FritosJS

In this assignment we are going to recreate the almighty jQuery. jQuery was hugely popular around 10 to 15 years ago but still being used in some projects - although not as widely used as before. jQuery encapsulates a lot of functionality within JavaScript, especially related to operations on the DOM. Take a look at https://jquery.com/ to see how the original library works.

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 *fritos* function can return multiple elements within its query, e.g. you are searching for a HTML class which many elements are associated with. If you apply a method on those elements it will be applied to all elements within the result set, unless specifically stated.

```
// This will register 'input' handler on all inputs within #my-form
fritos('#my-form input').onEvent('input', function () {});

// This will retrieve the parent for all associated elements
fritos('div .item').parent();
```

Assignment description

All methods below must be implemented using JavaScript. The methods should be stored within a single file called fritos.js.

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

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

```
const inputs = fritos('#my-form input');
```

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

```
// Chained method
fritos('input').parent('form').onEvent('input', function(event) {
  alert('Something happened');
});
```

4. **(5%)** Implement the method **parent** which accepts an optional parameter called selector which is a string. The method should return a list of all parents of the elements within the result set. If the optional parameter **selector** is provided only the parents matching the passed in selector should be returned.

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

```
const parent = fritos('input[type="password"]').parent();
const unknownParent = fritos('#password').parent('div');
```

5. **(5%)** Implement the method **ancestor** which accepts an optional parameter called selector which is a string. The method should return a list of all ancestors of the elements within the result set. If the optional parameter selector is provided only the ancestors matching the passed in selector should be returned. An ancestor is a node which is further up in the tree than a parent. The root node is the top most ancestor for all elements within the DOM.

```
<body>
  <main>
    <div class="items">
      <div class="item">
        <img src="" alt="" class="item-image" />
      </div>
      <div class="item">
        <img src="" alt="" class="item-image" />
      </div>
      <div class="item">
        <img src="" alt="" class="item-image" />
      </div>
    </div>
  </main>
</body>
```

```
// Returns the first ancestor <div class="items"></div>
fritos('.item-image').ancestor();
// Returns the ancestor <main></main>
fritos('.item-image').ancestor('main');
```

6. **(10%)** Implement the method **animate** which accepts two arguments: CSS properties and animation options. The method should only return the values selected from the initial selector. The function should perform an animation on all the items within the result set. The animation should be performed as described in the CSS properties and animation options. The CSS properties argument tells us what properties should be animated and the values noted in the CSS properties are the values the animation is moving towards. The animation options argument tells us how the animation should play out.

   The CSS properties argument allows all valid CSS properties but the notation should be in camel casing instead of the default kebab case as seen in CSS.

   The animation options arguments allow the following options: duration, delay, easing, iterationCount and fillMode.

```
<div class="moveable-item">
  I am about to move!
</div>
```

```
fritos('.moveable-item').animate({
  transform: 'translateX(100px)'
}, {
  // Time in milliseconds
  duration: 1000,
  // time, 'initial', 'inherit'
  delay: '2s',
  // ease, linear, ease-in, ease-in-out, cubic-bezier(n, n, n, n)
  easing: 'linear',
  // number, 'infinite', 'initial', 'inherit'
  iterationCount: 1,
  // none, forwards, backwards, both, initial, inherit
  fillMode: 'none'
});
```

7. **(5%)** Implement the method **find** which accepts a required parameter called selector which is a string. The method should return a list of all children of the elements within the result set. An empty result set should be returned if there is no selector provided.

```
<body>
  <div class="container">
    <div class="items">
      <div class="item"></div>
      <div class="item"></div>
      <div class="item"></div>
      <div class="item"></div>
      <div class="item"></div>
    </div>
  </div>
</body>
```

```
// Returns 5 <div class="item"></div>
fritos('.container').find('.item');
```

8. **(5%)** Implement the method **onEvent** which accepts two required parameters: event type and event function. The method should return the same result set as the initial selector yielded. The method should attach an event listener with the passed in event type to all items within the result set.

```
<form action="">
  <div class="form-group">
    <input type="input" name="input-1" />
  </div>
  <div class="form-group">
    <input type="input" name="input-2" />
  </div>
  <div class="form-group">
    <input type="input" name="input-3" />
  </div>
</form>
```

```
fritos('form .form-group input[type="input"]')
.onEvent('input', function(evt) {
  // This will print out the
  console.log(evt.target.value);
});
```

9. **(10%)** Implement the method **remoteCall** which accepts two required parameters: an external URL to a remote server and http request options. The method should not return anything - as this is not a regular chained method associated with a result set. The function should perform a network request to the provided external URL with the configurations passed in as arguments. The usage of axios is not allowed.

   The external URL to a remote server is only a string, e.g. https://example.com/api

   The HTTP request options provided as the second argument contains the following properties: method, timeout, headers, body, onSuccess, onError.

```
fritos.remoteCall('https://example.com/api/client', {
  // GET, PUT, POST, PATCH, DELETE, HEAD, CONNECT, OPTIONS, TRACE,
PATH
  method: 'POST',
  // The timeout specified in seconds (defaults to 45)
  timeout: 45,
  // An object representing the headers associated with the HTTP
request
  headers: {
    'Accept-Language': 'is-IS',
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  // A request body (in this case parsed as JSON)
  body: JSON.stringify({
    id: 1,
    name: 'John Doe'
  }),
  // A success function which is called if the HTTP request was
successful
  onSuccess: function(data) {
    // TODO: Use data
  },
  // An error function which is called if the HTTP request
encountered an error
  onError: function(err) {
    // TODO: Handle error
  }
});
```

10. **(15%)** Implement the method **validation** which accepts one required argument for validation properties. The validation will only work if there are children within the result set which can be validated e.g. input, textarea and select. The method returns a validation result for the form found within the result set. This method will only work when the result set contains a single value - if there is more than one value the first value within the result set will be used. This method cannot be chained further.

```
<form action="" id="user-credentials">
  <div class="form-group">
    <input type="text" name="email-address" />
  </div>
  <div class="form-group">
    <input type="password" name="password" />
  </div>
  <div class="form-group">
    <input type="password" name="confirm-password" />
  </div>
  <div class="form-group">
    <button type="input">Submit</button>
  </div>
</form>
```

```
const result = fritos("#user-credentials").validation({
  "email-address": [
    {
      message: "The email address is required",
      valid: (value) => value.length > 0,
    },
    {
      message: "The email address must be correctly formatted",
      // Regex to check if the email address is correctly formatted
      valid: (value) =>
        /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/g.test(value),
    },
  ],
  password: [
    {
      message:
        "The password must contain at least one character, number and special
character",
      valid: (value) =>
        // Regex to check if string is more than 8 in length contains
         // at least one character, number and special character
        /^(?=.*\d)(?=.*[a-zA-Z])(?=.*[^a-zA-Z0-9]).{8,}$/g.test(value),
    },
  ],
  "confirm-password": [
    {
      message: "The confirm password must match the password",
      valid: (value, parent) => {
        const password = parent.querySelector('input[name="password"]');
        return value === password.value;
      },
    },
  ],
});

console.log(result);
/**

If the form is valid an empty object is returned: {}
If the form contains any errors, the input values within the form will be
listed:
{
  emailAddress: 'The email address is required',
  password: 'The password must contain at least one character, number and
special character',
  'confirm-password': 'The confirm password must match the password'
}

 */
```

11. **(5%)** Implement the method **hide** which accepts no parameters. The method should hide all the elements within the result set.

```
<div class="items">
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
</div>
```

```
// Hides all elements
fritos('.item').hide();
```

12. **(5%)** Implement the method **prune** which accepts no parameters. The method should remove the parents of the elements within the result set - therefore pruning the elements up the tree by taking their parents place.

```
<!-- Before calling prune() -->
<form action="">
  <div class="form-group">
    <input type="text" name="input-1" />
  </div>
  <div class="form-group">
    <input type="text" name="input-2" />
  </div>
  <div class="form-group">
    <input type="text" name="input-3" />
  </div>
  <div class="form-group">
    <input type="text" name="input-4" />
  </div>
  <div class="form-group">
    <input type="text" name="input-5" />
  </div>
</form>
```

```
fritos('input[type="text"]').prune();
```

```
<!-- After calling prune() -->
<form action="">
  <input type="text" name="input-1" />
  <input type="text" name="input-2" />
  <input type="text" name="input-3" />
  <input type="text" name="input-4" />
</form>
```

13. **(10%)** Implement the method **raise** which accepts an optional parameter called level. The method should raise all the elements found in the result set to the desired level - by default the raise level is one. By raising the element, the elements swap with their parents until the desired level is reached. The ordering as seen in the example matter.

```
<!-- Before calling raise() -->
<form action="">
  <div class="form-group">
    <input type="text" name="input-1" />
  </div>
  <div class="form-group">
    <input type="text" name="input-2" />
  </div>
  <div class="form-group">
    <input type="text" name="input-3" />
  </div>
  <div class="form-group">
    <input type="text" name="input-4" />
  </div>
  <div class="form-group">
    <input type="text" name="input-5" />
  </div>
</form>
```

```
fritos('input[type="text"]').raise();
```

```
<!-- After calling raise() -->
<form action="">
  <input type="text" name="input-1" />
  <div class="form-group"></div>
  <input type="text" name="input-2" />
  <div class="form-group"></div>
  <input type="text" name="input-3" />
  <div class="form-group"></div>
  <input type="text" name="input-4" />
  <div class="form-group"></div>
  <input type="text" name="input-5" />
  <div class="form-group"></div>
</form>
```

```
<!-- After calling raise(2) -->
<input type="text" name="input-1" />
<input type="text" name="input-2" />
<input type="text" name="input-3" />
<input type="text" name="input-4" />
<input type="text" name="input-5" />
<form action="">
  <div class="form-group"></div>
  <div class="form-group"></div>
  <div class="form-group"></div>
  <div class="form-group"></div>
  <div class="form-group"></div>
</form>
```

14. **(5%)** Implement the method **attrs** which accepts two required arguments: attribute name and attribute value. The method should return the same result set as the initial selector yielded. The attribute name provided should be added or replaced by the new value after the method has executed.

```
<form action="">
  <div class="form-group">
    <input type="text" name="1" />
  </div>
  <div class="form-group">
    <input type="text" name="2" />
  </div>
  <div class="form-group">
    <input type="text" name="3" />
  </div>
  <div class="form-group">
    <input type="text" name="4" />
  </div>
  <div class="form-group">
    <input type="text" name="5" />
  </div>
</form>
```

```
fritos('form .form-group input').attrs('name', 'input');
```

```
<form action="">
  <div class="form-group">
    <input type="text" name="input" />
  </div>
  <div class="form-group">
    <input type="text" name="input" />
  </div>
  <div class="form-group">
    <input type="text" name="input" />
  </div>
  <div class="form-group">
    <input type="text" name="input" />
  </div>
  <div class="form-group">
    <input type="text" name="input" />
  </div>
</form>
```

15. **(5%)** Implement the method **val** which accepts an optional parameter called value. The method should return the same result set as the initial selector yielded. If the value parameter is passed in, the elements within the result set should be updated with the provided value. If no value parameter is passed in, the current value of the first element in the result set should be returned.

```
// Updates all values for the matching elements
fritos('input[type="text"]').val('Default value');

// Returns the value of the first matching element.
const value = fritos('input[type="text"]').val();
```

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 come negative, the student will get a 0 for this assignment.

Submission

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

Reading material

DOM info - https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
JavaScript: The Definitive Guide, 7th edition - Chapter 15