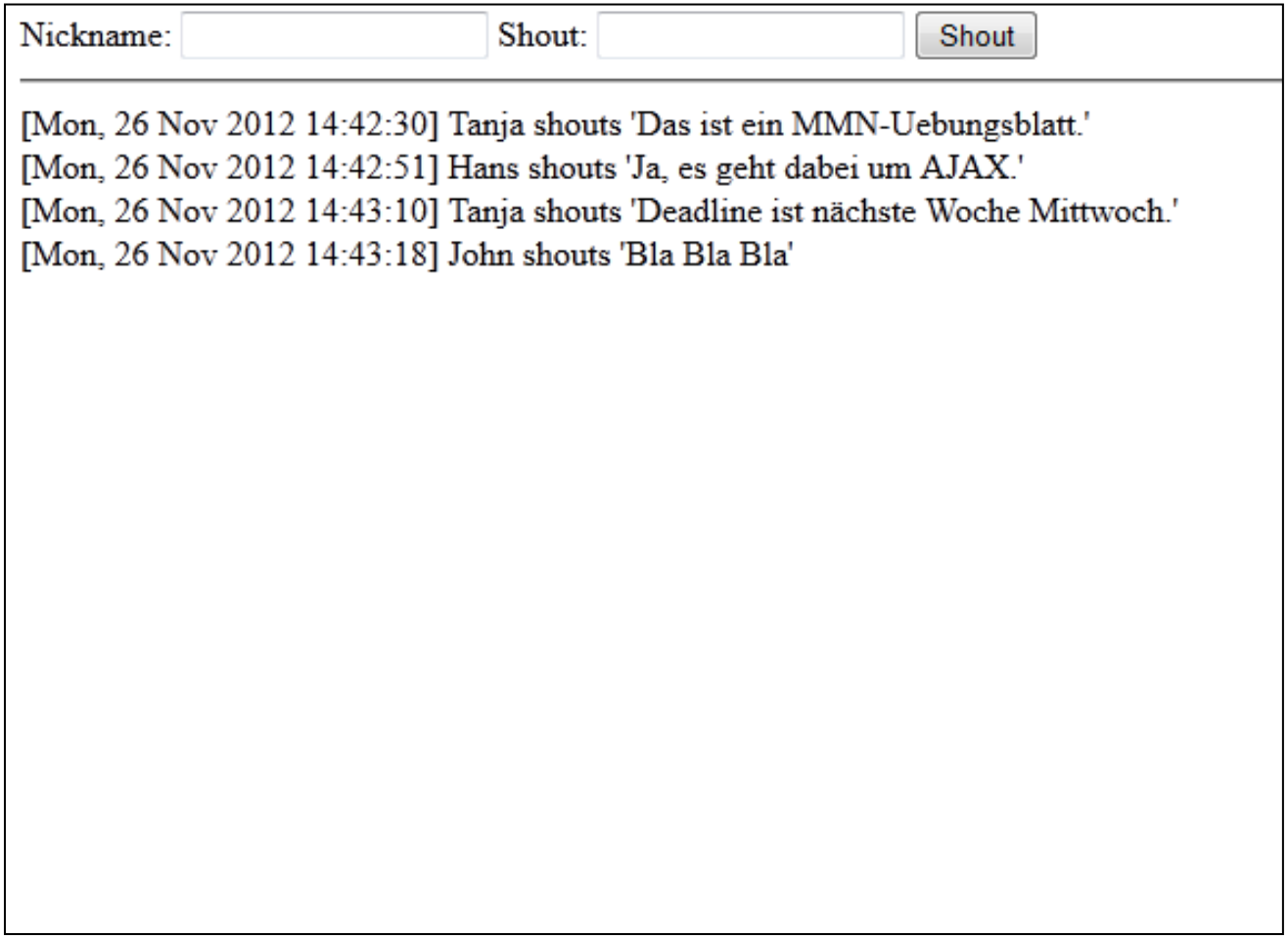
Multimedia im Netz

Wintersemester 2012/13

Übung 06

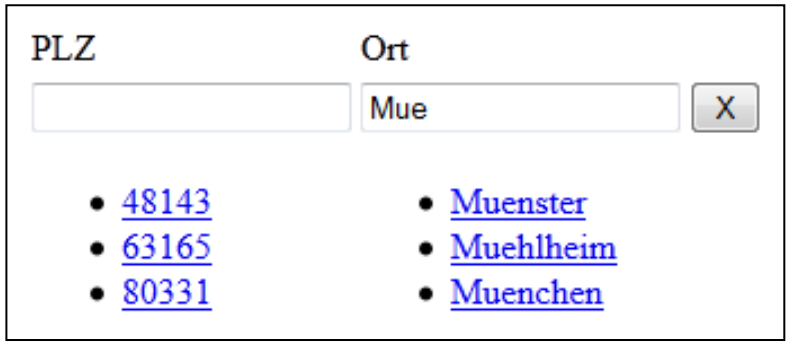
Lösung zu Übungsblatt 04

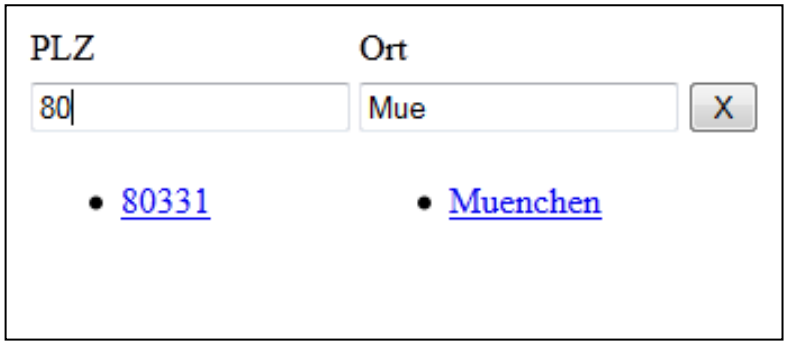
Shoutbox mit AJAX + PHP



Übungsblatt 05

- **Thema: JQuery**
- Abgabe: 12.12.2012; 11:00 Uhr





jQuery

jQuery

- http://jquery.com
- JavaScript-Bibliothek, aktuell Version 1.8.2
- Features:
  - Einfacher DOM-Zugriff
  - Event-Handling
  - Animationen
  - Einfache AJAX-Requests

- „jQuery is designed to change the way that you write JavaScript“
- Achtung: jQuery **IST** JavaScript!

Einbinden von jQuery

- Einbinden einer einzigen JavaScript-Datei
- 2 Möglichkeiten:
  - Herunterladen und Verweis auf Datei (offline):

```
<script type="text/javascript"
  src="jquery-1.8.2.min.js">
</script>
```

  - Hotlink zur Datei (empfohlen):

```
<script type="text/javascript"
  src="http://code.jquery.com/jquery-1.8.2.min.js">
</script>
```

Grundsätzliches

- Globale jQuery-Funktion: **$()**
- Parameter: beliebige CSS-Selektoren
- Rückgabe: jQuery-Objekt, das Methoden bereitstellt

- Beispiele:
  - `$("#id")`
  - `$("div.klasse")`
  - `$("input[type='text']")`

```
<div id="id">selektiert</div>

<div class="klasse">selektiert</div>
<div>nicht selektiert</div>
<div class="klasse klasse2">selektiert</div>

<input type="text" value="selektiert">
<input type="button" value="nicht selektiert">
```

- Außerdem: **$** stellt weitere Funktionen bereit, z.B.
  `$.inArray(value,array)`

DOM-Manipulation (1)

```
<div id="myDiv">Container</div>
```

- Text-Inhalt eines Elements auslesen:

```
var text = $("#myDiv").text();
```

- HTML-Inhalt eines Elements auslesen:

```
var html = $("#myDiv").html();
```

- HTML-Inhalt eines Elements setzen:

```
$("#myDiv").html("<b>Neuer Inhalt</b>");
```

- **Beachte:** `.html()` als Getter- und Setter-Methode

DOM-Manipulation (2)

```
<div id="myDiv">Container</div>
```

- HTML-Elemente erzeugen und in DOM einfügen:

```
$("#myDiv").after("<span>Inhalt</span>");
```

- Attribute auslesen/ändern:

```
$("#myDiv").addClass("container");
$("#myDiv").attr("id", "neueID");
```

- Weitere Beispiele:
  http://api.jquery.com/category/Manipulation/

Chaining

- Verkettung von Methodenaufrufen auf jQuery-Objekten
- Code effizienter und besser lesbar

- Beispiel:
  - Ohne Chaining:

```
$("#myDiv").removeClass("off");
$("#myDiv").addClass("on");
```

  - Mit Chaining:

```
$("#myDiv").removeClass("off").addClass("on");
```

DOM-Traversing

- Durchlaufen der DOM-Struktur mithilfe von Selektoren
- Selektieren/Filtern von Elementen

- Beispiele:
  - `$("#myDiv").next("div")`
    Findet das erste `div`-Element nach dem Element mit der ID `myDiv`
  - `$("ul").find("li.item")`
    Findet in allen `ul`-Elementen alle `li`-Elemente mit der Klasse `item`
  - Weitere Beispiele: http://api.jquery.com/category/Traversing/

Callback-Funktionen

- JavaScript: Funktionales Programmieren möglich
  - Funktionen als Parameter
  - Anonyme Funktionen
  - Funktionsdefinitionen „on the fly“

- Wichtiges Konzept bei jQuery: Callback-Funktionen
- Anonyme Funktion wird übergeben und von jQuery aufgerufen
- Beispiel:

```
$("li").each(function(){
    alert($(this).text());
});
```

$(document).ready()

```
$(document).ready(function(){
      //JavaScript-Code
});
```

- `$(document).ready()` garantiert, dass DOM-Dokument vollständig geladen ist und darauf zugegriffen werden kann
- Erst dann wird Callback-Funktion ausgeführt

Event-Handling (1)

- Event-Handler: Erkennen Ereignisse (z.B. Klick) und führen daraufhin eine festgelegte Aktion (= Callback-Funktion) aus

- Klick-Handler mit jQuery:

```
$("#myID").click(function(e){…});
```

- **Achtung:** Event-Handler funktionieren nur für Elemente, die bereits im DOM existieren, wenn der Handler gesetzt wird!

- Lösung (Handler auch für später hinzugefügte Elemente):

```
$(document).on("click", "#myID", function(e){…});
```

Event-Handling (2)

- Handler für viele Events möglich:
  click, change, focus, submit, keypress, …
  http://api.jquery.com/category/Events/

- Standard-Ereignisse verhindern mit `preventDefault()`:
  Beispiel:

```
$("a").click(function(event){
    event.preventDefault();
    alert("Link: " + $(this).attr("href"));
});
```

AJAX mit jQuery (1)

- jQuery bietet Funktionen für AJAX-Requests:
  - `$.ajax()`
  - `$.get(), $.post(), …`
- Vorteile: Einfachheit, Lesbarkeit, Browserkompatibilität
- Beispiel:

```
$.ajax({
    url: "...",    //URL für Request, z.B. PHP-Script
    success: function(data) { }   //Aufruf bei Erfolg
});
```

- **Beachte:** Erst in der Callback-Funktion (`success`) können empfangene Daten genutzt werden!

AJAX mit jQuery (2)

- Beispiele:
  - ```
    $.ajax({
        url: "script.php",
        type: "POST",
        data: { text: "MMN" },
        dataType: "html",
        success: function(d){
            $("#result").html(d);
        }
    });
    ```
  - Kurzform:
    ```
    $.post("script.php", { text: "MMN" }, function(d){
        $("#result").html(d);
    });
    ```
    ```

JSON

- **J**ava**S**cript **O**bject **N**otation
- Dient zur Übertragung strukturierter Daten zwischen Server und Client; Ersatz für XML
- JSON-Dokument = gültiges JavaScript

- JSON mit PHP erzeugen:

```
echo json_encode(array("key" => "value"));
//Ausgabe: {"key":"value"}
```

- JSON-Daten per AJAX-Request laden:

```
$.getJSON("script.php", function(jsonData){
    alert(jsonData.key);       //value
});
```