jQuery labs

Lab 1 - Selectors

In this lab you'll experiment with the different selectors offered by jQuery.

Step 1 - Select by element type
Use the element type selector to select all title (<h2>) elements to change the font color to gray instead of black.

Step 2 - Select by class
Use the class selector to select all "entry" elements and give them a border.

Step 3 - Select by id
Select the footer using the ID selector to give the footer a grayish background color.

Step 4 - Nested selection
Select all links in the menu (not the one in the footer) using a nested selector to style the links in the menu. A better way would be to do this using a css style, but it's just for exercise.

Step 5 - More advanced nested selection
In this step you will use a combination of selectors to style a table. First, create a selector to give each odd row in the table a different color from even rows. Second, create a selector to style the header of the table (just another <tr> in this example). Also create a selector to set the width of each first column to 200 pixels. Again, this could be done again using CSS (which would be a cleaner approach), but it's a good practice.

Lab 2 - Todo application

In this lab you'll build a simple Todo application using pure jQuery.

Step 1 - Creating a form
Create a form with a text field, a text area and a button. This form will be used to add new todo items.

Step 2 - Create a click handler
Add a click handler to the add button that adds a new item (using the data from the text input fields) to a local list of items. Because the list is not saved anywhere all data will be gone after a browser refresh, ignore this for this lab. Make sure the item is added to the list using a console.log in the event handler. Make sure the form doesn't get submitted (forcing a new browser request)!

Step 3 - Rendering items
The todo items should be displayed as a list. Create an empty as a placeholder for the items. Add code to the button's event handler to create a new html element using jQuery and add the element to the list of items.

Step 4 - Removing items
Add a “delete” button behind every item. Add an event handler for this button and a method that removes an element from the list. Use the jQuery data() attribute to identify the specific row to delete.

Step 5 - Display item details
Make each item clickable. When a user clicks an item the details of that item should be rendered in a html popup using a new, absolute positioned div. Add a close button the popup div to close it.

Lab 3 - Ajax

In this lab you will build a Twitter application. Twitter offers a nice RESTful JSON API which is a perfect example to try jQueries AJAX functionality.

Step 1 - Building the page
Start by creating the page that will display the tweets. Create an empty <ul> as a placeholder for the list of tweets, a text input field and a button.

Step 2 - Loading tweets
Add a click event listener to the button and use the $.ajax method to requests the “timeline” for the specified user. The URL for the timeline API is as follows:
http://api.twitter.com/1/statuses/user_timeline.json?screen_name=*screenname*

You can find more details about the API in the Twitter documentation here:
http://dev.twitter.com/doc/get/statuses/user_timeline

Note that you’re required to use jsonp instead of normal json as the data type to work around the problem of the same origin security rule. To be able to do this you’ll need to use the plain $.ajax method instead of the easier to use $.getJSON method. Use the console.log method to see if the result is a list of tweets.

Step 3 - Rendering tweets
Implement the success handler for the ajax call to render the list of tweets. Use the $.each method to iterate over the tweets and create a new <li> item to add to the <ul> placeholder for each tweet.

Step 4 - Cleanup the list
Display the timeline for one user, and than request the timeline for another user. You’ll notice that the second timeline is just appended to the list of the first user. Fix this by clearing the list of tweets before rendering a new timeline.

Step 5 - CSS
The current list doesn’t look very impressive. Try to improve this by adding some CSS to the page. Use the CSS3 box-shadow and -webkit-border-radius properties to get a modern look & feel.

Lab 4 - Creating a plugin

In this lab you'll create a jQuery plugin that enables table pagination.

Step 1 - Inspect the page

There is a predefined page for this lab that contains a large table of books. You're going to create a jQuery plugin that adds paging to this table.

Step 2 - Create the plugin

Add a new JavaScript file to the project and include this file on the page. Write a (empty) pagination() function and test if this code is called correctly if the method is called on a wrapped element set.

Step 3 - Implement pagination

Add code to hide all rows that are not part of the 'current' page. Make sure each page shows only 10 rows and add a next and previous button to jump to different pages. The buttons should be created from the plugin code, not directly on the page containing the table.

Step 4 - Protect from page overflow

The user is now able to show a page that doesn't contain any items by clicking the next button too many times, or by going to a negative page number. Protect the user from this by adding some code to dynamically disable the next/prev buttons.

Step 5 - Plugin configuration

Add a parameter object as an argument to the pagination method and add arguments for the number of items per page, and the css classes and texts for the buttons.

Labs jQuery UI

Lab 1 - Drag & Drop

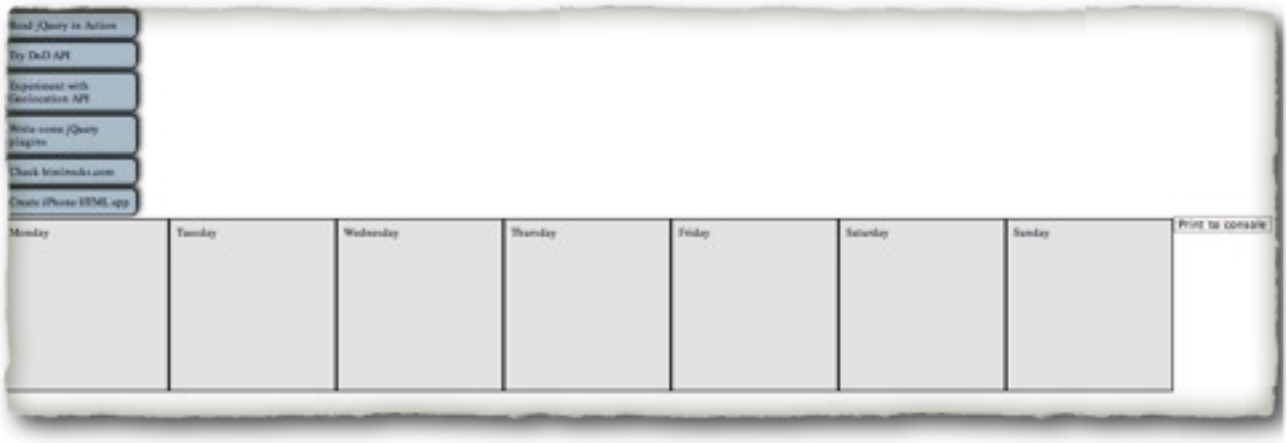
In this lab you will create a simple agenda application with drag & drop functionality using jQuery UI.

Step 1 - Investigate the page

Take a look at the page provided for the lab. It contains a bunch of divs that represent todo items and some divs that represent days in the agenda. There is no markup so everything is just shown as text elements.

Step 2 - Add CSS

First you're going to need some styling to get a page that looks like an agenda. Make sure your page looks similar to the following example.



Step 3 - Make todo items draggable

Select all todo items and call the draggable() method on them. The items should now be draggable, but they can be dropped anywhere on the page. Add two properties to the draggable(): "revert" and "scope". The scope must be set on the droppable too later to make sure they match. Only items with a matching scope will be droppable on a drop target. Test the application again and make sure items can still be dragged, but not dropped anywhere.

Step 4 - Handle drop events

Add code to handle drop events. The idea is to keep a global variable that contains the list of days with the list of todos for each day. This can be achieved by using a multi-dimensional array. A multi-dimensional array can be created using the following syntax in JavaScrpt:

```
var days = Array(7)
days[0] = new Array();
days[1] = new Array();
//...
```

Arrays can dynamically grow in JavaScript by using the push() method.

Step 5 - Check the schedule
To validate if the schedule is saved correctly you can add a button that prints the whole schedule to the console when the user clicks it.

Labs HTML 5

Lab 1 - Video

In this lab you'll create a video player with bookmarking functionality.

Step 1 - Display video

Add a video element to the page with a poster frame and custom buttons (so don't use the default controls). Make sure you can play, pause and restart a video.

Step 2 - Add bookmarking functionality

Add a "bookmark" button. When a user clicks the button an image should be rendered of the current frame. This image should be displayed below the video. When a user clicks a bookmark, the video should jump to the time saved by the bookmark. To get/set the current time in a video you use the currentTime property on the video element.