MakeBelieveJs

MakeBelieveJS

In this assignment your job is to create your own “jQuery” library. If you are not familiar with jQuery (http://www.jquery.com), then in general it is a library which encapsulates a lot of functionality within JavaScript and makes it easier to read. The jQuery library is extremely big and therefore you don’t need to write every functionality it provides only a handful of useful ones.

Web service

In order to test your implementation of **ajax** you need a web service. This web service is supplied and can be accessed via this location: https://serene-island-81305.herokuapp.com and its documentation can be viewed here: https://serene-island-81305.herokuapp.com/docs/. This is a simple web service which can be used like this: https://serene-island-81305.herokuapp.com/api/200 where 200 can be replaced with any valid status code in order to get a HTTP response with the associated status code. See documentation for more information.

General

The __ (*double underscore*) can return multiple elements within its query, e.g. you are searching for a HTML class which many elements are associated with. Therefore methods which are applied need to be applied to all those elements, e.g. __(‘#my-form input’).onInput(*someFunc*) should register oninput handlers for all input elements within the form with the id my-form. This is also the case when you are using methods such as parent(), grandParent(), ancestor(), etc…

Functionality

Below is the functionality which should be provided enumerated. Every functionality must be implemented with only JS. Everything should be implemented in one file called makeBelieve.js. You guessed it! Your library is called **MakeBelieveJS**.

1. **(5%)** jQuery uses **$** as its keyword to access its functionality. In this assignment you should use __ (*double underscore*) as a keyword for your functionality, so if you have a function called *hide* then it should be called __(“someQuery”).hide();. Define the __ keyword.

2. **(5%)** Implement the query selector, with the query selector I can get every element within the DOM with a valid CSS selector.

   Example:

```

// Retrieves a list of all inputs within a form with the id #my-form
var inputs = __('#my-form input');

```

3. **(5%)** All methods should be chainable, so you can call multiple MakeBelieve functions in a single chain

   Example:

```
// Chained method
__('input').parent('form').onInput(function(event) {
    alert('Something happened!');
});
```

4. **(5%)** Implement the method __(‘someSelector’).**parent**(‘someOtherSelector’ : optional) which takes in either one parameter which is a valid CSS selector or no parameter. The node list associated with the MakeBelieveElement should contain a list of all parents of the element/s based on the CSS selector. If there are no parents associated with the element, the node list associated with the MakeBelieveElement should be empty.

   Example:

```
<form>
    <input type="text" id="username" name="username" />
    <input type="password" id="password" name="password" />
</form>


var parent = __('#password').parent();
var formParent = __('#password').parent('form');
```

5. **(5%)** Implement the method __(‘someSelector’).**grandParent**(‘someOtherSelector’ : optional). This gets the first element which matched the query above this element which is considered an grandparent. A grandparent is considered someone who is not your parent but a parent of your parent.

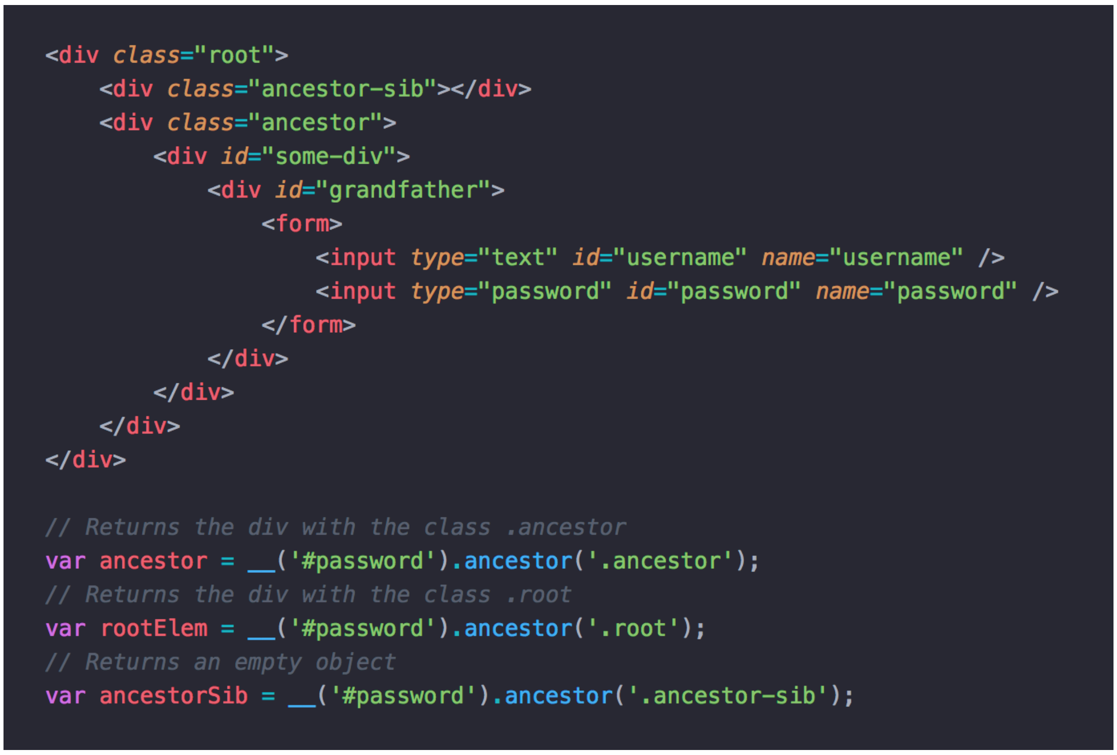
   Example:

```
<div id="grandfather">
    <form>
        <input type="text" id="username" name="username" />
        <input type="password" id="password" name="password" />
    </form>
</div>

// Returns the div with the id #grandfather
var grandParent = __('#password').grandParent();
// Returns the same div
var idGrandParent = __('#password').grandParent('#grandfather');
// Returns an empty object
var emptyGrandParent = __('#password').grandParent('#unknownId');
```

6. **(10%)** Implement the method __(‘someSelector’).**ancestor**(‘someOtherSelector’). This get the first element which matches the query above this element which is considered an ancestor. An ancestor is a person, typically one more remote than a grandparent, from whom one is descended.

   Example:

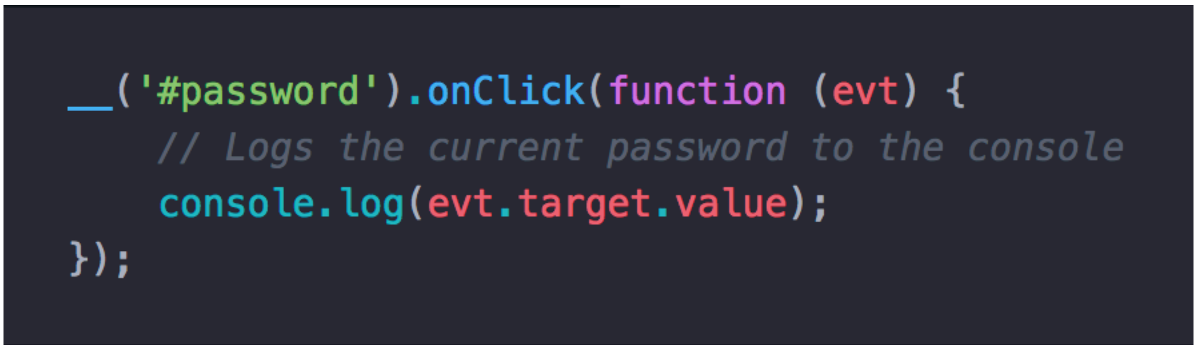


```
<div class="root">
    <div class="ancestor-sib"></div>
    <div class="ancestor">
        <div id="some-div">
            <div id="grandfather">
                <form>
                    <input type="text" id="username" name="username" />
                    <input type="password" id="password" name="password" />
                </form>
            </div>
        </div>
    </div>
</div>

// Returns the div with the class .ancestor
var ancestor = __('#password').ancestor('.ancestor');
// Returns the div with the class .root
var rootElem = __('#password').ancestor('.root');
// Returns an empty object
var ancestorSib = __('#password').ancestor('.ancestor-sib');
```

7. **(5%)** Implement a click handler which handles if an element is being clicked. The callback function provided when a click handler is evoked should have access to the event and *this* should point to the element being clicked.

   Example:



```
__('#password').onClick(function (evt) {
    // Logs the current password to the console
    console.log(evt.target.value);
});
```

8. **(5%)** Implement a method which allows you to insert a text to element. If there is text previously within the element it is overwritten.

   Example:

```
<p id="shakespeare-novel"></p>


__('#shakespeare-novel').insertText('To be, or not to be: this is the question.');
```

9. **(7.5%)** Implement a method which allows you to append new HTML to an element. It takes as parameter either a string which represents a valid HTML or an actual DOM element

   Example:

```
// Before
<div class="the-appender">
    <h2>I am the appender!</h2>
</div>

__('.the-appender').append('<p>I am an appended paragraph!</p>');
__('.the-appender').append(
    document.createElement('p')
        .appendChild(
            document.createTextNode('I am an appended paragraph!')
        )
    );

// After
<div class="the-appender">
    <h2>I am the appender!</h2>
    <p>I am an appended paragraph!</p>
    <p>I am an appended paragraph!</p>
</div>
```

10. **(7.5%)** Implement a method which allows you to prepend new HTML to an element. This is just like the function *append()* above, only now it inserts before every element inside instead of at the back. It should work for strings which are HTML formatted and actual DOM elements created using createElement.

    Example:

```
// Before
<div class="the-prepender">
    <h2>I am the prepender!</h2>
</div>

__('.the-prepender').prepend('<p>I am an prepended paragraph!</p>');
__('.the-prepender').prepend(
    document.createElement('p')
        .appendChild(
            document.createTextNode('I am an prepended paragraph!')
        )
    );

// After
<div class="the-prepender">
    <p>I am an prepended paragraph!</p>
    <p>I am an prepended paragraph!</p>
    <h2>I am the prepender!</h2>
</div>
```

11. **(5%)** Implement a method which deletes an element. If no element is found with the query it has no effect.

    Example:

```
// Before
<div class="some-div">
    <h2>Title</h2>
</div>

__('.some-div h2').delete();

// After
<div class="some-div"></div>
```

12. **(15%)** Implement a method which imitates the jQuery **ajax** method. It should take as parameter an object which contains the configuration for the HTTP request. Configurations such as:
    - **URL (required)**
    - **Method (***GET, POST, PUT, DELETE, etc.***)**
        - Defaults to GET
    - **Timeout (***seconds***)**
        - Defaults to no timeout
    - **Data**
        - Defaults to an empty object
    - **HTTP Headers (***A list of key/value pairs***)**
        - Defaults to an empty object
    - **Success callback**
        - Defaults to null
    - **Failed callback**
        - Defaults to null
    - **Before send callback**
        - Defaults to null

    Example:

```
__.ajax({
    url: '/some-url',
    method: 'GET',
    timeout: 10,
    data: {},
    headers: [
        { 'Authorization', 'my-secret-key' }
    ],
    success: function (resp) {
        // Triggered when there is a successful response from the server
    },
    fail: function (error) {
        // Triggered when an error occurred when connecting to the server
    },
    beforeSend: function (xhr) {
        // Triggered before the request and is passed in the XMLHttpRequest object
    }
});
```

13. **(5%)** Implement a method called **css()** which is used to change the direct css styles on the element. It takes in two strings, one for the which css element is being changed and the second for the value of that css element.

    Example:

```
__('#elemToChange').css('margin-bottom', '5px');
```

14. **(5%)** Implement a method called **toggleClass()** which toggles a css class for an element. Toggling meaning that if the class contains the class it removes it, otherwise adds it.

    Example:

```
__('#elemToChange').toggleClass('someClass');
```

15. **(5%)** Implement a submit handler for forms. This handler should be triggered whenever the targeted form is submitted.

    Example:

```
<form id="my-form">
    <input type="text" id="username" name="username" />
    <input type="password" id="password" name="password" />
</form>

__('#my-form').onSubmit(function (evt) {
    // Process the form
});
```

16. **(5%)** Implement an input handler for input tags. This handler should be triggered whenever the user modifies the content of the input.

    Example:

```
<form id="my-form">
    <input type="text" id="username" name="username" />
    <input type="password" id="password" name="password" />
</form>

__('#username').onInput(function (evt) {
    // Process the input
});
```

Notes
I will be aware that jQuery exists and therefore your code will be tested. If it is too similar to the original jQuery library students will be summoned to a meeting where they need to explain why their code is so similar. If the results from this meeting comes negative, the student will get a 0 for this assignment.

Submission
A single compressed (***.zip, *.rar**) file should be submitted to **Canvas**.

Useful tips
DOM info - https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
JavaScript: The Definitive Guide

- Chapter 15, 16, 17, 18 (**6th edition**)
- Chapter 15 (**7th edition**)