jQuery

This exercise aims at familiarizing students with the jQuery library. During this laboratory, students will obtain the following skills:

- using various selectors in order to find the desired elements in the document,
- separating the content of a document from its looks and behaviour,
- dynamically modifying the appearance of the website,
- advanced event handling,
- asynchronous loding and modification of website components,
- adding animations to websites,
- advanced DOM manipulations,
- using JSON objects,
- using promises.

To complete this exercise, you will need a text editor and a web browser.

Introduction

1. Create a *Selectors.html* file and fill it with the following code.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>jQuery</title>
  <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
  <script type="text/javascript">
    $(document).ready(function () {
      alert('Did you know that you can copy this text using Ctrl + C shortcut? ');
    });
  </script>
</head>
<body>
</body>
</html>
```

Let's look and the script in the head of this document. The $ sign is a function, which accepts an object or a selector and returns a jQuery object. This is necessary in order to use the functions from the library. You could also use the following notations. They are all equivalent.

```
$().ready(function () {...});
$(function () {...});
jQuery(document).ready(function () {...});
jQuery().ready(function () {...});
jQuery(function () {...});
```

The aim of the script from the example is to show a message as soon as the whole document tree is loaded (ready event). The same can be achieved by placing the script at the end of the document or using the onload event on the window object. However, there is a significant difference between these two approaches. The ready event doesn't wait until all objects are loaded (e.g., images), while the onload event is called only after the whole content is loaded. Therefore, use `$(document).ready` when you need the document tree and `window.onload` when you need to read for example the height of a loaded image.

Selectors

2. The previously introduced $ function accepts object or selector as a parameter. jQuery selectors are based on those you already know from CSS. Add the following code to the body of the document.

```
<p id="a">a</p><p>b</p>
<p>c</p><p>d</p>
<div>e</div><div class="c">f</div>
<table>
    <tr><td>First</td></tr>
    <tr><td>Second</td></tr>
    <tr><td>Third</td></tr>
    <tr><td>Fourth</td></tr>
    <tr><td>Fifth</td></tr>
    <tr><td>Sixth</td></tr>
</table>
<a href="wakalaka.html">Wakalaka</a>
<a href="asdf.pdf">PDF</a>
<a href="asdf.html">Asdf</a>
```

Change the script to the following code.

```
$(function () {
    $('#a').css('color', 'red').append(' appended text');
});
```

Make sure you remember the meaning of this selector. For all selected elements, we are setting a red color and appending some additional text. This example illustrates an important feature of jQuery, i.e., most methods return the object, on which they were called. This allows us to use convenient, chained calls.

3. Add the following code to the script (think about where exactly should you place it in the script).

```
$('div').css('color', 'green');
$('.c').css('font-size', 'x-large');
$('tr + tr').css('color', 'aqua');
```

Try to interpret what each line does.

4. Let us now look into group selectors.

```
$('table tr:first').css('color', 'yellow');
$('tr:last').css('color', 'yellow');
$('p:even').css('color', 'orange');
```

jQuery adds `:first`, `:last`, `:even`, and `:odd` pseudoclasses, which allow us to filter the desired elements from a given element collection. Make sure you understand the selectors used in this example.

5. Another group of selectors checks attribute values.

```
$('[href]').css('border', '1px solid black').css('padding', '3px');
$('a[href*=kalaka]').css('border', '1px solid pink');
$('a[href$=\'.pdf\']').css('border', '1px solid red');
```

Make sure you understand the selectors used in this example as well. jQuery offers several additional pseudoclasses dedicated for input elements, e.g., `:enabled`, `:disabled`, `:selected`, `:checked`.

Style

6. In the previous steps, we illustrated how selectors work by applying some CSS style to the selected elements. To achieve this goal, we used the css function, which comes in two versions: set and get. When you use the function passing two parameters, the function sets a given feature to a desired value. When you use the function passing only one parameter, the function reads the value of a given feature. Add the following line of code and observe this behavior.

```
alert('The color of an element with id="a" is ' + $('#a').css('color'));
```

7. Styling your website in a dynamic manner can be impressive, however, you should also pay attention that it is carried out in an easy to maintain manner. Just like when we were talking about separating the content of a website from its appearance, we want to maintain the same separation between appearance and behavior. That is why, it is not recommended to use the css function, but to assign elements with classes defined in CSS stylesheets. Methods `addClass()`, `removeClass()` and `hasClass()`, allow us to do just that. Create a new file entitled *Style.html* and fill it with the code from the first exercise. Add the following rule to the CSS.

```
.asdf{background: Pink;}
```

Add the following code to the body of your document.

```
<table>
    <tr><td>First</td></tr>
    <tr><td>Second</td></tr>
    <tr><td>Third</td></tr>
    <tr><td>Fourth</td></tr>
    <tr><td>Fifth</td></tr>
    <tr><td>Sixth/td></tr>
</table>
<div><button>Text A</button><button>Text B</button><span id="classic">Adding/removing
class</span></div>
<div><button>Text C</button><span id="toggle">Toggling class</span></div>
```

Add the following code to the scripts in your document.

```
$(function () {
    $('button:contains(\'A\')').click(function () {
        $('#classic').addClass('asdf');
    });
    $('button:contains(\'B\')').click(function () {
        $('#classic').removeClass('asdf');
    });
    $('button:contains(\'C\')').click(function () {
        $('#toggle').toggleClass('asdf');
    });
});
```

Observe the result and make sure you understand how it works.

8. Let us now add alternate coloring to table rows. Add the following line to your scripts.

```
$('tr:even').addClass('asdf');
```

9. Try to write the following selector on your own.

   - Select all items of a numbered list with class a, which do not contain any link inside any element with class b.

Events

10. Create a new file entitled *Events.html*, fill it with the code from the first exercise and insert the following table into the body of the document.

```
<table>
    <tr><td>One</td></tr>
    <tr><td>Two</td></tr>
    <tr><td>Three</td></tr>
    <tr><td>...</td></tr>
</table>
```

    Now write a script which will allow you to edit the contents of any row when you click on it, according to the following guidelines.

    - In the document ready function, call the following code: `$('td').one('click', edit);`. It assigns the `edit` function to the `click` event on every `td` element for one-time execution. We want it to be a one-time execution only because we will change the behavior of this element after that, as described below.
    - Write a new function called `edit`.
        - In the `td` element that is being clicked, using the `html` function, create an input element with id `"edit"` and value equal to the text of the clicked `td`. This will add an editable field in place of the content of a given table cell.
        - Set focus on the created input element (hint: use the jQuery selector and select the added element by id).
        - Now let's make sure the table goes back to normal when we leave the editable field (when it loses focus). Set the `blur` event function (opposite to `focus`) of the created input element to: a) set the text of the table cell to be the value of the input and b) reassign the `edit` function to the `click` event for one-time execution.

    In order to access the object that the event is being called on, you can either rely on the parameters of the event functions (`e.target`) or simply use the `this` reference (which in JavaScript represents the object on which the function is being currently called). Just make sure in either case to wrap it in the jQuery function. Also, in case of relying on the `this` reference always watch out what context you're in, because `this` refers to different objects in different functions, except when you're using arrow functions, which do not define a separate context!

    Make sure you understand the difference between `text`, `html` and `val` functions.

11. jQuery allows us do define our own events. This can be achieved using the `bind()` function which accepts two parameters: name of the event and a function called when this event is triggered. An event can be triggered using the trigger function, which accepts the name of the event as a parameter. Add the following line in the appropriate place in your script.

```
$('button').bind('asdf', function () { alert('The cake is a lie!'); });
```

    Trigger the event inside the `edit` function.

```
$('button').trigger('asdf');
```

    Add a `<button>` to the website and test your solution.

    Added events can be easily removed using the `unbind()` function. When called with a parameter it disables a given event and when called without parameters it disables all events

connected with a given element. Test this function by adding the following code at the end of your `$()` function.

```
$('*').unbind();
```

AJAX

12. Let us now create several subsites which we will load into a main website dynamically. First, create three html files entitled exactly as the three previous files but add AJAX at the beginning, e.g., for *Events.html* file create *AJAXEvents.html* file. Copy the contents of body elements into the respectively named new files. Create a new file called *AJAX.html* and place the following code inside. Test our website in Firefox browser.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
    <style>
        a { margin: 10px; }
        #content { border: 1px solid black; padding: 20px; width: 50%; }
    </style>
    <script type="text/javascript">
        $(function () {
            $('a').click(function () {
                $('#content').load('AJAX' + $(this).attr('href'));
                return false;
            });
        });
    </script>
</head>
<body>
    <header><nav>
        <a href="Selectors.html">Page 1</a>
        <a href="Styles.html">Page 2</a>
        <a href="Events.html">Page 3</a>
    </nav></header>
    <div id="content"></div>
</body>
</html>
```

Make sure you understand the script.

This solution has a drawback. Although the ability to navigate through the website without reloading your page gives it a “desktop feeling”. However, see what happens when you try to use the browser back button. Test how the website will behave with JavaScript turned off.

Animations

13. Let us add some animations to our website. First, we will use predefined fadeIn/fadeOut methods. Surround the load function call from previous example with fadeout/fadeIn function calls, as shown below.

```
$('#content').fadeOut();
$('#content').load('AJAX' + $(this).attr('href'));
$('#content').fadeIn();
```

Can you explain this weird behavior?

Change the click event assignment to the following code.

```
$('a').click(function () {
    let thisA = $(this);
    $('#content').fadeOut('medium', function () {
        $('#content').load('AJAX' + thisA.attr('href'), function () {
            $('#content').fadeIn();
        });
    });
    return false;
});
```

What has changed? Make sure you understand how this script works. Why couldn't we use the `this` reference anymore?

14. Change the call to predefined fadeIn/Out functions to custom animation function animate. The function allows us to specify CSS properties which we would like to change in a dynamic manner.

```
$('a').click(function () {
    $('#content').animate({height: '0px', width: '0px'}, () => {
        $('#content').load('AJAX' + $(this).attr('href'), () => {
            $('#content').animate({ height: '500px', width: '500px'});
        });
    });
    return false;
});
```

How come we're able to use the `this` reference again?

What's the `return false` for? Try to remove it from the script and see what happens.

JSON

15. In this example, you will learn how to manipulate JSON objects. Create a new file entitled *guitars.json* and paste the following content inside. Now write a website with a single button which will load the data from this file and display it in a form of a list. You will find that the browser doesn't allow you to download a local file unless it is hosted on the server, so in order to walk around this problem you may need to place the files on a local server (e.g., https://www.apachefriends.org/pl/index.html) for this exercise to work.
[HINT] You can use `getJSON` function from jQuery to load the data and the `append` function to append one element at a time to the website.

```
{
    "Guitars": {
        "Guitar": [
            { "Brand": "Gibson", "Model": "Les Paul" },
            { "Brand": "Fender", "Model": "Stratocaster" },
            { "Brand": "Charvel", "Model": "Guthrie Govan" },
            { "Brand": "Ibanez", "Model": "AZ224F" },
            { "Brand": "Tyler", "Model": "Burning Water" }
        ]
    }
}
```

Promises

16. The final assignment is not related to jQuery, but to a commonly used mechanism in modern JavaScript called promises. The goal is to write your own mechanism for asynchronously downloading data, based on the `XMLHttpRequest` object and promises. First, you may want to visit one (or both) of the links below to familiarize yourself with the concept of promises.
    - https://developers.google.com/web/fundamentals/primers/promises
    - https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261

    After that, write a function which for a given `url` address parameter will return a promise which makes an HTTP call to the given address. Parts of the `XMLHttpRequest` object which may be useful to you are: `open` and `send` functions and the `onload` and `onerror` events. In the function which will be called on the `onload` event: 1) resolve the promise passing the contents of the response if the status is `200`, 2) reject the promise passing the status and the status text description. In case of the `onerror` event, reject the promise with an appropriate error message. After writing the function, write a simple website which will use it to dynamically load some content and display it.