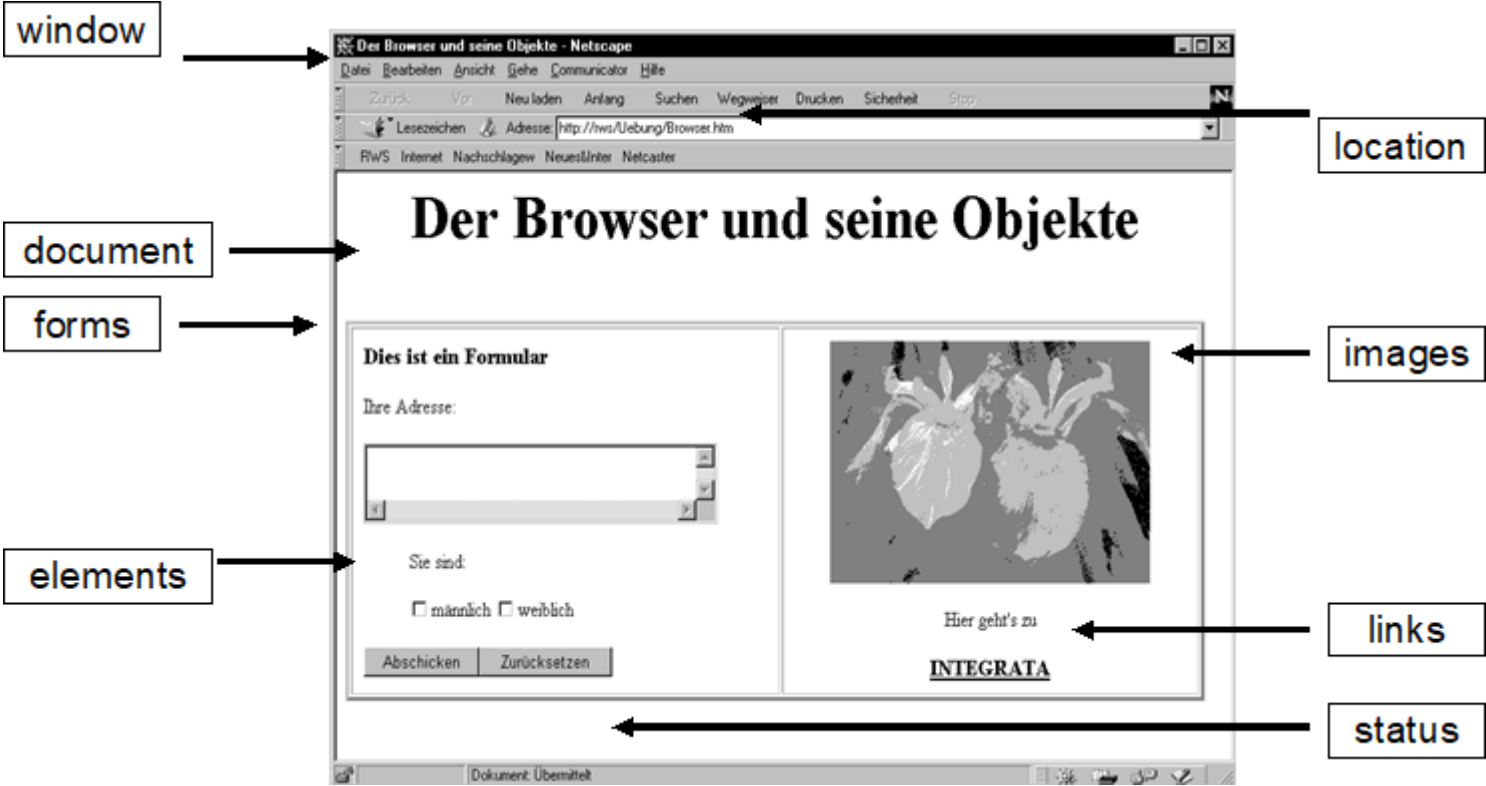


Browser-Objekte und JavaScript

- Einige Objekte stehen einem JavaScript-Programm im Browser automatisch zur Verfügung:
 - Das vom Browser dargestellte Dokument
 - JavaScript Zugriff auf alle Elemente der HTML-Seite, kann deren Attribute auslesen aber auch setzen sowie auch an beliebigen Stellen neue Elemente hinzufügen oder vorhandene entfernen
- Der Browser selbst
 - Es können Dialogfenster eingeblendet, neue Fenster geöffnet werden
- JavaScript kann Daten aus Cookies lesen oder in Cookies schreiben



Der Browser aus Sicht von JavaScript





Das Browser Object Model

- Dieses Modell liefert einen Zugriff sehr beschränkten Zugriff auf die darstellende HTML-Seite
 - Die ersten JavaScript-Anwendungen beschränkten sich häufig auf die Formular-Validierung und deshalb war auch nur der Zugriff auf das Formular notwendig
 - Alle anderen Bereiche der HTML-Seite, also zum Beispiel Absätze und Tabellen, sind hier nicht aufgenommen
- Für den Zugriff auf ein Element von document gelten folgende Regeln:
 - Die Attribute sind Arrays. Somit ist ein Zugriff über einen Index möglich.

```
window.document.forms[0].elements[0]
```

liefert das erste Element innerhalb des ersten Formulars.
 - Eine zwar einfache, aber nicht sonderlich wartbare Variante
 - Falls HTML-Elemente über ein gesetztes name-Attribut verfügen sind diese auch als Hash oder als Eigenschaft zugreifbar:

```
window.document.forms["inputForm"]["inputField"]  
window.document.forms.inputForm.inputField
```



Bestandteile des Browser Object Models

<code>window</code>	Anzeigefenster
<code>frames</code>	Frame-Fenster
<code>document</code>	Dokument im Anzeigefenster
<code>node</code>	Alle Knoten des Elementbaums
<code>all</code>	Alle HTML-Elemente des Dokuments (Microsoft)
<code>style</code>	CSS-Attribute von HTML Elementen
<code>anchors</code>	Verweisanker im Dokument
<code>applets</code>	Java-Applets im Dokument
<code>forms</code>	Formulare im Dokument
<code>elements</code>	Formularelemente eines Formulars
<code>options</code>	Optionen einer Auswahlliste eines Formulars



Bestandteile des Browser Object Models

<code>images</code>	Grafikreferenzen im Dokument
<code>embeds</code>	Multimedia-Referenzen im Dokument
<code>layers</code>	Layer im Dokument - Netscape
<code>links</code>	Verweise im Dokument
<code>event</code>	Anwenderereignisse
<code>history</code>	besuchte Seiten
<code>location</code>	URIs
<code>navigator</code>	Browser-Informationen
<code>mimeType</code>	MimeType-Informationen
<code>plugins</code>	installierte Plugins
<code>screen</code>	Bildschirm-Informationen



Das Document Object Model

- Das neuere DOM-Modelle ermöglicht den Voll-Zugriff auf alle Elemente eines Dokuments
- Dazu wird die HTML-Seite als XML-Dokument aufgefasst, der Zugriff auf alle Bestandteile der Seite erfolgt mit XML-Begriffen
 - Elemente und deren Attribute
 - Children, Parent, Siblings



Das Event-Modell



Events: DOM Level 0

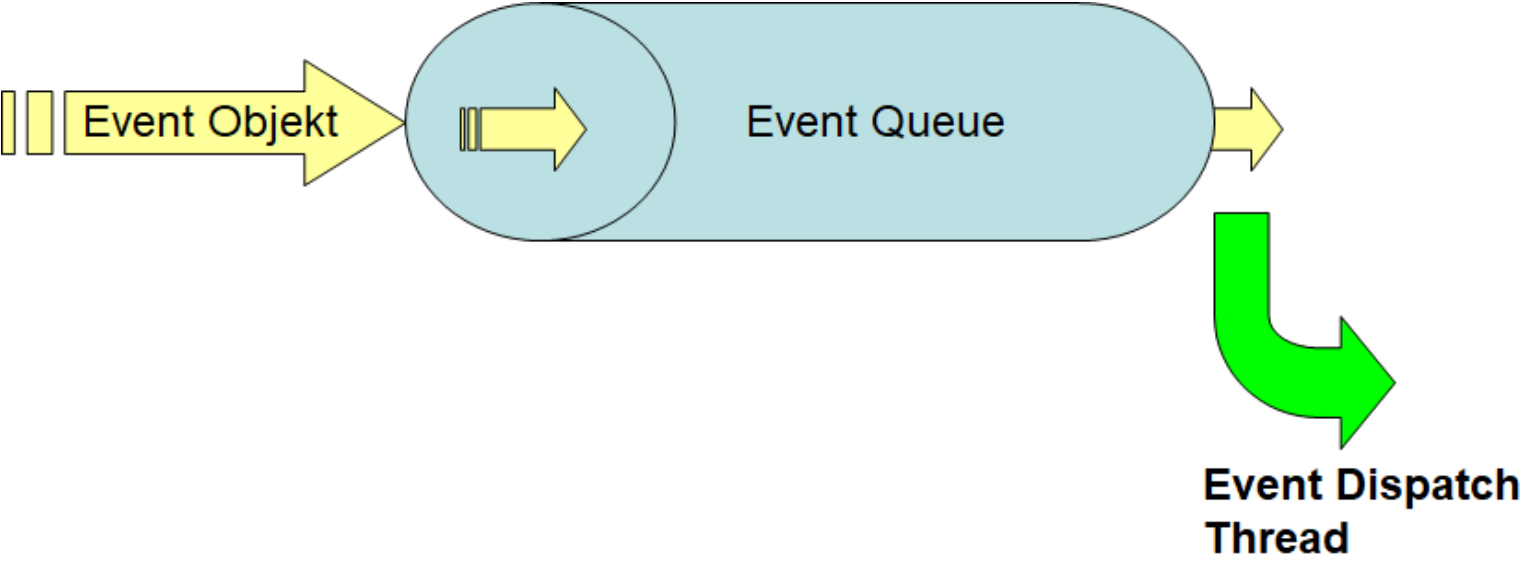
- Jedes HTML-Element hat einen festen Satz vordefinierter Event-Handler-Funktionen
- Führt der Benutzer eine Aktion aus so feuert die HTML-Komponente einen Event
 - Technisch gesehen ist die HTML-Komponente natürlich der Empfänger eines Benutzer-Initiierten Events
 - Dieser wird jedoch intern an die Handler weitergeleitet
- Der Zugriff auf dieses Event-Objekt erfolgt entweder über einen Funktions-Parameter oder über die `event`-Eigenschaft des aktuellen `window`s
 - Dies ist leider Browser-abhängig!



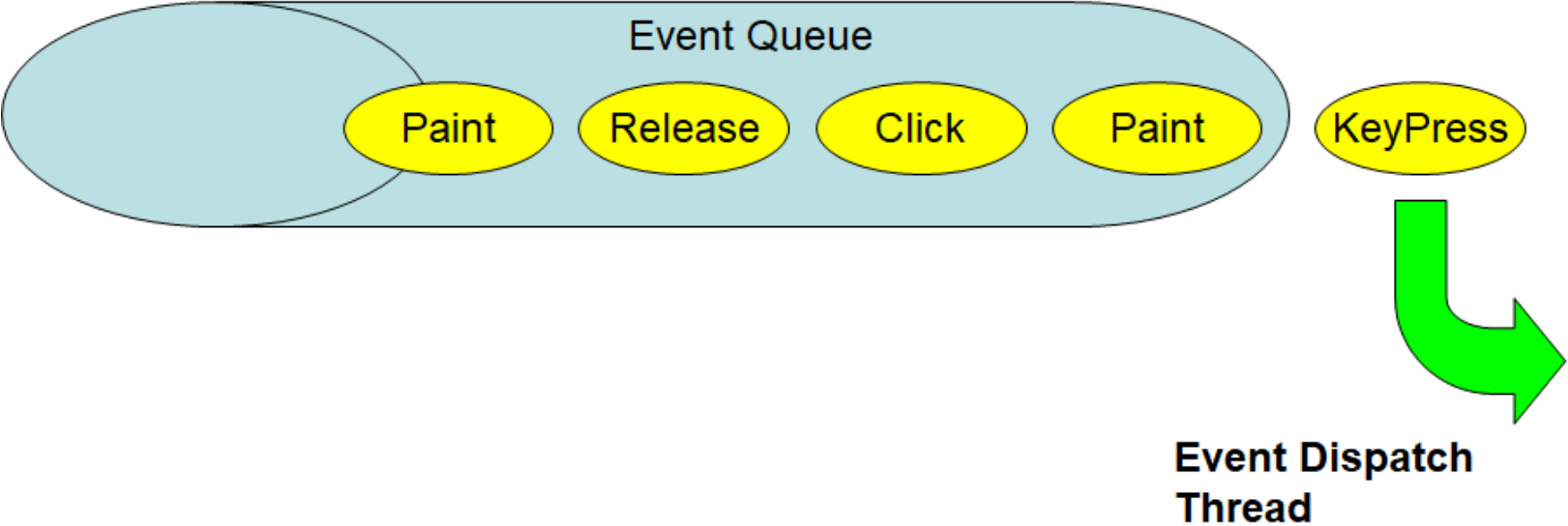
Event-Handler

- Die folgende Liste führt die Handler-Methoden von DOM Level 0 auf:
 - `onclick`
 - `ondblclick`
 - `onhelp`
 - `onkeydown`
 - `onkeypress`
 - `onkeyup`
 - `onmousedown`
 - `onmousemove`
 - `onmouseout`
 - `onmouseover`
 - `onmouseup`

Die Event-Queue



Befüllte Event Queue



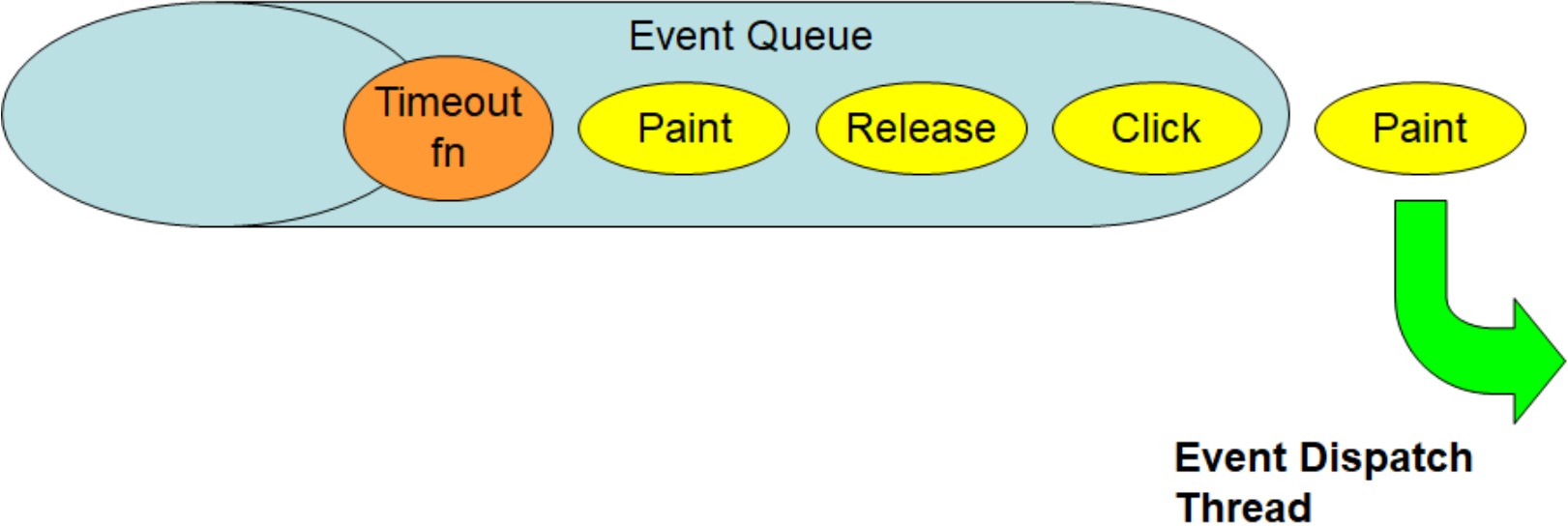


Timer

- Benutzung eines Timers
 - `setTimeout(fn, delay)`
 - Das übergebene Funktions-Objekt wird nach dem optionalen Delay als Event ausgeführt

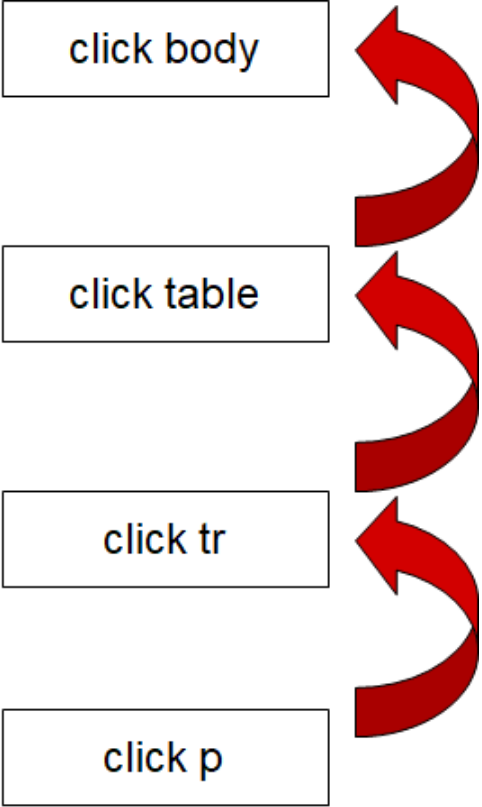
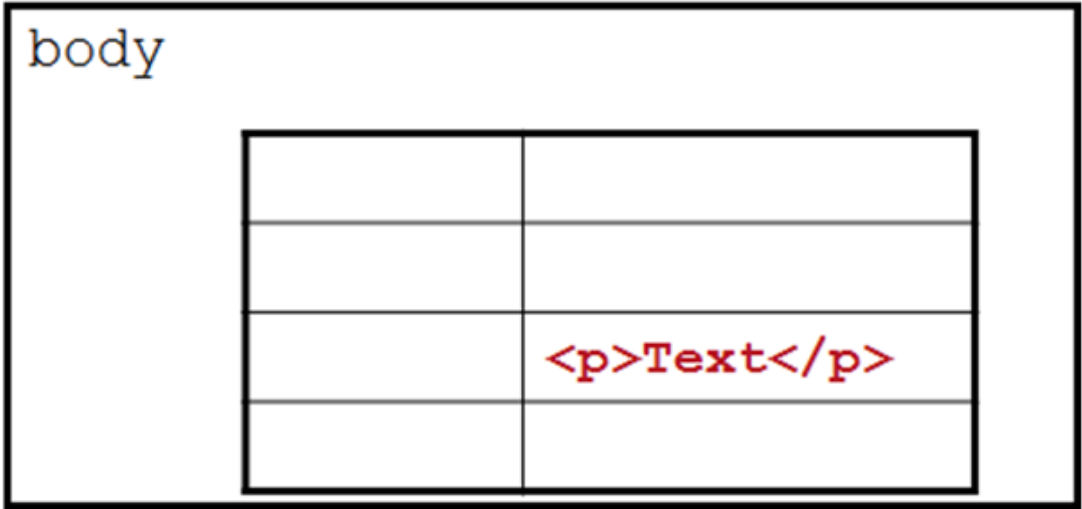


Event Queue und Timer





Event Bubbling





Event Bubbling

- Die Event-Verarbeitung wird standardmäßig auch für alle Eltern-Komponenten angestoßen
 - Schaltfläche – Formular – Paragraph – Body
- Soll dieses Bubbling unterbrochen werden genügt es, dass der Handler auf dem Event-Objekt die `cancelBubble`-Eigenschaft auf `true` setzt
 - Auch dies ist leider Browser-abhängig



DOM Level 2

- Im Unterschied zu den Level 0-Handlern können an alle Elemente mehrere Listener in eine Liste eingetragen werden:

```
addEventListener(eventType, listener, useCapture)
```

- Der `eventType` ist gleich dem Namen der entsprechenden Handler-Methode, jedoch ohne „on“.
- `listener` ist die auszuführende Event-Funktion
- `useCapture` ist ein logischer Wert der anzeigt, in welcher Phase der Listener aufgerufen werden soll
- Im DOM-Level 2-Modell wird vor der Bubble-Phase die Capture-Phase ausgeführt
 - Dies ist der Weg vom Root zum Event-Ziel, also genau umgekehrt wie in der Bubble-Phase
- DOM Level 2 wird leider noch nicht von allen Browsern einheitlich unterstützt



Window Events

(Kursiv: Neu in HTML5)

- `onafterprintNew`
- `onbeforeprintNew`
- `onbeforeunloadNew`
- `onerrorNew`
- `onhaschangeNew`
- `onload`
- `onmessageNew`
- `onofflineNew`
- `ononlineNew`
- `onpagehideNew`
- `onpageshowNew`
- `onpopstateNew`
- `onredoNew`
- `onresizeNew`
- `onstorageNew`
- `onundoNew`
- `onunload`



Beispielanwendungen



Referenzbeispiele



Beispiele der W3WSchool



Referenzbeispiele

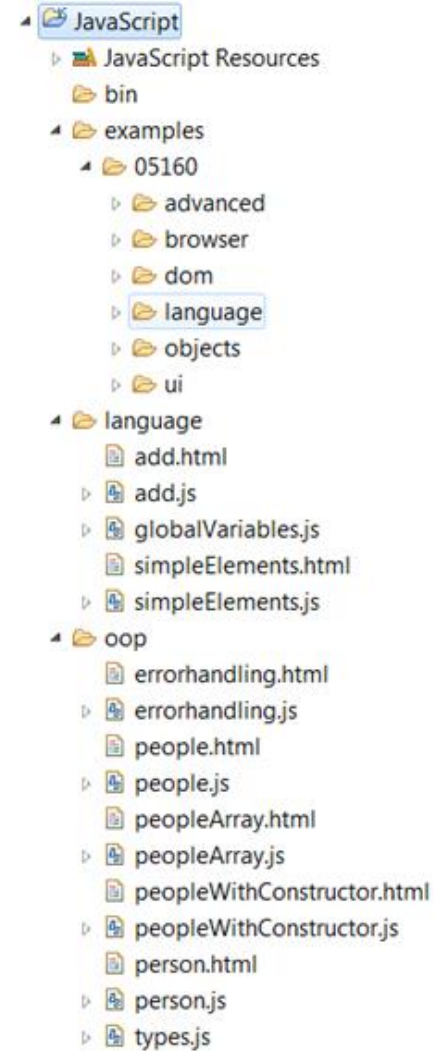


Praktikum

- Das Training enthält ein umfangreiches Praktikum
- Referenzlösungen werden in elektronischer Form zur Verfügung gestellt und sind frei verwendbar



Datei-Struktur der Beispiel- Anwendungen





Beispiele der W3WSchool



Übersicht

JS HTML DOM

DOM Intro

DOM Methods

DOM Document

DOM Elements

DOM HTML

DOM CSS

DOM Animations

DOM Events

DOM EventListener

DOM Navigation

DOM Nodes

DOM Nodelist

JS Browser BOM

JS Window

JS Screen

JS Location

JS History

JS Navigator

JS Popup Alert

JS Timing

JS Cookies

JS Examples

JS Examples

JS HTML DOM

JS HTML Input

JS HTML Objects

JS HTML Events

JavaScript HTML DOM Events Examples

[« Previous](#)

Examples of using JavaScript to react to events

Input Events

[onblur](#) - When a user leaves an input field

[onchange](#) - When a user changes the content of an input field

[onchange](#) - When a user selects a dropdown value

[onfocus](#) - When an input field gets focus

[onselect](#) - When input text is selected

[onsubmit](#) - When a user clicks the submit button

[onreset](#) - When a user clicks the reset button

[onkeydown](#) - When a user is pressing/holding down a key

[onkeypress](#) - When a user is pressing/holding down a key

[onkeyup](#) - When the user releases a key

[onkeyup](#) - When the user releases a key

[onkeydown vs onkeyup](#) - Both

Mouse Events

[onmouseover/onmouseout](#) - When the mouse passes over an element

[onmousedown/onmouseup](#) - When pressing/releasing a mouse button

[onmousedown](#) - When mouse is clicked: Alert which element

[onmousedown](#) - When mouse is clicked: Alert which button

[onmousemove/onmouseout](#) - When moving the mouse pointer over/out of an image

[onmouseover/onmouseout](#) - When moving the mouse over/out of an image

[onmouseover an image map](#)






TryIt-Editor

Edit This Code:

See Result »

Result:



```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    var x = document.getElementById("fname");
    x.value = x.value.toUpperCase();
}
</script>
</head>
<body>

Enter your name: <input type="text" id="fname"
onblur="myFunction()">

<p>When you leave the input field, a function is triggered
which transforms the input text to upper case.</p>

</body>
</html>
```

Enter your name:

When you leave the input field, a function is triggered which transforms the input text to upper case.



Weiterführende Themen



AJAX



JavaScript Frameworks



AJAX



Asynchrones JavaScript und XML (AJAX)

- AJAX steht als Abkürzung für „Asynchronous JavaScript And XML“ und wird innerhalb von Webanwendungen zum ereignisabhängigen und benutzergesteuerten Nachladen von Daten verwendet
- Mit AJAX ist es möglich Teile einer Webseite abhängig von Benutzeraktionen mit Daten vom Webserver zu aktualisieren

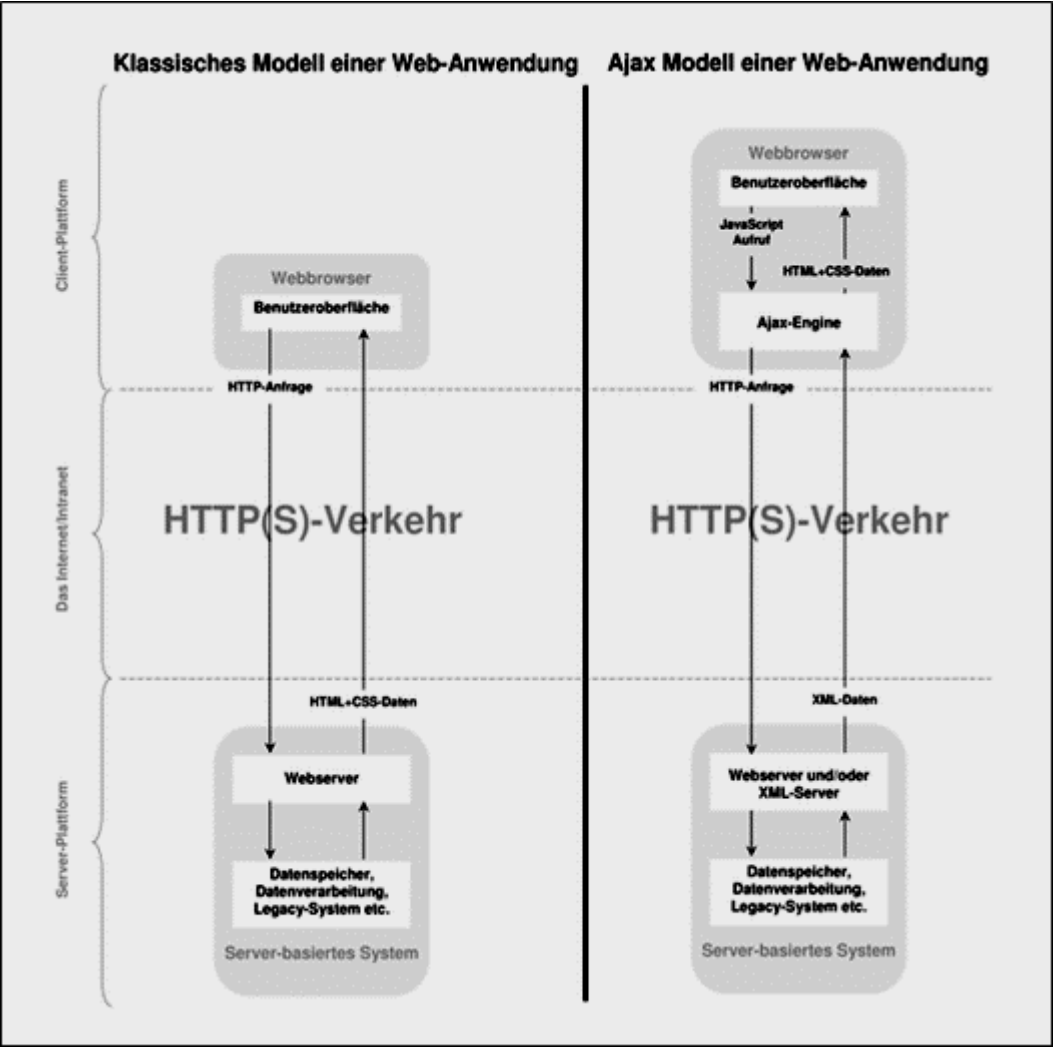


Komponenten einer AJAX-Anwendung

- AJAX als solches ist keine neue Technologie, sondern im Grunde nichts anderes als eine intelligente Kombination verschiedener Webtechnologien, ergänzt um ein Kommunikationsobjekt welches zum Datenaustausch verwendet wird
- AJAX besteht aus folgenden Webtechnologien:
 - XHTML für Inhalte und Layout
 - CSS für die Formatierung
 - DOM (Document Object Model) für Adressierung und Zugriff auf die einzelnen XML / XHTML – Dokumentbestandteile
 - JavaScript als clientseitige Programmiersprache über welche Kommunikation und Datenmanipulation ausgeführt werden
 - Das XMLHttpRequest-Objekt, bzw. MSXML.XMLHTTP-Objektes (Microsoft) über das die Kommunikation mit dem Webserver abläuft



Web Anwendungen mit AJAX





JavaScript Frameworks



JavaScript Frameworks

- JavaScript ist in den letzten Jahren immer beliebter geworden. Deshalb ist es nicht verwunderlich, dass auch hier immer mehr Bibliotheken entstanden sind, die vereinfachte und erweiterte Funktionalitäten anbieten
- Auf Grund der besonderen Natur von JavaScript sind diese Bibliotheken meistens frei verfügbar und Open Source
- Die folgende Auflistung stellt definitiv keinerlei Anspruch auf Vollständigkeit, es ist eine subjektive Auswahl einiger praxiserprobter Frameworks



Die Effekt-Bibliothek Scriptaculous

- Hierin sind eine ganze Reihe geläufiger Effekte (Einfliegen, Ausblenden, Vergrößern etc.) realisiert. In der Praxis wird niemand mehr eine Laufschrift selber programmieren, das Problem ist als gelöst zu betrachten.



DOM-Manipulation mit jQuery

- Die Beispiele zum Document Object Model sind relativ umständlich zu programmieren
- jQuery liefert eine deutliche Vereinfachung bei der Erzeugung von HTML-Elementen und unterstützt geläufige Abfragesprachen (XPath, CSS-Selektoren), um ganze Bereiche eines Dokuments einfach manipulieren zu können



Remoting

- JavaScript läuft zwar in der Regel nur auf dem Client, kann aber sehr wohl mit Server-Applikationen kommunizieren. So ist im Java-Umfeld das „Direct Web Remoting“ sehr beliebt, mit dem quasi direkt eine JavaScript-Funktion eine Serverseitige Java-Methode aufrufen kann
- Ähnliche Bibliotheken gibt es aber auch für .NET oder PHP



Generiertes JavaScript

- Sind Web Anwendungen von Serverseitigen Entwicklern zu erstellen wird die Komplexität und Mächtigkeit von JavaScript häufig unterschätzt
- Um den Einarbeitungsaufwand möglichst gering zu halten kann JavaScript auch automatisch erzeugt werden
- Beispiele hierfür sind
 - das Google Web Toolkit (GWT) oder
 - die JavaServer Faces (JSF)



Angular, React und Vue

- Die neuesten Player auf dem Markt der UI-Frameworks
- Stützen sich auf die HTML5-Spezifikation
 - Intensive Benutzung eigener HTML-Attribute
- Fokus auf Single Page Applications



ECMA2016



Neuerungen



Browser-Kompatibilität



Klassen



Scoped Variables und
Konstanten



Collections



Vereinfachte
Funktionsdeklaration



Generators und Proxies



Promises



Neuerungen



Highlights

- Klassen und Vererbung
- Generator Functions und Proxies
- Scoped Variables und Konstanten
- Neue Collection-Datentypen
 - Set
 - Map
- Vereinfachte Funktionslitterale mit Arrow Functions
- Promise-API



<http://es6-features.org>

ECMAScript 6 — New Features: Overview & Comparison

[Tweet](#) [Star](#) 1,652 [Fork me on GitHub](#)

Constants

Constants

Scoping

Block-Scoped Variables

Block-Scoped Functions

Arrow Functions

Expression Bodies

Statement Bodies

Lexical this

Extended Parameter Handling

Default Parameter Values

Rest Parameter

Spread Operator

Template Literals

String Interpolation

Custom Interpolation

Raw String Access

Extended Literals

Binary & Octal Literal

Unicode String & RegExp Literal

Enhanced Regular Expression

Regular Expression Sticky Matching

Enhanced Object Properties

Property Shorthand

Computed Property Names

Constants

Support for constants (also known as "immutable variables"), i.e., variables which cannot be re-assigned new content. Notice: this only makes the variable itself immutable, not its assigned content (for instance, in case the content is an object, this means the object itself can still be altered).

ECMAScript 6 — syntactic sugar: reduced | traditional

```
const PI = 3.141593
PI > 3.0
```

✓

ECMAScript 5 — syntactic sugar: reduced | traditional

```
// only in ES5 through the help of object properties
// and only in global context and not in a block scope
Object.defineProperty(typeof global === "object" ? global : window, "PI", {
  value: 3.141593,
  enumerable: true,
  writable: false,
  configurable: false
});
```

✗



Browser- Kompatibilität



Aktueller Stand

- Praktisch kein aktueller Browser unterstützt den kompletten ECMAScript-Standard
 - Anwendungen, die für das freie Internet konzipiert sind, müssen auch noch alte Versionen unterstützen
- Allerdings können einige Features von ECMA2016 auch in klassischem JavaScript abgebildet werden
 - Allerdings deutlich umständlicher
 - Beispiel: Klasse
 - Eine Klassendefinition kann durch das "Module-Pattern" umgesetzt werden



Transpilation

- Der Vorgang des Umsetzens von einer Script-Sprache in eine andere wird als "Transpilation" bezeichnet
- Im JavaScript-Umfeld existieren verschiedene Transpiler-Ansätze:
 - TypeScript
 - Eine eigene Programmiersprache mit ECMAScript-ähnlicher Syntax aber mit der Möglichkeit einer statischen Typisierung
 - Entwicklungsumgebungen können somit verbesserte Code-Assists anbieten
 - Zur Ausführung wird dann daraus JavaScript transpiliert
 - Babel
 - Transpiler nach JavaScript für verschiedene Sprachen
 - Benutzung
 - Im Build-Prozess wird JavaScript generiert und vom Server bereitgestellt
 - Einbinden des Babel-Scripts in der HTML-Seite des Browsers und Transpilation "on the fly"



Klassen



Einfache Klassen

```
class Book{
    constructor(isbn, title) {
        this.title = title;
        this.isbn = isbn;
    }
    get isbn() {
        return this.isbn;
    }
    get title() {
        return this.title;
    }
    set title(value) {
        this.title = value;
    }
    info() {
        return "Book: isbn=" + isbn + ", title=" + title;
    }
}
```



Vererbung

```
class SchoolBook extends Book{  
    constructor(isbn, title, topic){  
        super(isbn, title);  
        this.topic = topic;  
    }  
  
    info(){  
        return super.info + ", topic=" + topic;  
    }  
}
```



Scoped Variables und Konstanten



let und const

- `let` beschränkt den Gültigkeitsbereich einer Variable auf den deklarierenden Scope
 - Also beispielsweise einem Block einer Schleife
- `const` deklariert eine Konstante



Collections



Map

- Eine Map besteht aus key-value-Paaren
 - In anderen Sprachen als Dictionary oder assoziatives Array bezeichnet

```
map = new Map(); //oder mit Vorbelegung
```

```
map = new Map(['key1', 'value1'], ['key2',  
'value2']);
```

```
map.set('key', 'value');
```

```
map.get('key');
```

```
map.size;
```

```
map.clear();
```

- Iteration

```
for (let key of map.keys()) {}
```

```
for (let value of map.values()) {}
```




Set

- Eine Set besteht aus Unikaten
 - In anderen Sprachen als Dictionary oder assoziatives Array bezeichnet

```
var set = new Set();  
set.add("Hugo")  
set.add("Emil")  
set.add("Hugo")  
set.has("Hugo")  
set.size; //-> 2
```



Vereinfachte Funktionsdeklaration



Arrow-Syntax für Funktionen

- Eine vereinfachte Schreibweise für Funktions-Definitionen
 - beispielsweise für Parameter-Übergabe

```
(res) => console.log(res + " at " + new Date())
```



Generators und Proxies



Generators

- Generators sind spezielle Funktionen, die den Kontrollfluss an die aufrufende Funktion zurück delegieren
 - Dafür wird die `yield`-Funktion eingeführt

```
function* sampleGenerator() {  
    print('First');  
    yield("Hugo");  
    print('Second');  
};  
//...  
let gen = sampleGenerator();  
print(gen.next());  
print(gen.next());
```



Proxies

- Proxies erweitern ("dekorieren") bereits vorhandene Funktionen und Objekte
- Dieses Design-Pattern ist in untypisierten Sprachen sehr einfach umzusetzen

```
var handler = {  
    get: function (target, name)  
        return Reflect.get(target, name)},  
    apply: function (receiver, ...args) {  
        print("applying...")  
    }  
};  
//...  
obj = new Proxy(obj, handler);
```



Promises



Das Promise-API

- Das Promise-API ordnet verschachtelte Callback-Funktionen in eine Sequenz von Funktionsaufrufen

```
var p = new Promise(function(resolve, reject) {  
  setTimeout(() => resolve(4), 2000);});  
p.then((res) => { res += 2; console.log(res + "  
at " + new Date());});  
p.then((res) => console.log(res + " at " + new  
Date()))
```