

Internettechniken

HTML DOM

Prof. Dr. Jürgen Heym

Hochschule Hof

HTML DOM

W3C Document Object Model (DOM)

- Das Document Object Model (DOM) ist ein W3C-Standard.
- Das W3C-DOM definiert einen Standard für den Zugriff auf die Elemente eines Dokuments und deren Manipulation.

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

- Das DOM ist in drei Teile/Level aufgeteilt:
 - Core DOM Standardmodell für strukturierte Dokumente
 - XML DOM Standardmodell für XML-Dokumente
 - HTML DOM Standardmodell für HTML-Dokumente
- Das DOM definiert die Objekte und Eigenschaften aller Elemente des Dokuments und Methode, diese zu manipulieren.

HTML DOM

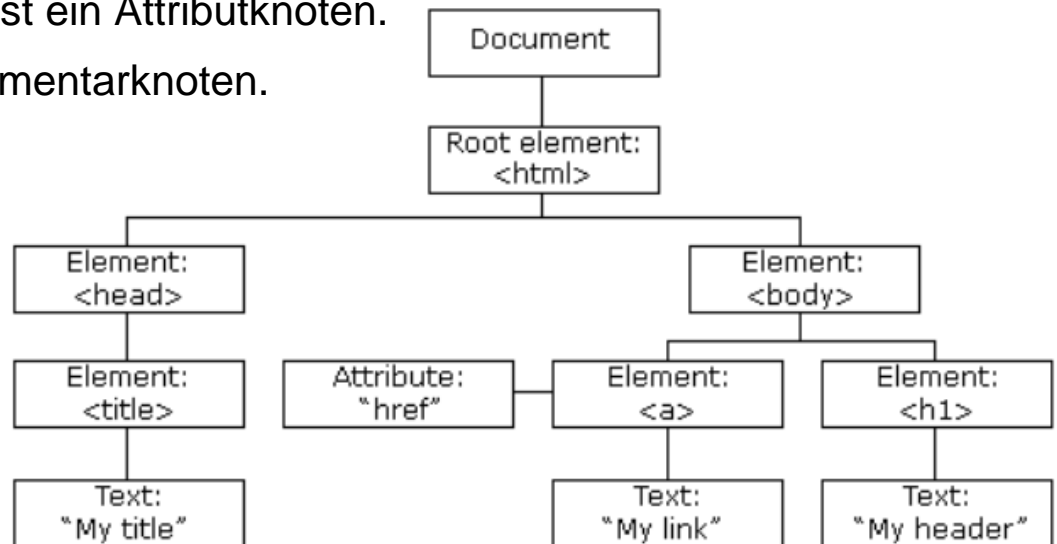
HTML Document Object Model (DOM)

- Das HTML DOM ist
 - ein Standard-Objektmodell für HTML-Dokumente,
 - eine Standard-Programmierschnittstelle für HTML-Dokumente ,
 - Unabhängig von der Plattform und der Programmiersprache und
 - ein W3C-Standard.
- Das HTML DOM definiert die Objekte und Eigenschaften aller HTML-Elemente des Dokuments und Methoden, um auf diese zuzugreifen.
- Das HTML DOM ist ein Standard um HTML-Elemente zu lesen, ändern, hinzuzufügen und zu löschen.

HTML DOM

HTML DOM Knoten

- Im HTML DOM dreht sich alles um Knoten:
 - Das Gesamtdokument ist ein Dokumentenknoten.
 - Jedes HTML-Element ist ein Elementknoten.
 - Die Texte in HTML-Elementen sind Textknoten.
 - Jedes HTML-Attribut ist ein Attributknoten.
 - Kommentare sind Kommentarknoten.



HTML DOM

HTML DOM Beispiel

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

- Der Wurzelknoten ist durch das html-Tag bestimmt.
- Alle anderen Knoten sind innerhalb des html-Tags.
- Der html-Knoten hat zwei Kinderknoten head und body.
- Der head-Knoten enthält einen title-Knoten.
- Der body-Knoten enthält einen h1- und einen p-Knoten.

HTML DOM

HTML DOM Beispiel

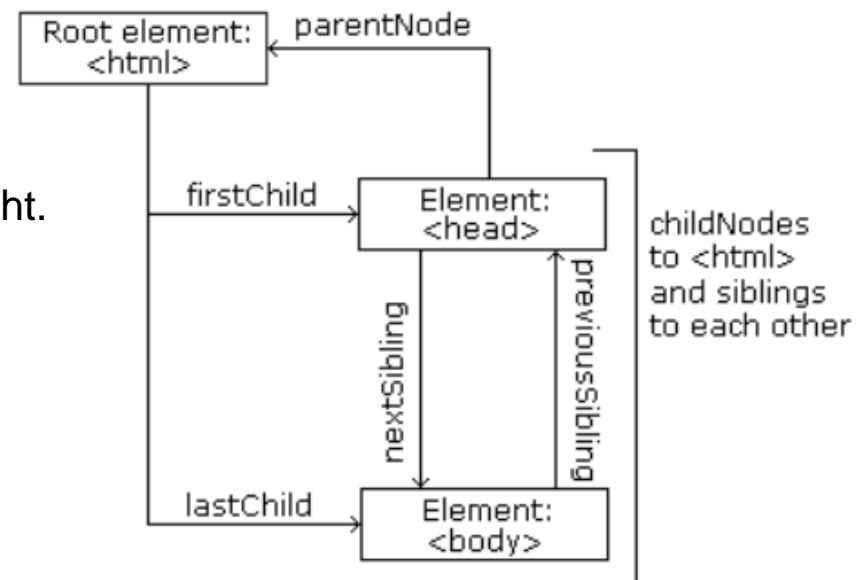
```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

- Ein Elementknoten enthält nie Text! Text wird immer in Textknoten abgelegt!
- In unserem Beispiel `<title>DOM Tutorial</title>` enthält der Elementknoten title einen Textknoten mit dem Wert "DOM Tutorial".
- "DOM Tutorial" ist nicht der Wert des title-Knotens!
- Im HTML DOM kann der Wert des Textknotens über die Eigenschaft `innerHTML` manipuliert werden.

HTML DOM

HTML DOM Knotenbaum

- Die Knoten in einem DOM-Knotenbaum haben eine hierarchische Beziehung zueinander.
- Man spricht von Eltern, Kindern und Geschwistern, um die Beziehung der Knoten zueinander zu beschreiben.
- Der Wurzelknoten wird auch als “top node” oder “root” bezeichnet.
- Jeder Knoten hat exakt einen Elternknoten, nur der Root-Knoten nicht.
- Jeder Knoten kann eine beliebige Anzahl von Kindknoten haben.
- Einen Knoten ohne Kindknoten bezeichnet man als Blatt (leaf).
- Geschwister sind Knoten mit gleichem Elternknoten.



HTML DOM

HTML DOM Eigenschaften und Methoden

- HTML DOM Eigenschaften
 - `x.innerHTML` Wert des Textknoten innerhalb eines Elementes `x`.
 - `x.nodeName` Knotenname
 - `x.nodeValue` Wert oder Inhalt des Knotens
 - `x.parentNode` Elternknoten zu Element `x`
 - `x.childNodes` Kindknoten zu Element `x`
 - `x.attributes` Attribute des Elements `x`
 - `x.firstChild` Erster Kindknoten des Elements `x`
 - `x.lastChild` Letzter Kindknoten des Elements `x`

- HTML DOM Methoden
 - `x.getElementById(id)` Element mittels seiner *id* als Objekt auslesen.
 - `x.getElementsByTagName(name)` Alle Element einer bestimmten Tag-Klasse auslesen.
 - `x.appendChild(node)` Einen neuen Kindknoten einfügen.
 - `x.removeChild(node)` Einen Kindknoten entfernen.

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel: innerHTML-Eigenschaft

```
<html>
<body>

<p id="intro">Hello World!</p>

<script type="text/javascript">

    txt=document.getElementById("intro").innerHTML;

    document.write("<p>The text from the intro paragraph: " + txt + "</p>");

</script>

</body>
</html>
```

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel: Auslesen aller p-Tag-Inhalte

...

```
x=document.getElementsByTagName("p");
```

```
for (i=0;i<x.length;i++)  
{  
    document.write(x[i].innerHTML);  
    document.write("<br />");  
}
```

...

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel: Eigenschaften firstChild und lastChild

```
<html>
<body>

<p id="intro">Hello World!</p>

<script type="text/javascript">
    x=document.getElementById("intro");
    document.write(x.firstChild.nodeValue);
</script>

</body>
</html>
```

HTML DOM

HTML DOM Eigenschaften und Methoden

- Spezielle Knoten
 - **document.documentElement**
referenziert den Root-Knoten des Dokuments.
 - **document.body**
referenziert das body-Tag eines Dokuments.

HTML DOM

HTML DOM Eigenschaften und Methoden

- Eigenschaften aller Knoten
 - Im HTML DOM ist jeder Knoten ein Objekt.
 - Objekte haben Methoden und Eigenschaften, die mittels JavaScript manipuliert werden können.
 - Drei wichtige Eigenschaften jedes Knotens sind:
 - **nodeName**
 - **nodeValue**
 - **nodeType**

- nodeName-Eigenschaft
 - Die nodeName-Eigenschaft bestimmt den Namen des Knotens.
 - Der nodeName ist nur lesbar (read-only).
 - Der nodeName eines Elements ist identisch zum Tag-Namen.
 - Der nodeName eines Attributs ist identisch zum Attributnamen.
 - Der nodeName eines Textknotens ist immer #text.
 - Der nodeName des Gesamtdokuments ist immer #document.

HTML DOM

HTML DOM Eigenschaften und Methoden

- nodeValue-Eigenschaft
 - Die nodeValue-Eigenschaft bestimmt den Inhalt des Knotens.
 - Die nodeValue-Eigenschaft ist für Elementknoten nicht definiert.
 - Die nodeValue-Eigenschaft eines Textknotens ist der Text selbst.
 - Die nodeValue-Eigenschaft für Attributknoten ist der Attributwert.

HTML DOM

HTML DOM Eigenschaften und Methoden

- nodeType-Eigenschaft
 - Die nodeType-Eigenschaft kann nur gelesen werden und bestimmt den Knotentyp.

Element type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel
 - Änderung der Hintergrundfarbe des body-Tags.

```
<html>  
<body>  
  
<script type="text/javascript">  
    document.body.backgroundColor="lavender";  
</script>  
  
</body>  
</html>
```

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel
 - Änderung des Inhalts des Elementes „p1“.

```
<html>
```

```
<body>
```

```
<p id="p1">Hello World!</p>
```

```
<script type="text/javascript">
```

```
    document.getElementById("p1").innerHTML="New text!";
```

```
</script>
```

```
</body>
```

```
</html>
```

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel
 - Änderung einer Eigenschaft bei Mausklick.

```
<html>
```

```
<body>
```

```
<input type="button"  
      onclick="document.body.bgColor='lavender' ;"  
      value="Change background color" />
```

```
</body>
```

```
</html>
```

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel
 - Änderung des Inhalts des Elementes „p1“ über sein style-Objekt.

```
<html>
<head>
<script type="text/javascript">
function ChangeBackground()
{
    document.body.style.backgroundColor="lavender";
}
</script>
</head>

<body>
<input type="button" onclick="ChangeBackground() "
    value="Change background color" />
</body>
</html>
```

HTML DOM

HTML DOM Eigenschaften und Methoden

- Beispiel
 - Änderung von Font und Farbe es Elements „p1“.

```
<html>
<head>
<script type="text/javascript">
function ChangeStyle()
{
    document.getElementById("p1").style.color="blue";
    document.getElementById("p1").style.fontFamily="Arial";
}
</script>
</head>

<body>
<p id="p1">Hello world!</p>
<input type="button" onclick="ChangeStyle() "
    value="Change style" />
</body>
</html>
```

HTML DOM

HTML DOM Ereignisse (events)

- Ereignisse (events)

Jedes Element einer Webseite hat bestimmte Möglichkeiten Ereignisse an JavaScript-Funktionen weiterzuleiten:

- Mausklick
- Tastendruck
- Laden einer neuen Seite oder eines Bildes.
- Bewegung der Maus über einen Hotspot.
- Auswahl eines Eingabefeldes.
- Abschicken eines Formulars.

Beispiel: `E-mail: <input type="text" id="email" onchange="checkEmail()" />`

Referenz: Siehe Vorlesung "Javascript 1".

HTML DOM

Beispiele

- Beispiel

Wie viele Anker enthält ein Dokument? Antwort: Nur Anker mit name-Attribut werden gezählt!

```
<html>
<body>
<a name="html">HTML Tutorial</a><br />
<a name="css">CSS Tutorial</a><br />
<a name="xml">XML Tutorial</a><br />
<a href="/js/">JavaScript Tutorial</a>

<p>Number of anchors:
<script type="text/javascript">
    document.write(document.anchors.length) ;
</script>
</p>
</body>
</html>
```


HTML DOM

Beispiele

- Beispiel

Rückgabe der Eigenschaft innerHTML des ersten Ankers.

```
<html>
<body>

<a id="html">HTML Tutorial</a><br />
<a id="css">CSS Tutorial</a><br />
<a id="xml">XML Tutorial</a>

<p>innerHTML of the first anchor:
<script type="text/javascript">
    document.write(document.anchors[0].innerHTML);
</script>
</p>

</body>
</html>
```

HTML DOM

Beispiele

- Beispiel

Wie viele Formulare enthält das aktuelle Dokument?

```
<html>
<body>

<form name="Form1"></form>
<form name="Form2"></form>
<form></form>

<p>Number of forms:
<script type="text/javascript">
    document.write (document.forms.length) ;
</script>
</p>

</body>
</html>
```

HTML DOM

Beispiele

- Beispiel

Wie heißt das erste Formular?

```
<html>
<body>

<form name="Form1"></form>
<form name="Form2"></form>
<form></form>

<p>Name of first form:
<script type="text/javascript">
document.write (document.forms [0] .name) ;
</script>
</p>

</body>
</html>
```

HTML DOM

Beispiele

- Beispiel

Wie viele Bilder sind im dokument referenziert?

```
<html>
<body>




<p>Number of images:
<script type="text/javascript">
    document.write(document.images.length) ;
</script>
</p>

</body>
</html>
```

siehe: http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_doc_images

HTML DOM

Beispiele

- Beispiel

Welche Cookies sind gesetzt?

```
<html>
```

```
<body>
```

Cookies associated with this document:

```
<script type="text/javascript">  
    document.write(document.cookie) ;  
</script>
```

```
</body>
```

```
</html>
```

HTML DOM

Beispiele

- Beispiel

Öffnen eines neuen Fensters und einfügen von Text in das neue HTML DOM.

```
<html>
<body>

<script type="text/javascript">
    var w=window.open();
    w.document.open();
    w.document.write("<h1>Hello World!</h1>");
    w.document.close();
</script>

</body>
</html>
```

HTML DOM

Beispiele

- ... und noch viel mehr Beispiele finden Sie hier:

`http://www.w3schools.com/jsref/dom_obj_document.asp`

Internettechniken

JavaScript Teil 1

Prof. Dr. Jürgen Heym

Hochschule Hof

JavaScript

Einleitung

- JavaScript ist eine der wichtigsten Script-Sprachen im Web.
- JavaScript wird auf unzähligen Webseiten für verschiedene Aufgaben eingesetzt:
 - Dynamische Funktionen
 - Formularvalidierung
 - Kommunikation zu einem Server

JavaScript

Einleitung

Was ist JavaScript?

- JavaScript wurde entwickelt, um HTML-Seiten Interaktivität hinzuzufügen.
- JavaScript ist eine Skriptsprache.
- JavaScript ist eine Leichtgewicht-Programmiersprache.
- JavaScript wird in HTML eingebettet.
- JavaScript wird von einem Interpreter ausgeführt, es gibt keinen JavaScript-Compiler.
- JavaScript ist lizenzfrei und kann von jedem eingesetzt werden.

JavaScript

Einleitung

JavaScript == ECMAScript

- JavaScript ist eine Implementation des ECMA Skriptsprachen Standards.
- *ECMA == European Computer Manufacturers Association.*
- JavaScript entspricht dem Standard ECMA-262.
- JavaScript wurde von Brendan Eich in Netscape Navigator 2.0 erstmals eingeführt und ist seit 1996 in allen Browsern verfügbar.
- Der offizielle ECMAScript-262 Standard wurde 1997 verabschiedet und 1998 als ISO 16262 übernommen.
- JavaScript befindet sich immer noch in der Weiterentwicklung.

JavaScript

Beispiele

Beispiel 1: Text in ein HTML-Dokument schreiben

```
<html>
<body>

<h1>My First Web Page</h1>

  <script type="text/javascript">
    document.write("<p>" + Date() + "</p>");
  </script>

</body>
</html>
```

JavaScript

Beispiele

Beispiel 2: HTML-Elemente überschreiben

```
<html>  
<body>
```

```
<h1>My First Web Page</h1>
```

```
<p id="demo"></p>
```

```
<script type="text/javascript">  
    document.getElementById("demo").innerHTML=Date();  
</script>
```

```
</body>  
</html>
```

JavaScript

Grundlagen

Manche Browser unterstützen kein JavaScript!

- Diese Browser stellen JavaScript als Seiteninhalt dar!
- JavaScript sollte daher in HTML-Kommentarzeichen eingebettet werden. Dies ist Teil des JavaScript-Standards.
- Beispiel:

```
...  
  <script type="text/javascript">  
    <!--  
      document.getElementById("demo").innerHTML=Date();  
    //-->  
  </script>  
...
```

Der Doppelslash in der vorletzten Zeile ist ein JavaScript-Kommentar, der verhindert, dass „-->“ ausgeführt wird.

JavaScript

Grundlagen

JavaScript kann an unterschiedlichen Stellen stehen:

1. im Header des HTML-Dokuments oder
2. Im Body des HTML-Dokuments .

Wir können eine unbegrenzte Anzahl JavaScripts in einem Dokument ausführen lassen.

JavaScripte können im Header und im Body oder in beiden stehen.

Häufig findet man JavaScript im Header oder ganz am Ende eines Dokuments im Body-Bereich.

Beispiel 3: HTML-Elemente überschreiben

```
<html>

<head>
  <script type="text/javascript">
    function displayDate()
    {
      document.getElementById("demo").innerHTML=Date();
    }
  </script>
</head>

<body>
<h1>My First Web Page</h1>
<p id="demo"></p>

<button type="button" onclick="displayDate()">Display Date</button>
</body>

</html>
```


JavaScript

Grundlagen

JavaScript kann auch in externe Dateien ausgelagert werden:

```
<html>
<head>
...
<script type="text/javascript" src="xxx.js"></script>
...
</head>
<body>
...
</body>
</html>
```

JavaScript

Grundlagen --- Statements

- JavaScript berücksichtigt Groß-/Kleinschreibung.
- Jedes JavaScript-Statement ist ein Kommando für den Browser.
- JavaScript-Statements enden normalerweise mit einem Strichpunkt, müssen aber nicht!
- Ein JavaScript ist eine Sequenz von JavaScript-Statements.
- Kommentare
 - einzeiliger Kommentar: `//`
 - Mehrzeilige Kommentare: `/* */`

JavaScript

Grundlagen --- Variablen

- JavaScript-Variablen
 - berücksichtigen Groß-/Kleinschreibung und
 - Müssen mit einem Buchstaben oder Unterstrich beginnen.

```
<html>
<body>
<script type="text/javascript">
    var firstname;
    firstname="Hege";
    document.write(firstname);
</script>
</body>
</html>
```

JavaScript

Grundlagen --- Variablen

- JavaScript-Variablendeklaration
 - Schlüsselwort: var
 - mit und ohne Wertzuweisung

```
var firstname;
```

```
var x=5;
```

```
var carname="volvo";
```

- Es gibt in JavaScript lokale globale Variablen
 - Lokale Variablen sind nur innerhalb einer Funktion gültig und werden am Ende der Funktion ungültig.
 - Globale Variablen sind funktionsübergreifend gültig und werden am Ende der Seitenverarbeitung ungültig.
 - Globale Variablen werden ohne das Schlüsselwort var deklariert.

```
var firstname;           // lokale Variable
x=5;                     // globale Variablen
carname="volvo";
```

JavaScript

Grundlagen --- Arithmetische Operatoren

- Mit der Vorbelegung $y=5$ gilt:

Operator	Description	Example	Result	
+	Addition	$x=y+2$	$x=7$	$y=5$
-	Subtraction	$x=y-2$	$x=3$	$y=5$
*	Multiplication	$x=y*2$	$x=10$	$y=5$
/	Division	$x=y/2$	$x=2.5$	$y=5$
%	Modulus (division remainder)	$x=y\%2$	$x=1$	$y=5$
++	Increment	$x=++y$	$x=6$	$y=6$
		$x=y++$	$x=5$	$y=6$
--	Decrement	$x=--y$	$x=4$	$y=4$
		$x=y--$	$x=5$	$y=4$

JavaScript

Grundlagen --- Zuweisungsoperatoren

- Mit der Vorbelegung $x=10$ und $y=5$ gilt:

Operator	Example	Same As	Result
=	$x=y$		$x=5$
+=	$x+=y$	$x=x+y$	$x=15$
-=	$x-=y$	$x=x-y$	$x=5$
=	$x=y$	$x=x*y$	$x=50$
/=	$x/=y$	$x=x/y$	$x=2$
%=	$x%=y$	$x=x\%y$	$x=0$

JavaScript

Grundlagen --- Zeichenketten


- Verkettung von Zeichenketten

```
txt1 = "What a very";  
txt2 = "nice day";  
txt3 = txt1 + " " + txt2;  
document.write(txt3);
```


JavaScript

Grundlagen --- Vergleichsoperatoren

- Mit der Vorbelegung `x=5` gilt:



Operator	Description	Example
<code>==</code>	is equal to	<code>x==8</code> is false <code>x==5</code> is true
<code>===</code>	is exactly equal to (value and type)	<code>x===5</code> is true <code>x==="5"</code> is false
<code>!=</code>	is not equal	<code>x!=8</code> is true
<code>></code>	is greater than	<code>x>8</code> is false
<code><</code>	is less than	<code>x<8</code> is true
<code>>=</code>	is greater than or equal to	<code>x>=8</code> is false
<code><=</code>	is less than or equal to	<code>x<=8</code> is true

JavaScript

Grundlagen --- Logische Operatoren

- Mit der Vorbelegung $x=6$ und $y=3$ gilt:

Operator	Description	Example
&&	and	$(x < 10 \ \&\& \ y > 1)$ is true
	or	$(x == 5 \ \ y == 5)$ is false
!	not	$!(x == y)$ is true

- Bedingte Zuweisung

```
variablename = (condition) ? value1 : value2
```

Beispiel:

```
greeting = (visitor=="PRES") ? "Dear President " : "Dear ";
```

JavaScript

Grundlagen --- Bedingte Verzweigung

- Bedingte Verzweigung

```
if (condition)
{
    code to be executed if condition is true
}
```

- Beispiel

```
<script type="text/javascript">
    // Write a "Good morning" greeting if the time is less than 10
    var d=new Date();
    var time=d.getHours();

    if (time<10)
    {
        document.write("<b>Good morning</b>");
    }
</script>
```

JavaScript

Grundlagen --- Bedingte Verzweigung

- Bedingte Verzweigung 2

```
if (condition)
{
    code to be executed if condition is true
}
else
{
    code to be executed if condition is not true
}
```

- Beispiel

```
if (time<10)
{    document.write("<b>Good morning!</b>");    }
else
{    document.write("<b>Good day!</b>");    }
```

- Bedingte Verzweigung 3

```
if (condition1)
{
  code to be executed if condition1 is true
}
else if (condition2)
{
  code to be executed if condition2 is true
}
else
{
  code to be executed if neither condition1 nor condition2 is true
}
```

JavaScript

Grundlagen --- Bedingte Verzweigung

- Beispiel mit Zufallslink

```
<html>
<body>

<script type="text/javascript">
  var r=Math.random();
  if (r>0.5)
  {
    document.write("<a href='http://www.fh-hof.de'>FH</a>");
  }
  else
  {
    document.write("<a href='http://www.uni-bt.de'>Uni BT</a>");
  }
</script>

</body>
</html>
```

JavaScript

Grundlagen --- Bedingte Verzweigung

- Switch-Statement

```
switch(n)
{
case 1:
    execute code block 1
    break;
case 2:
    execute code block 2
    break;
default:
    code to be executed if n is different from case 1 and 2
}
```

JavaScript

Grundlagen --- Bedingte Verzweigung

- Switch-Statement

```
<script type="text/javascript">
  var d=new Date();
  var theDay=d.getDay();
  switch (theDay)
  {
    case 5: document.write("Freitag");
            break;
    case 6: document.write("Samstag");
            break;
    case 0: document.write("Sonntag");
            break;
    default:
            document.write("Bald ist Wochenende!");
  }
</script>
```


JavaScript

Grundlagen --- PopUp-Boxen

- Alert-Popup

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
    alert("I am an alert box" + '\n' + "with line break!");
}
</script>
</head>
<body>

<input type="button" onclick="show_alert()" value="Show alert box" />

</body>
</html>
```

JavaScript

Grundlagen --- PopUp-Boxen

- Confirm-Popup

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
    var r=confirm("Knopf drücken!");
    if (r==true){ alert("OK!"); }
    else { alert("Cancel!"); }
}
</script>
</head>
<body>

<input type="button" onclick="show_confirm()" value="Bestätigung" />

</body>
</html>
```

JavaScript

Grundlagen --- PopUp-Boxen

- Prompt-Popup

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name", "Harry Potter");
if (name!=null && name!="")
    {
    document.write("Hello " + name + "! How are you today?");
    }
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript

Grundlagen --- Funktionen

- Syntax

```
function functionname(var1,var2,...,varX)
{
    Funktionsblock
}
```

- Beispiel

```
...
<script type="text/javascript">
    function product(a,b)
    {
        return a*b;
    }
</script>
</head>
<body>
    <script type="text/javascript">
        document.write(product(4,3));
    </script>
...

```

Beispiel

```
<html>
<head>
<script type="text/javascript">
    function myfunc(txt) {alert (txt);}
</script>
</head>
<body>

<form>
onclick="myfunc('Hello')" value="Call function">
</form>
```

<p>By pressing the button above, a function will be called with "Hello" as a parameter. The function will alert the parameter.</p>

```
</body>
</html>
```

- For-Loop

```
for (variable=start;variable<=end;variable=variable+increment)
{
  code to be executed
}
```

- Beispiel

```
...
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
  document.write("The number is " + i);
  document.write("<br />");
}
</script>
...
```

JavaScript

Grundlagen --- Schleifen

- While-Loop

```
while (variable<=endvalue)
{
    code to be executed
}
```

- Beispiel

```
...
<script type="text/javascript">
var i=0;
while (i<=5)
{
    document.write("The number is " + i);
    document.write("<br />");
    i++;
}
</script>
...
```

JavaScript

Grundlagen --- Schleifen

- Do-While-Loop

```
do
  {
    code to be executed
  }
while (variable <= endvalue);
```

- Beispiel

```
...
<script type="text/javascript">
var i=0;
do
  {
    document.write("The number is " + i);
    document.write("<br />");
    i++;
  }
while (i<=5);
</script>
```


- Schleifen unterbrechen

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
  {
    if (i==3)
      {
        break;
      }
    document.write("The number is " + i);
    document.write("<br />");
  }
</script>
</body>
</html>
```

- Schleifen beim nächsten Wert fortsetzen

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
  {
    if (i==3)
      {
        continue;
      }
    document.write("The number is " + i);
    document.write("<br />");
  }
</script>
</body>
</html>
```

- For-In-Schleife

Die for-in-Schleife iteriert über die Eigenschaften eines Objektes.

```
for (variable in object)
{
  code to be executed
}
```

- Beispiel

```
var person={fname:"John",lname:"Doe",age:25};

for (x in person)
{
  document.write(person[x] + " ");
}
```

JavaScript

Grundlagen --- Ereignisse

- Ereignisse können von JavaScript detektiert werden.

```
<html>
<head>
<script type="text/javascript">
function displayDate()
{
    document.getElementById("demo").innerHTML=Date();
}
</script>
</head>

<body>
<h1>My First Web Page</h1>
<p id="demo"></p>
<button type="button" onclick="displayDate()">Display Date</button>
</body>
</html>
```

JavaScript

Grundlagen --- Ereignisse

- Ereignisse sind
 - Mausklicks
 - Webseite oder Bild laden
 - Mausbewegung über einen Hotspot der Seite
 - Auswahl eines Eingabefeldes in einem Formular
 - Abschicken eines Formulars
 - Tastendruck
- Trigger
 - onLoad, onUnload
 - onFocus, onBlur, onChange
 - onSubmit
 - onMouseover

JavaScript

Grundlagen --- Ereignisse

IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** W3C Standard.

Attribute	The event occurs when...	IE	F	O	W3C
<u>onblur</u>	An element loses focus	3	1	9	Yes
<u>onchange</u>	The content of a field changes	3	1	9	Yes
<u>onclick</u>	Mouse clicks an object	3	1	9	Yes
<u>ondblclick</u>	Mouse double-clicks an object	4	1	9	Yes
<u>onerror</u>	An error occurs when loading a document or an image	4	1	9	Yes
<u>onfocus</u>	An element gets focus	3	1	9	Yes
<u>onkeydown</u>	A keyboard key is pressed	3	1	No	Yes
<u>onkeypress</u>	A keyboard key is pressed or held down	3	1	9	Yes
<u>onkeyup</u>	A keyboard key is released	3	1	9	Yes
<u>onload</u>	A page or image is finished loading	3	1	9	Yes
<u>onmousedown</u>	A mouse button is pressed	4	1	9	Yes
<u>onmousemove</u>	The mouse is moved	3	1	9	Yes
<u>onmouseout</u>	The mouse is moved off an element	4	1	9	Yes
<u>onmouseover</u>	The mouse is moved over an element	3	1	9	Yes
<u>onmouseup</u>	A mouse button is released	4	1	9	Yes
<u>onresize</u>	A window or frame is resized	4	1	9	Yes
<u>onselect</u>	Text is selected	3	1	9	Yes
<u>onunload</u>	The user exits the page	3	1	9	Yes

- Maus- und Tastaturereignisse

Property	Description	IE	F	O	W3C
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered	6	1	9	Yes
button	Returns which mouse button was clicked when an event was triggered	6	1	9	Yes
clientX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
clientY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered	6	1	9	Yes
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered	6	1	9	Yes
relatedTarget	Returns the element related to the element that triggered the event	No	1	9	Yes
screenX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
screenY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered	6	1	9	Yes

JavaScript

Grundlagen --- Fehlerbehandlung

- Try & Catch

```
try
  {
    //Run some code here
  }
catch(err)
  {
    //Handle errors here
  }
```


JavaScript

Grundlagen --- Fehlerbehandlung

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
    try
    {
        addAlert("Welcome guest!");
    }
    catch(err)
    {
        txt="There was an error on this page.\n\n";
        txt+="Error description: " + err.description + "\n\n";
        txt+="Click OK to continue.\n\n";
        alert(txt);
    }
}
</script>
</head>

<body>
    <input type="button" value="View message" onclick="message()" />
</body>
</html>
```

Debugging JavaScript

Joe Oakes

jxo19@psu.edu

Why Perform Debugging

- Helps you understand the coding syntax
- Helps you understand the code logic better
- Helps you understand the code and follow code pathways
- Helps you find bugs and logic errors
- Makes you a better programmer

```
<!DOCTYPE html>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
  <a id= "add" onclick="addTwoNumbers()" href="#test">add two numbers</a>
```

```
<script>
```

```
  function addTwoNumbers(var1, var2){
```

```
    return var1+var2;
```

```
  }
```

```
</script>
```

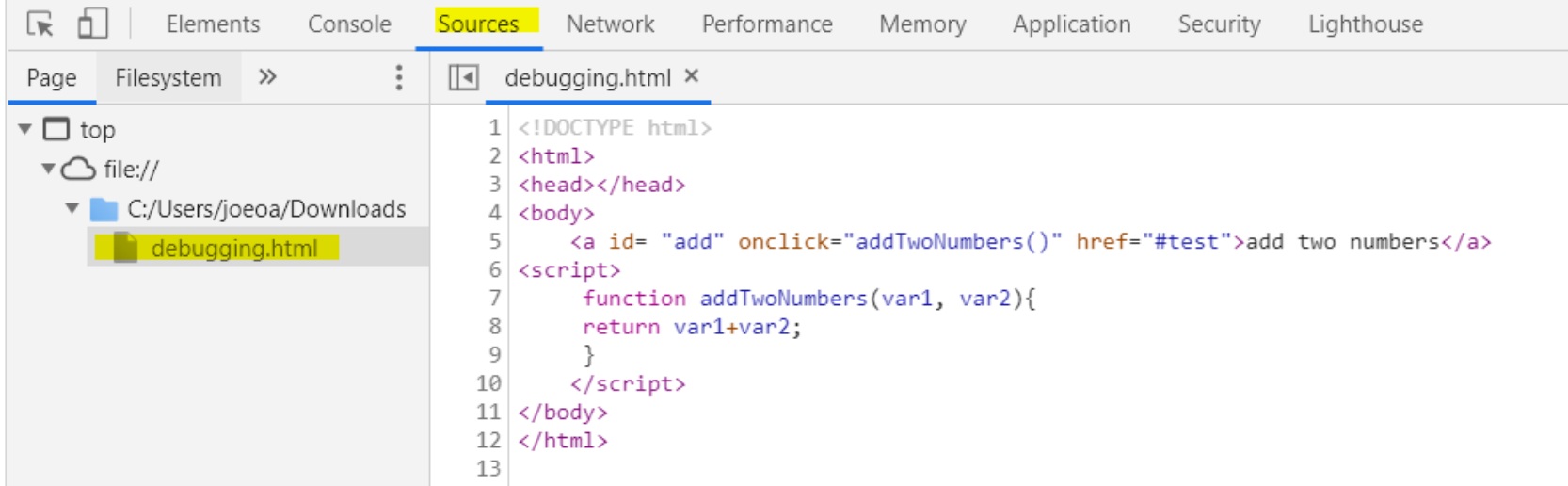
```
</body>
```

```
</html>
```

Source HTML and JavaScript code you can use to cut and paste into an editor

Open the Browser's Developer Tools
Control+Shift+I

[add two numbers](#)



The screenshot shows a browser window with the developer tools open. The 'Sources' tab is selected, displaying the source code of a file named 'debugging.html'. The code is as follows:

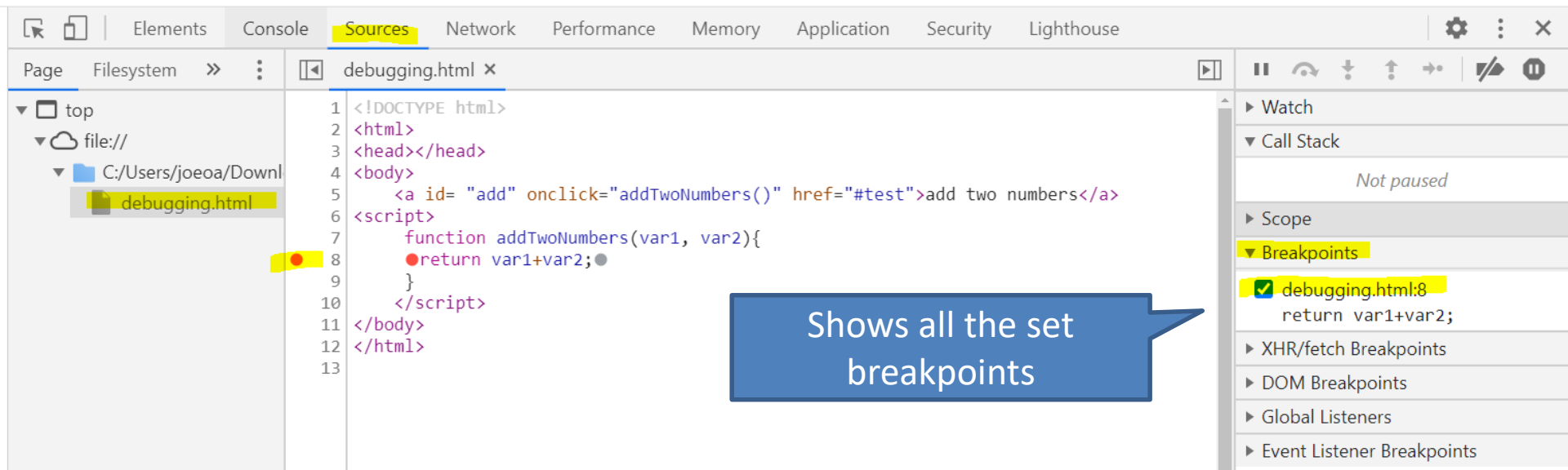
```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers()" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){
8     return var1+var2;
9   }
10  </script>
11 </body>
12 </html>
13
```

JavaScript Debugging Details

- **Breakpoints:** Set code execution points to stop
- **Instruction Pointer:** Current location in the code
- **Step over:** Step over a function it still executes
- **Step into:** Step into a function to trace through it
- **Step out:** Step out of a function
- **Call stack:** show what function calls have been push onto the stack
- **Watch:** watch variables/objects in the memory
- **Scope:** show variables in scope

Debugging Details: Breakpoint

- Breakpoint: Set code execution points to stop
 - Set the location by clicking on the gutter area
 - This can be toggled by clicking it again



Debugging Details: Breakpoint

- Reload the page and select the link
- Notice var1 and var2 are undefined variables

The screenshot shows the Chrome DevTools interface. The 'Sources' panel is open, displaying the code for 'debugging.html'. A red dot indicates a breakpoint is set at line 8, which is the 'return' statement of the 'addTwoNumbers' function. The function signature is 'function addTwoNumbers(var1, var2)'. The console shows 'var1 = undefined, var2 = undefined'. The call stack on the right shows the function was called from an 'onclick' event. The scope panel shows 'this' is 'Window' and 'var1' and 'var2' are 'undefined'.

var1 and var2 are undefined

```
<!DOCTYPE html>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
  <a id= "add" onclick="addTwoNumbers(2,4)" href="#test">add two numbers</a>
```

```
<script>
```

```
  function addTwoNumbers(var1, var2){
```

```
    return var1+var2;
```

```
  }
```

```
</script>
```

```
</body>
```

```
</html>
```

Added two input
argument values

debugging.html x

```
1 <!DOCTYPE html>
```

```
2 <html>
```

```
3 <head></head>
```

```
4 <body>
```

```
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
```

```
6 <script>
```

```
7   function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
```

```
8   ● return var1+var2; ●
```

```
9   }
```

```
10 </script>
```

```
11 </body>
```

```
12 </html>
```

```
13
```



Paused on breakpoint

Watch

Call Stack

addTwoNumbers

debugging.html:8

onclick

debugging.html:5

Scope

Local

this: Window

var1: 2

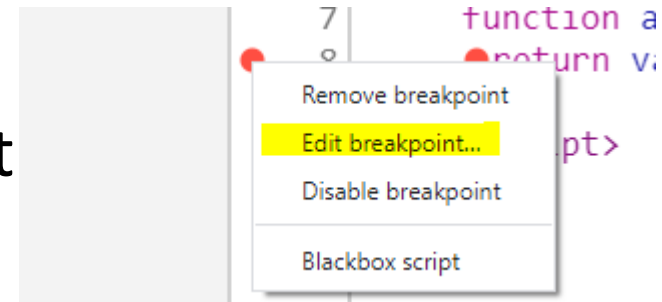
var2: 4

Global

Window

Debugging Details: Conditional Breakpoint

- Edit Breakpoint: right click
 - You can set conditional breakpoint

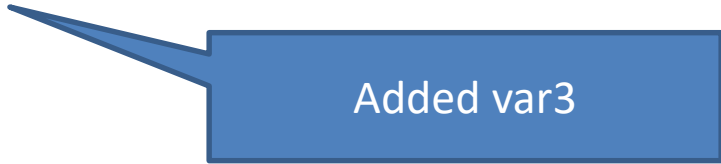


```
debugging.html x
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers()" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){
8     return var1+var2;
9   }
10 </script>
11 </body>
12 </html>
13
```

Line 8: Conditional breakpoint ▼

Expression to check before pausing, e.g. x > 5

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <a id= "add" onclick="addTwoNumbers(2,4)" href="#test">add two numbers</a>
<script>
  function addTwoNumbers(var1, var2){
    return var3 = var1+var2;
  }
</script>
</body>
</html>
```



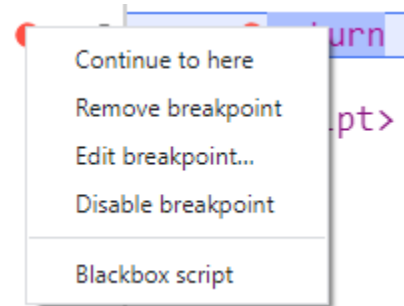
```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){
8     return var3 = var1+var2;
9   }
10 </script>
11 </body>
12 </html>
```

Line 8: Conditional breakpoint ▾

var3 > 5

Debugging Details: Conditional Breakpoint

- Conditional Breakpoint hit



The screenshot shows a browser's developer console with a code editor on the left and a debug console on the right. The code editor displays the following HTML and JavaScript code:

```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
8   return var3 = var1+var2;
9   }
10 </script>
11 </body>
12 </html>
13
```

The line `return var3 = var1+var2;` on line 8 is highlighted in yellow, indicating a breakpoint hit. The debug console on the right shows the following information:

- Paused on breakpoint** (highlighted in yellow)
- Watch**
- Call Stack**
 - addTwoNumbers debugging.html:8
 - onclick VM847 debugging.html:5
- Scope**
- Local**
 - this: Window
 - var1: 2
 - var2: 4
- Global** Window
- Breakpoints**
 - debugging.html:8 return var3 = var1+var2;

Debugging Details: Blackboxing

- Blackbox
 - When you don't know the internal workings of a given system
 - Blackboxing gives you a way to denote library (or other abstraction) code so that the debugger can route around it.

Continue to here
Remove breakpoint
Edit breakpoint...
Disable breakpoint

Blackbox script

Framework Blackboxing

Debugger will skip through the scripts and will not stop on exceptions thrown by them.

Debugging Details: Controls

- Debugger Controls



- **Resume:** Continue to the next breakpoint
- **Step over:** Step over a function it still executes
- **Step into:** Step into a function to step through it
- **Step out:** Step out of a function
- **Step:** Step to the next line
- **Deactivate Breakpoints**
- **Pause on exceptions**

Debugging Details: Examine Values

- You can use the Console to examine or change values

The screenshot displays the Chrome DevTools interface with the Sources panel open to a file named `debugging.html`. The code contains a function `addTwoNumbers` with a breakpoint set on line 8, which is the `return` statement. A blue callout box points to this line with the text: "This line has not been executed so var3 is not defined yet". The right-hand sidebar shows the "Paused on breakpoint" state, with the "Call Stack" and "Scope" panels visible. The "Scope" panel shows local variables `var1: 2` and `var2: 4`. The bottom console shows an error: "Uncaught ReferenceError: var3 is not defined" at the `eval` function, which was triggered by the `onclick` event on the `add` button.

```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
8   ●return var3 = var1+var2;●
9   }
10 </script>
11 </body>
12 </html>
13
```

Paused on breakpoint

Call Stack

- addTwoNumbers debugging.html:8
- onclick VM45 debugging.html:5

Scope

- Local
 - this: Window
 - var1: 2
 - var2: 4
- Global Window

Breakpoints

Line 8, Column 3 Coverage: n/a

Console

```
> var3
Uncaught ReferenceError: var3 is not defined
at eval (eval at addTwoNumbers (debugging.html:1), <anonymous>:1:1)
at addTwoNumbers (debugging.html:8)
at HTMLAnchorElement.onclick (VM45 debugging.html:5)
```

Debugging Details: Examine Values

- Use the Step icon to execute the line
- Clear the Console using the icon



The screenshot shows the Chrome DevTools interface with the Sources panel open to a file named `debugging.html`. The code is as follows:

```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
8   return var3 = var1+var2;
9   }
10 </script>
11 </body>
12 </html>
13
```

The function call on line 5 and the function definition on lines 7-9 are highlighted. The return statement on line 8 is also highlighted. The status bar at the bottom indicates the current position is at Line 8, Column 27.

The right-hand side of the interface shows the 'Debugger paused' state. The 'Call Stack' panel shows the current call to `addTwoNumbers` at line 8 of `debugging.html`, triggered by an `onclick` event at line 5. The 'Scope' panel shows the local variables `var1: 2` and `var2: 4`, and the `Return value: 6`.

The bottom of the image shows the Console panel with the command `> var3` entered, and the result `< 6` displayed below it.

Debugging Details: Watch

- You can use the Watch feature to watch a variable
- Click on the + to select the variable to watch

The screenshot displays the Chrome DevTools interface. The main editor shows the following HTML and JavaScript code:

```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
8     return var3 = var1+var2;
9   }
10 </script>
11 </body>
12 </html>
13
```

The 'Debugger paused' panel on the right shows the following details:

- Watch:** var3: 6
- Call Stack:** addTwoNumbers debugging.html:8, onclick VM45 debugging.html:5
- Scope:** Local, Return value: 6, this: Window, var1: 2, var2: 4

A blue callout box with the text "Use the + to add an expression" points to the '+' icon in the Watch panel.

> var3

<- 6

Debugging Details: Assignment

- You can change the value of a variable in the console

The screenshot displays the Chrome DevTools interface during a debugging session. The Sources panel shows the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head></head>
4 <body>
5   <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
6 <script>
7   function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
8   return var3 = var1+var2;
9   }
10 </script>
11 </body>
12 </html>
13
```

The Watch panel on the right shows the following variables:

- var3: 7

The Console panel at the bottom shows the following commands and output:

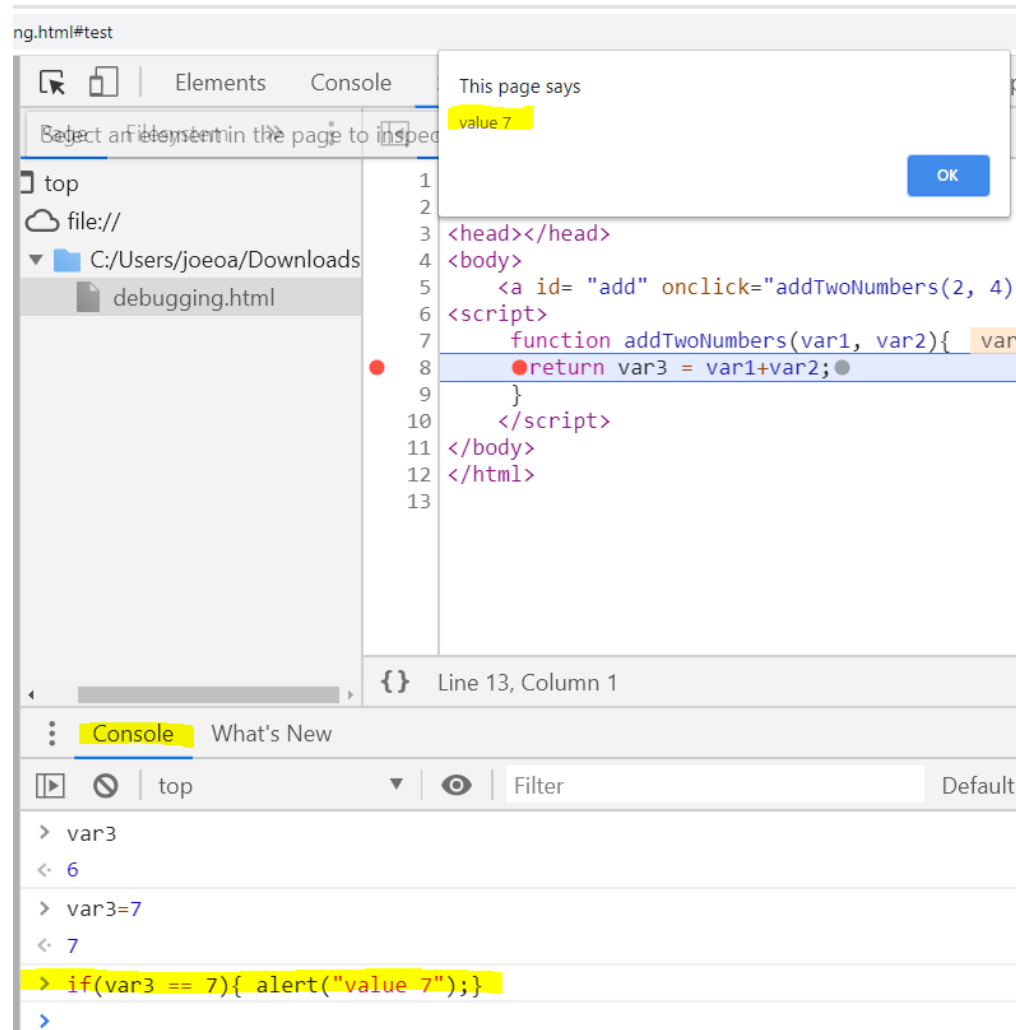
```
> var3
< 6
> var3=7
< 7
```

Two blue callout boxes provide additional context:

- A callout box pointing to the Watch panel contains the text: "Make sure to refresh".
- A callout box pointing to the Console panel contains the text: "Using the assignment operator changed the value from 6 to 7".

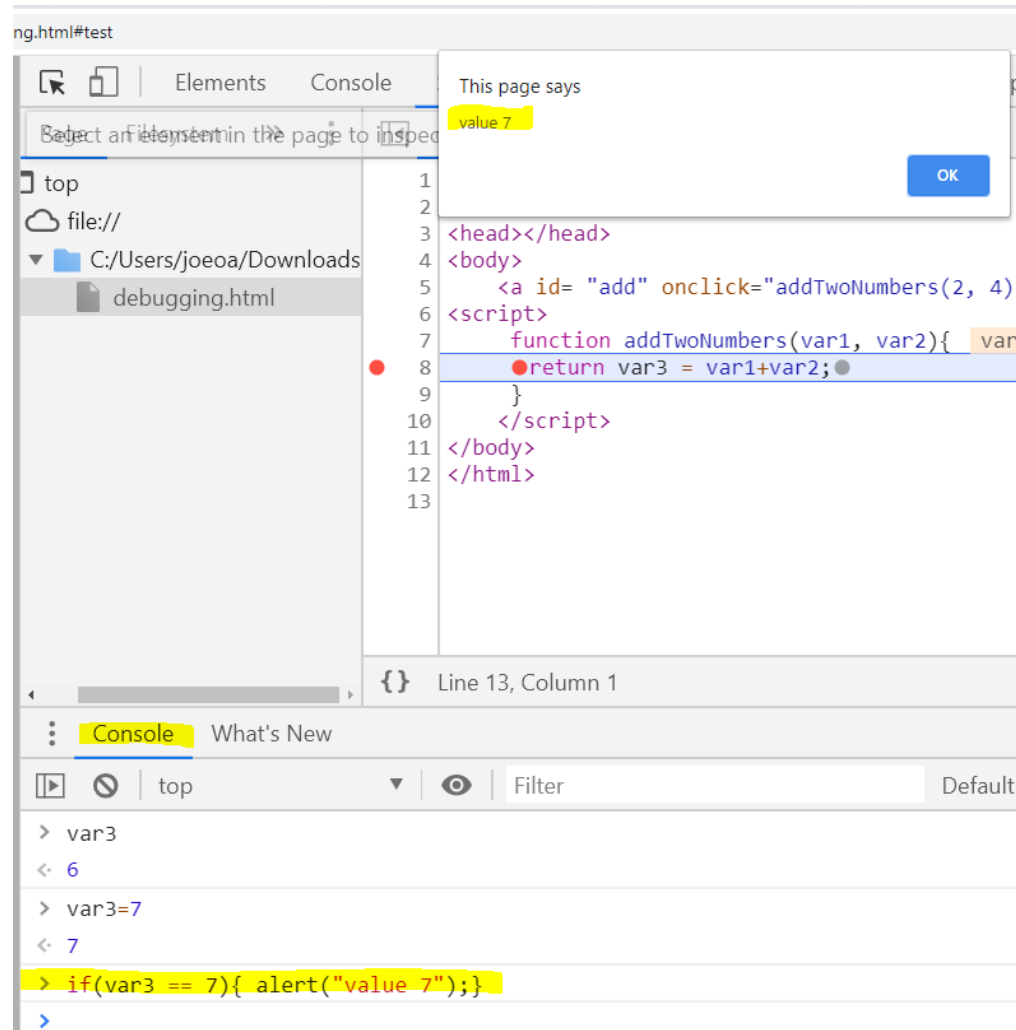
Debugging Details: Console

- You can put Javascript code in the console to execute



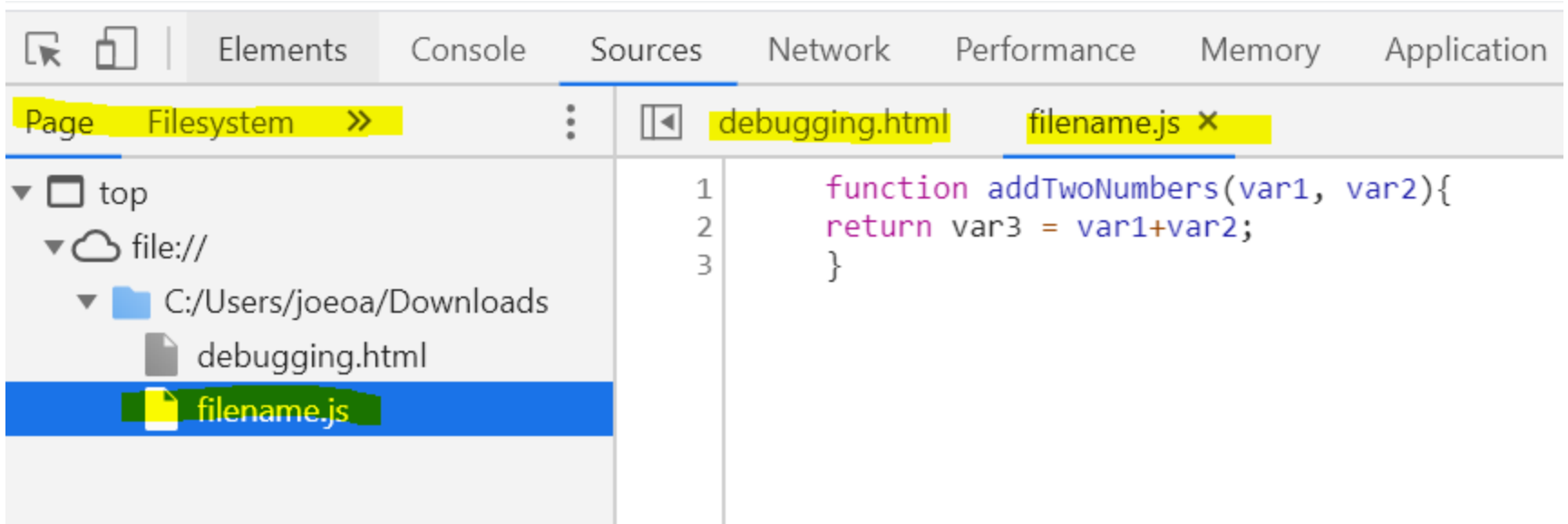
Debugging Details: Console

- You can put Javascript code in the console to execute



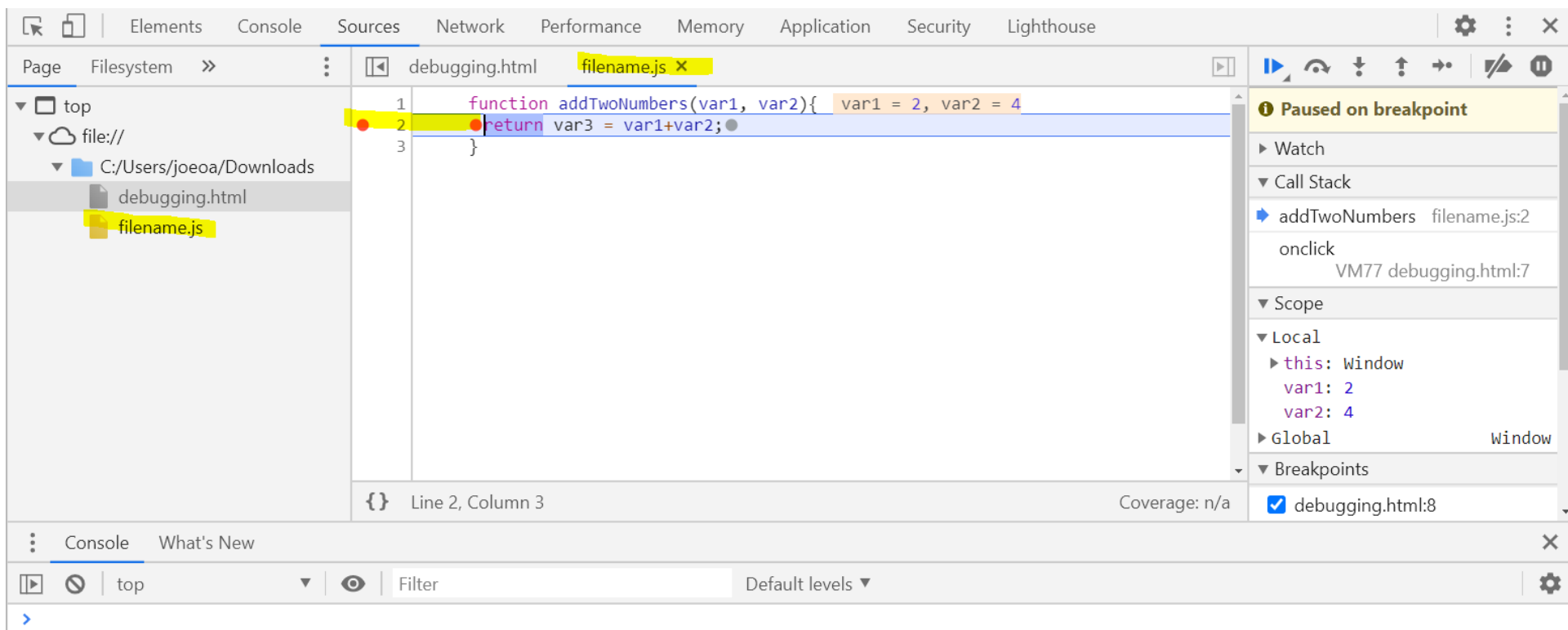
Debugging Details: External File

- Select Page -> Filesystem -> filename.js



Debugging Details: External File

- Select Page -> Filesystem -> filename.js
- Set your breakpoints in the gutter area



Debugging Details: Breakpoints

- You have many breakpoints set
 - You can manage them in the Breakpoints area

The screenshot shows a web browser's developer tools interface. The main pane displays the source code for 'filename.js' with two functions: 'addTwoNumbers' and 'subTwoNumbers'. Both functions have a red dot breakpoint on their respective 'return' statements. The right-hand sidebar contains several panels: 'Watch' (showing 'total: <not available>'), 'Call Stack' (showing 'Not paused'), 'Scope' (showing 'Not paused'), and 'Breakpoints'. The 'Breakpoints' panel is expanded and shows two active breakpoints: 'filename.js:2' and 'filename.js:5', both with checkboxes checked and the corresponding code lines highlighted in yellow.

```
1 function addTwoNumbers(var1, var2){
2   return var3 = var1+var2;
3 }
4 function subTwoNumbers(var1, var2){
5   return var3 = var1-var2;
6 }
7
```

▼ Watch + ↻
total: <not available>

▼ Call Stack
Not paused

▼ Scope
Not paused

▼ Breakpoints

- filename.js:2
return var3 = var1+var2;
- filename.js:5
return var3 = var1-var2;

Debugging Details: Breakpoints

- When the breakpoint is hit notice the area will turn yellow

```
1 function addTwoNumbers(var1, var2){
2   return var3 = var1+var2;
3 }
4 function subTwoNumbers(var1, var2){ var1 = 4, var2 = 4
5   return var3 = var1-var2;
6 }
7
```

Paused on breakpoint

Watch

Call Stack

- subTwoNumbers filename.js:5
- onclick VM188 debugging.html:8

Scope

Breakpoints

- filename.js:2
return var3 = var1+var2;
- filename.js:5
return var3 = var1-var2;

Breakpoint hit

JavaScript: Variable Scope

- **Variable scope:** what is the value of a specific variable name at the current line of code being executed
- JavaScript allows for both **global** and **local** scope of a variable
- **Global:** defined in the main JavaScript
- **Local:** defined in a function a new variable is created in memory
- Outside of the function referencing global

```
1 function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
2 return var3 = var1+var2;
3 }
```

Paused on breakpoint

Watch

Call Stack

addTwoNumbers filename.js:2

onclick

VM77 debugging.html:7

Scope

Local

this: Window

var1: 2

var2: 4

Global

Window

Local variables

JavaScript: Variable Scope

- A variable can either be global or local
 - **Global variable** can be referenced from anywhere in the script
 - **Local variable** only exists with the function in which it is declared

The screenshot displays a JavaScript debugger interface. The main window shows a code editor with the following code:

```
1 var total = 50;
2 function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
3 return var3 = var1+var2;
4 }
```

A breakpoint is set at line 3. The right-hand pane shows the 'Paused on breakpoint' state. The 'Watch' pane contains the expression `total: 50`. The 'Call Stack' pane shows the function `addTwoNumbers` at `filename.js:3`. The 'Scope' pane shows the local variables `var1: 2` and `var2: 4`, and the global scope containing `total: 50`. A blue callout box points to the global scope with the text: "Global variables Vertical Scroll down to see total".

Below the code editor, the console shows the following output:

```
status: ""
statusbar: BarProp {visi...
stop: f stop()
styleMedia: StyleMedia {...
toolbar: BarProp {visibl...
top: Window {parent: Win...
total: 50
trustedTypes: TrustedTyp...
```

JavaScript: Variable Scope

- myVar on line 2 is a global variable
- myVar on line 5 is a local variable
- Notice line 9 will use the global variable value since it is in scope

```
01 <script>
02   var myVar = 1;
03   function writeIt(){
04     var myVar = 2;
05     document.write(myVar);
06     writeMore();
07   }
08   function writeMore(){
09     document.write(myVar);
10   }
11 </script>
```

JavaScript: Log and line of code Breakpoint

- debugger allows you set in the code where the breakpoint will be hit
- console.log() allows you write to the console area as the code is executing

The image shows a code editor window with two tabs: 'debugging.html' and 'filename.js'. The code in 'filename.js' is as follows:

```
1 function addTwoNumbers(var1, var2){ var1 = 2, var2 = 4
2   var3 = var1+var2;
3   debugger;
4   return var3;
5 }
```

The 'debugger;' statement on line 3 is highlighted with a blue background, indicating a breakpoint. Below the code editor, the browser's developer tools are open to the 'Sources' tab. The file explorer on the left shows the file structure, including 'debugging.html' and 'filename.js'. The code editor in the 'Sources' tab shows the same code as above, but with a yellow highlight on the 'console.log("var3:" + var3);' statement on line 8. The status bar at the bottom indicates the current position is 'Line 3, Column 4'. The 'Console' tab is also visible, showing the output 'var3:0'.

JavaScript: Console Log

- debugger allows you set in the code where the breakpoint will be hit
- console.log() allows you write to the console area as the code is executing

The screenshot displays a web browser's developer console. The top-left pane shows the file explorer with 'debugging.html' and 'filename.js'. The top-right pane shows the JavaScript code with line numbers 1 through 19. The code defines three functions: 'addTwoNumbers', 'subTwoNumbers', and 'ajax'. The 'subTwoNumbers' function is highlighted in yellow, showing 'console.clear()', 'console.log("var3:" + var3)', 'console.log(document.head)', and 'console.log(document.body)'. The 'ajax' function is also highlighted in yellow, showing 'var xhr = new XMLHttpRequest()', 'xhr.open('GET', 'https://reqres.in/api/users', true)', and 'xhr.send()'. The bottom pane shows the console output, which includes 'Console was cleared', 'var3:0', and the DOM structure of the page, including the head and body elements with buttons for 'add two numbers', 'sub two numbers', and 'Ajax request'.

```
1 function addTwoNumbers(var1, var2){
2   var3 = var1+var2;
3   debugger;
4   return var3;
5 }
6 function subTwoNumbers(var1, var2){
7   var3 = var1-var2;
8   console.clear();
9   console.log("var3:" + var3);
10  console.log(document.head);
11  console.log(document.body);
12  return var3;
13 }
14 function ajax(){
15   var xhr = new XMLHttpRequest();
16   xhr.open('GET', 'https://reqres.in/api/users', true);
17
18   xhr.send();
19 }
```

Line 10, Column 5

Console What's New

top Filter Default levels

Console was cleared

var3:0

```
<head>
  <script type="text/javascript" src="filename.js"></script>
</head>
<body>
  <a id="add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a>
  <br>
  <a id="sub" onclick="subTwoNumbers(4, 4)" href="#test">sub two numbers</a>
  <br>
  <a id="ajax" onclick="ajax()" href="#test">Ajax request</a>
</body>
```

JavaScript: AJAX

The `XMLHttpRequest` method `send()` sends the request to the server. If the request is asynchronous (which is the default), this method returns as soon as the request is sent and the result is delivered using events. If the request is synchronous, this method doesn't return until the response has arrived.

```
debugging.html  filename.js x
1  function addTwoNumbers(var1, var2){
2  return var3 = var1+var2;
3  }
4  function subTwoNumbers(var1, var2){
5  return var3 = var1-var2;
6  }
7  function ajax(){
8      var xhr = new XMLHttpRequest();
9      xhr.open('GET', 'https://reqres.in/api/users', true);
10
11     xhr.send();
12
13     xhr.onload = function() {
14         let responseObj = xhr.response;
15         alert(responseObj);
16     };
17 }
```

```
/<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
</head>
```

```
<body>
```

```
  <a id= "add" onclick="addTwoNumbers(2, 4)" href="#test">add two numbers</a><br />
```

```
    <a id= "sub" onclick="subTwoNumbers(4, 4)" href="#test">sub two numbers</a><br />
```

```
    <a id= "ajax" onclick="ajax()" href="#test">Ajax request</a>
```

```
</body>
```

```
</html>
```

```
function addTwoNumbers(var1, var2){  
  return var3 = var1+var2;  
}
```

```
function subTwoNumbers(var1, var2){  
  return var3 = var1-var2;  
}
```

```
function ajax(){  
  var xhr = new XMLHttpRequest();  
  xhr.open('GET', 'https://reqres.in/api/users', true);  
  
  xhr.send();  
  
  xhr.onload = function() {  
    let responseObj = xhr.response;  
    alert(responseObj);  
  };  
}
```

HTML for AJAX

JavaScript code for AJAX

JavaScript: AJAX

- Notice the three xhr methods: open, send, onload

```
1 function addTwoNumbers(var1, var2){
2   return var3 = var1+var2;
3 }
4 function subTwoNumbers(var1, var2){
5   return var3 = var1-var2;
6 }
7 function ajax(){
8   var xhr = new XMLHttpRequest();
9   xhr.open('GET', 'https://reqres.in/api/users', true);
10
11  xhr.send();
12
13  xhr.onload = function() {
14    let responseObj = xhr.response;
15    alert(responseObj);
16  };
17 }
18
```

Paused on breakpoint

- ▶ Watch
- ▶ Call Stack
- ▶ Scope
- ▶ Breakpoints
- ▼ XHR/fetch Breakpoints +
No breakpoints
- ▶ DOM Breakpoints
- ▶ Global Listeners
- ▼ Event Listener Breakpoints
 - ▶ Animation
 - ▶ Canvas
 - ▶ Clipboard
 - ▶ Control
 - ▶ DOM Mutation

Line 15, Column 5 Coverage: n/a

Console What's New

```
> xhr.response
{"page":1,"per_page":6,"total":12,"total_pages":2,"data":[{"id":1,"email":"george.bluth@reqres.in","first_name":"George","last_name":"Bluth","avata...
```

JavaScript: AJAX

- You can set a breakpoint on any XHR or fetch

The screenshot displays a web browser's developer console with a JavaScript file named 'filename.js' open. The code defines three functions: 'addTwoNumbers', 'subTwoNumbers', and 'ajax'. The 'ajax' function creates an XMLHttpRequest object and sends a GET request to 'https://reqres.in/api/users'. A breakpoint is set on the 'xhr.send()' line (line 11). The right-hand sidebar shows the 'Paused on XHR or fetch' status and a list of breakpoint categories, with 'Any XHR or fetch' checked.

```
1 function addTwoNumbers(var1, var2){
2   return var3 = var1+var2;
3 }
4 function subTwoNumbers(var1, var2){
5   return var3 = var1-var2;
6 }
7 function ajax(){
8   var xhr = new XMLHttpRequest(); xhr = XMLHttpRequest {onreadystatechange
9   xhr.open('GET', 'https://reqres.in/api/users', true);
10
11  xhr.send();
12
13  xhr.onload = function() {
14    let responseObj = xhr.response;
15    alert(responseObj);
16  };
17 }
18
```

Paused on XHR or fetch
https://reqres.in/api/users

- Watch
- Call Stack
- Scope
- Breakpoints
- XHR/fetch Breakpoints +
 - Any XHR or fetch
 - DOM Breakpoints
 - Global Listeners
 - Event Listener Breakpoints

JavaScript: AJAX

- You can set a breakpoint on an Event Listener - XHR

The screenshot shows a code editor with a JavaScript file named 'filename.js'. The code defines two functions: 'addTwoNumbers' and 'subTwoNumbers'. The 'ajax' function uses XMLHttpRequest to fetch data from 'https://reqres.in/api/users'. A breakpoint is set on line 14, where the response is assigned to 'responseObj' and an alert is shown. The 'Event Listener Breakpoints' panel on the right is expanded to show 'XHR' with checked options for 'readystatechange', 'load', 'loadstart', and 'loadend'.

```
1 function addTwoNumbers(var1, var2){
2   return var3 = var1+var2;
3 }
4 function subTwoNumbers(var1, var2){
5   return var3 = var1-var2;
6 }
7 function ajax(){
8   var xhr = new XMLHttpRequest();
9   xhr.open('GET', 'https://reqres.in/api/users', true);
10
11  xhr.send();
12
13  xhr.onload = function() {
14    let responseObj = xhr.response;
15    alert(responseObj);
16  };
17 }
18
```

Event Listener Breakpoints

- Animation
- Canvas
- Clipboard
- Control
- DOM Mutation
- Device
- Drag / drop
- Geolocation
- Keyboard
- Load
- Media
- Mouse
- Notification
- Parse
- Picture-in-Picture
- Pointer
- Script
- Timer
- Touch
- WebAudio
- Window
- Worker
- XHR
 - readystatechange
 - load
 - loadstart
 - loadend

Line 14, Column 23 Coverage: n/a

JavaScript: Log and line of code Breakpoint

- debugger allows you set in the code where the breakpoint will be hit
- console.log() method allows you write to the console area as the code is executing

```
function addTwoNumbers(var1, var2){
  var3 = var1+var2;
  debugger;
  return var3;
}
function subTwoNumbers(var1, var2){
  var3 = var1-var2;
  console.log("var3:" + var3);
  return var3;
}
```

JavaScript: Events

- Events
 - Actions that are preformed by a user that can be detected by JavaScript
 - Every element in the DOM has certain events linked to them
 - Examples
 - Mouse click
 - Page or image loading
 - Mouse over
 - Selecting an input field
 - Submitting form
 - Keystroke

JavaScript: Events

- Events Contd.
 - onLoad and unload
 - Triggered when the user enters or leaves a page
 - Can be used for browser detection and cookies
 - onFocus, onBlur, and onChange
 - Used to validate form fields
 - onSubmit
 - Used to validate all form fields before it is submitted
 - onMouseOver and onMouseOut
 - Used for animations and effects

Internettechniken

JavaScript Teil 2 --- AJAX

Prof. Dr. Jürgen Heym

Hochschule Hof

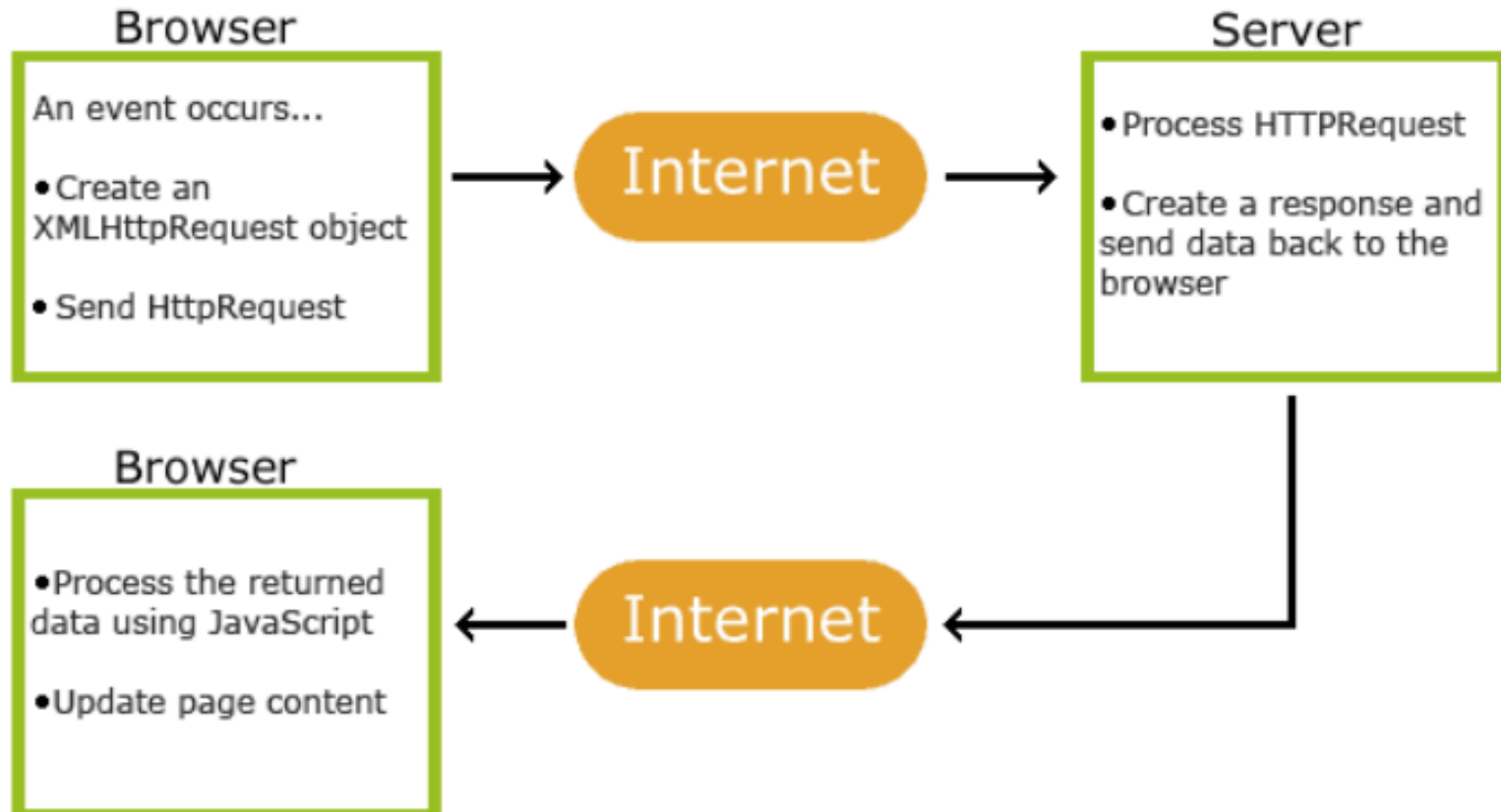
JavaScript

Einleitung

- AJAX == Asynchronous JavaScript and XML
- AJAX unterstützt den Datenaustausch mit einem Server und die Teile einer Webseite auszutauschen, ohne die gesamte Seite neu zu laden.
- AJAX ist keine neue Programmiersprache, sondern ein anderer Weg bekannt Standards einzusetzen.
 - XMLHttpRequest Objekt (asynchroner Datenaustausch)
 - JavaScript/DOM
 - CSS
 - XML
- AJAX–Anwendungen sind (meist) unabhängig von Browser und Plattform.

JavaScript

Datenflussdiagramm



- Beispielanwendung
 - Das Dokument enthält eine DIV-Sektion, die dynamisch mit Inhalten vom Server gefüllt werden soll, sobald der Knopf aktiviert wird.

```
<html>
```

```
<body>
```

```
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
```

```
<button type="button" onclick="loadXMLDoc()">Change Content</button>
```

```
</body>
```

```
</html>
```


JavaScript

Beispiel

- Beispielanwendung --- 2. Schritt
 - Fügen Sie einen JavaScript-Rumpf hinzu.

```
...  
<head>  
<script type="text/javascript">  
    function loadXMLDoc()  
    {  
        //.... AJAX script goes here ...  
    }  
</script>  
</head>  
...
```

JavaScript

Beispiel

- Beispielanwendung --- 3. Schritt
 - Wir erzeugen ein XMLHttpRequest-Objekt für den Datenaustausch mit dem Server.

...

```
var xmlhttp;  
if (window.XMLHttpRequest)  
{ // code for IE7+, Firefox, Chrome, Opera, Safari  
  xmlhttp=new XMLHttpRequest();  
}  
else  
{ // code for IE6, IE5  
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
}
```

...

JavaScript

Beispiel

- Beispielanwendung --- 4. Schritt
 - Als nächstes nutzen wir die Methoden `open` und `send` des `XMLHttpRequest`-Objekts, um eine Datei nachzuladen.

...

```
xmlhttp.open("GET", "ajax_info.txt", true);
```

```
xmlhttp.send();
```

...

Method	Description
<code>open(method,url,async)</code>	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. <i>method</i> : the type of request: GET or POST <i>url</i> : the location of the file on the server <i>async</i> : true (asynchronous) or false (synchronous)
<code>send(string)</code>	Sends the request off to the server. <i>string</i> : Only used for POST requests

JavaScript

GET-Request

- Beispiele für GET-Requests

...

```
xmlhttp.open("GET", "demo_get.asp", true);  
xmlhttp.send();
```

...

- Könnte eine Seite aus dem Cache liefern.
Deswegen fügt meine eine zufällige ID der URL hinzu:

...

```
xmlhttp.open("GET", "demo_get.asp?t=" + Math.random(), true);  
xmlhttp.send();
```

...

JavaScript

POST-Request

- POST-Requests sind robuster und sicherer.
Wir können damit ein Formular „imitieren“.
Der Header muss in diesem Fall gesetzt werden:

...

```
xmlhttp.open("POST", "ajax_test.asp", true);  
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xmlhttp.send("fname=Henry&lname=Ford");
```

...

Method	Description
<code>setRequestHeader(<i>header</i>, <i>value</i>)</code>	Adds HTTP headers to the request. <i>header</i> : specifies the header name <i>value</i> : specifies the header value

JavaScript

Asynchron --- True oder False?

- Damit wir echtes AJAX nutzen, ist der Parameter „asynchron“ auf den Wert „true“ zu setzen.
- Man muss in diesem Fall eine Funktion definieren, die ausgeführt wird, sobald die Antwort fertig ist.

...

```
xmlhttp.onreadystatechange=function()  
{  
  if (xmlhttp.readyState==4 && xmlhttp.status==200)  
  {  
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;  
  }  
}  
xmlhttp.open("GET","ajax_info.txt",true);  
xmlhttp.send();
```

...

JavaScript

Gesamtes Beispiel ajax-1.html + ajax_info.txt

```
<html>
<head>
<script type="text/javascript">
function loadXMLDoc()
{
var xmlhttp;
if (window.XMLHttpRequest)
  { // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  { // code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
}
</script>
</head>
<body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>

</body>
</html>
```

JavaScript

HttpResponse

- Server-Antwort als String oder XML
 - Um die Antwort vom Server zu verarbeiten nutzen wir die Eigenschaften `responseText` oder `responseXML` des XHR-Objekts:

Property	Description
<code>responseText</code>	get the response data as a string
<code>responseXML</code>	get the response data as XML data

- Beispiel für `responseText`

```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```


JavaScript

responseXML

- Server-Antwort als String oder XML
 - Beispiel für responseXML

```
xmlDoc=xmlhttp.responseXML;  
  
txt="";  
  
x=xmlDoc.getElementsByTagName("ARTIST");  
  
for (i=0;i<x.length;i++){  
    txt=txt + x[i].childNodes[0].nodeValue + "<br />";  
}  
  
document.getElementById("myDiv").innerHTML=txt;
```

- Sobald der Request zum Server geschickt wurde, möchte man Aktionen ausführen, die von der Antwort abhängig sind.
- Das Ereignis “onreadystatechange” wird immer, wenn der Zustand “readyState” sich ändert, ausgelöst.
- Die Eigenschaft “readyState” enthält den Status des XMLHttpRequest.
- Die drei wichtigsten Eigenschaften des XMLHttpRequest-Objekts sind:

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

- Wir spezifizieren die Funktion, die ausgeführt werden soll, wenn die Antwort des Servers vorliegt:

```
xmlhttp.onreadystatechange=function(  
{  
  if (xmlhttp.readyState==4 && xmlhttp.status==200)  
  {  
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;  
  }  
}
```

- Sobald man mehrere AJAX-Funktionen nutzen möchte sollte man Callback-Funktionen einsetzen:

```
...  
<script type="text/javascript">  
var xmlhttp;  
function loadXMLDoc(url,cfunc)  
{  
if (window.XMLHttpRequest)  
    {  
        // code for IE7+, Firefox, Chrome, Opera, Safari  
        xmlhttp=new XMLHttpRequest();  
    }  
else  
    {  
        // code for IE6, IE5  
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
    }  
xmlhttp.onreadystatechange=cfunc;  
xmlhttp.open("GET",url,true);  
xmlhttp.send();  
}  
...
```

- In myFunction wird die soeben definierte Funktion loadXMLDoc genutzt:

...

```
function myFunction1 ()
{
loadXMLDoc ("ajax_info.txt", function ()
    {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
        document.getElementById ("myDiv").innerHTML=xmlhttp.responseText;
        }
    });
}
</script>
```

...

JavaScript

Callback-Funktion

- ... und hier wird myFunction eingesetzt:

...

```
<body>
```

```
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
```

```
<button type="button" onclick="myFunction1 () ">Change Content</button>
```

```
</body>
```

```
</html>
```

Internettechniken

JavaScript Teil 3 --- PopUp-Fenster

Prof. Dr. Jürgen Heym

Hochschule Hof

JavaScript

PopUp-Fenster mit dem Window-Objekt

- Das JavaScript-Objekt „Window“ repräsentiert ein offenes Fenster in einem Browser (Root-Window, PopUp-Windows, Frames, etc.).
- Das Window-Objekt ist nicht standardisiert, aber alle Browser unterstützen es.
- Das Windows-Objekt hat Eigenschaften (properties) und Methoden (methods).

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
defaultStatus	Sets or returns the default text in the statusbar of a window
document	Returns the Document object for the window (See Document object)
frames	Returns an array of all the frames (including iframes) in the current window
history	Returns the History object for the window (See History object)
innerHeight	Sets or returns the the inner height of a window's content area
innerWidth	Sets or returns the the inner width of a window's content area
length	Returns the number of frames (including iframes) in a window
location	Returns the Location object for the window (See Location object)
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window (See Navigator object)
opener	Returns a reference to the window that created the window
outerHeight	Sets or returns the outer height of a window, including toolbars/scrollbars
outerWidth	Sets or returns the outer width of a window, including toolbars/scrollbars
pageXOffset	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
pageYOffset	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window
parent	Returns the parent window of the current window
screen	Returns the Screen object for the window (See Screen object)
screenLeft	Returns the x coordinate of the window relative to the screen
screenTop	Returns the y coordinate of the window relative to the screen
screenX	Returns the x coordinate of the window relative to the screen
screenY	Returns the y coordinate of the window relative to the screen
self	Returns the current window
status	Sets the text in the statusbar of a window
top	Returns the topmost browser window

JavaScript

Methoden des Window-Objekts

Method	Description
<u>alert()</u>	Displays an alert box with a message and an OK button
<u>blur()</u>	Removes focus from the current window
<u>clearInterval()</u>	Clears a timer set with setInterval()
<u>clearTimeout()</u>	Clears a timer set with setTimeout()
<u>close()</u>	Closes the current window
<u>confirm()</u>	Displays a dialog box with a message and an OK and a Cancel button
<u>createPopup()</u>	Creates a pop-up window
<u>focus()</u>	Sets focus to the current window
<u>moveBy()</u>	Moves a window relative to its current position
<u>moveTo()</u>	Moves a window to the specified position
<u>open()</u>	Opens a new browser window
<u>print()</u>	Prints the content of the current window
<u>prompt()</u>	Displays a dialog box that prompts the visitor for input
<u>resizeBy()</u>	Resizes the window by the specified pixels
<u>resizeTo()</u>	Resizes the window to the specified width and height
<u>scroll()</u>	
<u>scrollBy()</u>	Scrolls the content by the specified number of pixels
<u>scrollTo()</u>	Scrolls the content to the specified coordinates
<u>setInterval()</u>	Calls a function or evaluates an expression at specified intervals (in milliseconds)
<u>setTimeout()</u>	Calls a function or evaluates an expression after a specified number of milliseconds

JavaScript

Die Methode `windows.open()`

- Syntax

```
window.open(URL, name, specs, replace)
```

Parameter	Description
URL	Optional. Specifies the URL of the page to open. If no URL is specified, a new window with <code>about:blank</code> is opened
name	Optional. Specifies the target attribute or the name of the window. The following values are supported: <ul style="list-style-type: none">• <code>_blank</code> - URL is loaded into a new window. This is default• <code>_parent</code> - URL is loaded into the parent frame• <code>_self</code> - URL replaces the current page• <code>_top</code> - URL replaces any framesets that may be loaded• <code>name</code> - The name of the window
specs	Optional. A comma-separated list of items. The following values are supported:
replace	Optional. Specifies whether the URL creates a new entry or replaces the current entry in the history list. The following values are supported: <ul style="list-style-type: none">• <code>true</code> - URL replaces the current document in the history list• <code>false</code> - URL creates a new entry in the history list

JavaScript

Die Methode `windows.open()`

- Option specs
 - Komma-separierte Liste folgender Optionen

<code>channelmode=yes no 1 0</code>	Whether or not to display the window in theater mode. Default is no. IE only
<code>directories=yes no 1 0</code>	Whether or not to add directory buttons. Default is yes. IE only
<code>fullscreen=yes no 1 0</code>	Whether or not to display the browser in full-screen mode. Default is no. A window in full-screen mode must also be in theater mode. IE only
<code>height=pixels</code>	The height of the window. Min. value is 100
<code>left=pixels</code>	The left position of the window
<code>location=yes no 1 0</code>	Whether or not to display the address field. Default is yes
<code>menubar=yes no 1 0</code>	Whether or not to display the menu bar. Default is yes
<code>resizable=yes no 1 0</code>	Whether or not the window is resizable. Default is yes
<code>scrollbars=yes no 1 0</code>	Whether or not to display scroll bars. Default is yes
<code>status=yes no 1 0</code>	Whether or not to add a status bar. Default is yes
<code>titlebar=yes no 1 0</code>	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box. Default is yes
<code>toolbar=yes no 1 0</code>	Whether or not to display the browser toolbar. Default is yes
<code>top=pixels</code>	The top position of the window. IE only
<code>width=pixels</code>	The width of the window. Min. value is 100

JavaScript

Die Methode windows.open()

- Beispiel

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
function FensterOeffnen (Adresse) {

    MeinFenster = window.open(Adresse, "Zweitfenster",
        "width=300,height=400,left=100,top=200"); MeinFenster.focus();

}
</script>
</head>
<body>
<p><a href="datei.htm" onclick="FensterOeffnen(this.href); return
    false">Link mit Fenster</a>
</p>
</body>
</html>
```

Internettechniken

JavaScript / AJAX / PHP

Prof. Dr. Jürgen Heym

Hochschule Hof

JavaScript / AJAX / PHP

Übersicht

- Aufgabenstellung

Programmieren Sie eine Beispielanwendung, die ein HTML-Eingabefeld für Postleitzahlen (PLZ) und eines für den Ortsnamen realisiert, das folgende Eigenschaften aufweist:

1. Sobald ein weiteres Zeichen in das Eingabefeld PLZ des Formulars eingegeben wird, werden passende Vorschläge für vorhandene Ortsnamen mit dieser PLZ unterbreitet. Maximal jedoch 7 Vorschläge.
2. Jeder Vorschlag soll als Link ausgeführt werden, der bei Betätigung die PLZ und den Ortsnamen in das Formular überträgt. Die Hinweise werden anschließend gelöscht.

Lösung --- HTML-Formular

```
<html>
<head>
<script src="1-plz.js" type="text/javascript"></script>
</head>
<body>

<h3>Geben Sie bitte ihre Postleitzahl ein:</h3>

<form action="">
PLZ: <input type="text" id="PLZ" onkeyup="showCity(this.value)" /><br />
Ort: <input type="text" id="ORT" />
</form>

</body>
</html>
```


JavaScript / AJAX / PHP

1-plz.js

Lösung --- AJAX / JavaScript (Variante 1 mit statischer PHP-Abfrage)

Dateinamen: 1-plz.html, 1-plz.js und 1-sucheORT.php

```
function showCity(str) {
    var xmlhttp;

    if (str.length==0) {
        document.forms[0].ORT.value="Ihre Stadt ...";
        return;
    }

    xmlhttp=new XMLHttpRequest();

    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
            document.forms[0].ORT.value = xmlhttp.responseText;
        }
    }

    xmlhttp.open("GET","1-sucheORT.php?q="+str,true);
    xmlhttp.send();
}
```

JavaScript / AJAX / PHP

1-sucheORT.php

Lösung --- AJAX / JavaScript (Variante 1 mit statischer PHP-Abfrage)

Dateinamen: 1-plz.html, 1-plz.js und 1-sucheORT.php

```
<?php  
  
echo "1-sucheORT.php ...";  
  
?>
```

JavaScript / AJAX / PHP

2-plz.html

Lösung --- AJAX / JavaScript (Variante 2 mit statischer PHP-Abfrage)

Dateinamen: 2-plz.html, 2-plz.js und 2-sucheORT.php

```
<html>
<head>
<script src= "2-plz.js" type="text/javascript"></script>
</head>
<body>

<h3>Geben Sie bitte ihre Postleitzahl ein:</h3>

<form action="">
<table>
<tr><td>PLZ</td><td><input type="text" name="PLZ" onkeyup="showCity(this.value)"
    /></td></tr>
<tr><td>Ort</td><td><input type="text" name="ORT" /></td></tr>
</table>
</form>
<p>Vorschläge: <span id="txtVorschlag"></span></p>

</body>
</html>
```

JavaScript / AJAX / PHP

2-plz.js

Lösung --- AJAX / JavaScript (Variante 1 mit statischer PHP-Abfrage)

Dateinamen: 2-plz.html, 2-plz.js und 2-sucheORT.php

```
function showCity(str) {
    var xmlhttp;

    if (str.length==0) {
        document.forms[0].ORT.value="Ihre Stadt ...";
        return;
    } else {
        document.forms[0].ORT.value="";
    }

    xmlhttp=new XMLHttpRequest();

    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {

            document.getElementById("txtVorschlag").innerHTML=xmlhttp.responseText;
        }
    }

    xmlhttp.open("GET", "2-sucheORT.php?plz="+str, true);
    xmlhttp.send();
}
```

JavaScript / AJAX / PHP

2-plz.html

Lösung --- AJAX / JavaScript (Variante 1 mit statischer PHP-Abfrage)

Dateinamen: 2-plz.html, 2-plz.js und 2-sucheORT.php

```
<?php

$plz = $_GET["plz"];

$out = "<p>";
$out .= "<ul>";
$out .= "<li>";
$out .= "<a href=' ' onclick='alert(\"Hallo Welt\")';>PLZ = $plz ...</a>";
$out .= "</li>";
$out .= "</ul>";
$out .= "</p>";

echo $out;

?>
```