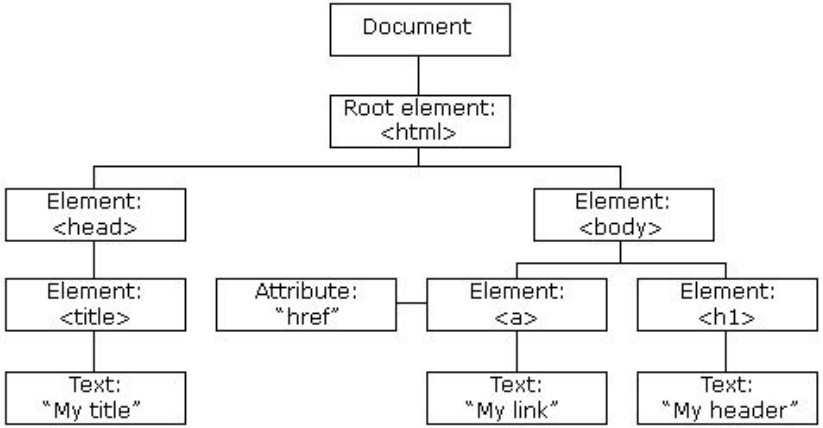
JavaScript & DOM

The DOM - Document Object Model

When a web page is loaded, the browser creates a **Document Object Model** of the page.
The **HTML DOM** model is constructed as a tree of **Objects**:

```
<html>
  <head><title>My Title</title></head>
  <body>
    <a href="gallery.html">My Link</a>
    <h1>My header</h1>
  </body>
</html>
```



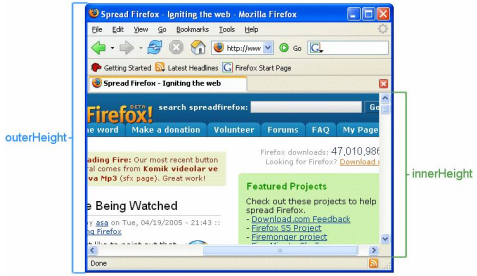
It represents the page so that programs can change the document structure, style, and content.
The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

document object

The document object is the **root** node of the HTML document. Some Useful properties/methods:

Property / Method	Description
activeElement	Returns the currently focused element in the document
addEventListener()	Attaches an event handler to the document
body	Sets or returns the document's body (the <body> element)
cookie	Returns all name/value pairs of cookies in the document
createElement()	Creates an Element node
documentElement	Returns the Document Element of the document (the <html> element)
getElementById() , getElementsByClassName() getElementsByName() , getElementsByTagName()	Return element(s) by a specific name/class/tag/id
querySelector() , querySelectorAll()	Returns the first element or elements that matches a specified CSS selector(s) in the document
title	Sets or returns the title of the document

window object



A **global** variable representing the window in which the script is running. Some Useful properties/methods:

Property / Method	Description
console	Provides methods for logging information to the browser's console
document	Returns the Document object for the window
history	Returns the History object for the window
innerHeight , innerWidth	Returns the width/height of a window's content area (viewport) including scrollbars (if rendered)
localStorage	Allows to save key/value pairs in a web browser. Stores the data with no expiration date
location	Returns the Location object for the window
navigator	Returns the Navigator object for the window (that contains browser information)
outerHeight , outerWidth	Returns the width/height of the whole browser window including sidebar

Tip: Every HTML element with id "x" or a global var x can be accessed via window.x

DOM Selecting - querySelector / querySelectorAll

```
var match = document.querySelector('div');
```

querySelector() returns the **first** DOM element that match a given CSS selector.

```
var matches = document.querySelectorAll('div.check');
```

querySelectorAll() returns a list of DOM elements that match a given CSS selector.

```
for (i=0; i<matches.length; i++)
    console.log(matches[i].innerHTML);
```

Then we can loop over these elements and do something with their properties/methods

Walking the DOM

`<html>` = **`document.documentElement`**

The topmost document node is `document.documentElement`. That's DOM node of `<html>` tag.

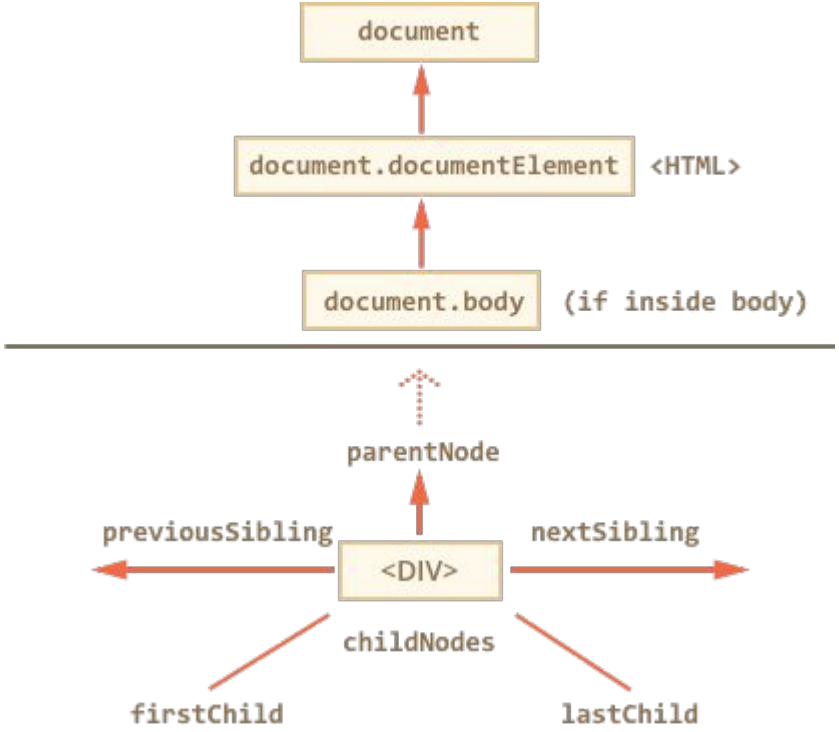
`<body>` = **`document.body`**

Another widely used DOM node is the `<body>` element – `document.body`.

`<head>` = **`document.head`**

The `<head>` tag is available as `document.head`.

From: https://javascript.info/dom-navigation



Walking the DOM - childNodes vs. children

```
<html>
<body>
  <div>Begin</div>
  <ul>
    <li>Information</li>
  </ul>
  <div>End</div>
  <script>
    for (var i = 0; i < document.body.childNodes.length; i++) {
      console.log( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT
    }
    for (var i = 0; i < document.body.children.length; i++) {
      console.log( document.body.children[i] ); // Element only - DIV, UL, ..., SCRIPT
    }
  </script>
</body>
</html>
```

DOM Manipulation - Create a DOM Element

```
var el = document.createElement('div');
```

Use the **`createElement()`** method for creating a DOM element

```
el.innerHTML = '<p>Hello World!</p>';
```

Fill the new element with any HTML content

```
document.body.appendChild(el);
```

"el" can now be inserted into the DOM tree

```
var textnode = document.createTextNode('Hello World!');
el.appendChild(textnode);
```

Alternatively, use DOM methods for creating content nodes and append them to the new element.

DOM Manipulation - Create a DOM Element with Loop

```
var ul = document.querySelector('ul');
for (var index = 0; index < 10; index++) {
    var li = document.createElement('li');
    li.className = 'item';
    // li.setAttribute('class', 'item');
    li.textContent = 'List Item '+ index;
    ul.appendChild(li);
}
```

Select ul element from the DOM

10 times loop

Create a li element and assign it to a variable **li**

Put a class attribute with the value "item"

The equivalent method is to use setAttribute

Put a content 'List Item 0' , 'List Item 1'...

Append the li element to the ul child

DOM Manipulation - Remove and empty a DOM element

```
// remove an element
var el = document.querySelector('div'); // select the first returned <div> element
el.parentNode.removeChild(el);

// empty an element
var el = document.querySelector('div');
el.innerHTML = '';
```

JSFiddle: http://jsfiddle.net/gocode/v7ckjys6/

DOM Manipulation - Styling a DOM element

```
var el = document.querySelector('div');

el.style.backgroundColor = 'green';
el.style.display = 'none';
el.style['border-radius'] = '5px';
```

JSFiddle - Style with multiple values at once: http://jsfiddle.net/gocode/vaw1phrx/

HTML Element Methods

```
document.querySelector("button").click();
```

click() simulates a mouse click on an element.

```
document.getElementById("myTextField").focus();
```

focus() sets focus on the specified element, if it can be focused.

```
document.getElementById("myTextField").blur();
```

blur() removes keyboard focus from the current element.

DOM Manipulation Cheat-Sheet

Finding HTML Elements	
document.querySelector(query)	Find element by query
document.querySelectorAll(query)	Find elements by query
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements	
element.innerHTML = *content*	Change the inner HTML of an element
element.attribute = value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = style*	Change the style of an HTML element

Adding and Deleting Elements	
document.createElement(element)	Create an HTML element
document.createTextNode(string)	Creates a new text node with the node value of string
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
node.replaceChild(newNode, oldNode)	Replaces the child node oldNode of node with newNode
document.write(text)	Write into the HTML output stream
newNode = node.cloneNode(bool)	Creates newNode as a copy (clone) of node. If bool is true, the clone includes clones of all the child nodes of the original.

Events

Things that happening in browser and running code in response.

```
var myHTML = document.querySelector('html');
myHTML.onclick = function() {
      // WRITE HERE WHAT TO DO
      // WHEN YOU CLICK ON HTML ELEMENT
};
```

Select HTML element and assign it into a variable

Assign an anonymous function into the variable

onclick event

For Example:

```
var myHTML = document.querySelector('html');
myHTML.onclick = function() {
      alert('CLICKED!');
};
```

Events - addEventListener function

```
target.addEventListener(type, listener[, options]);
```

addEventListener() sets up a function that will be called whenever the specified event is delivered to the target.

Event Type Event Handler

```
myHTML.addEventListener('click', function() {
    alert('CLICKED!');
});
```

Events - Inline events handlers

We can write events on the attribute of the desired element:

```
<button onclick="alert('alert text')">BUTTON</button>
```

Event Type Event Handler

We can also use our declared function as a event handler:

```
<button onclick="myFunc('text')">BUTTON</button>
```

Tip: it is not a good idea to mix up your HTML and your JavaScript (unless you're using a framework)

Events - Event Object

The **event object** is automatically passed to event handlers to provide extra features and information.

Including:

target

Returns the element that triggered the event

type

Returns the name of the event

clientX & clientY (MouseEvent only)

Returns the horizontal/vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered

button (MouseEvent only)

Returns which mouse button was pressed when the mouse event was triggered

event.code (on KeyboardEvent)

Get the pressed key name (usually used on a "onkeydown" event)

```
<input type="text" onkeydown="printCode(event)">
<div>THE CODE: </div><div class="code"></div>
<script>
function checkCode(event) {
  return event.code;
};

function printCode(event) {
  document.querySelector('.code').innerHTML = checkCode(event);
}
</script>
```

Note that's not supported in all browsers. For IE & Edge use the old "event.keyCode"

Input event.target.value

Get the input value (usually used on a "oninput" event with input)

```
<style> input {
      display: block;
      margin: 10px 0; }
</style>
<p>How many people would you like
            to invite to the party?</p>
<input type="number" oninput="showInputs(event.target.value)" />
<p>Party members:</p>
<div class="container"></div>
<button>Send Invitations</button>
```

```
<script>
    function showInputs(value) {
      var container = document.querySelector('.container');
      if (container) {
        container.innerHTML = '';
      }

      if (value > 0) {
        for (var index = 0; index < value; index++) {
          var input = document.createElement('input');
          input.placeholder = 'Full Name';
          container.appendChild(input);
        }
      }
    }
</script>
```

Think how to show the "send" button only if there're some members to invite

Mouse Over & Mouse Out events example

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Document</title>
  </head>
  <body>
    <div class="mouse-over-box" style="background-color:#FBB0A6;width:120px;height:20px;padding:40px;">
      Mouse Over Me!
    </div>
    <script>
      document.querySelector('.mouse-over-box').addEventListener('mouseover', function(e) {
          e.target.innerHTML = 'Ouch!!! <br/>Please go out!';
        });
      document.querySelector('.mouse-over-box').addEventListener('mouseout', function(e) {
          e.target.innerHTML = 'Thanks!!!';
        });
    </script>
  </body>
</html>
```

e.target - (event target)
A reference to the object
that dispatched the event

Special Events - onload & DOMContentLoaded

onload event will be called only after the DOM and associated resources (like images) got loaded

```
<body onload="function() { alert('BODY LOADED!'); }"> .... <body>
```

Or, alternatively:

```
window.addEventListener('load', function(event) {
  alert('Body Loaded!');
});
```

DOMContentLoaded event will be called once the DOM is loaded - for example - it won't wait for the resources like images to get loaded

```
document.addEventListener('DOMContentLoaded', function(event) {
  alert('DOMContentLoaded');
});
```

Events Types

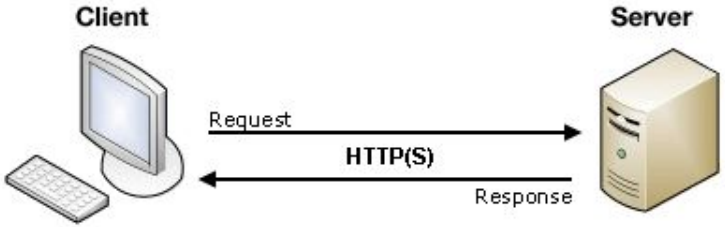
There're many events types!

keypress
keydown
keyup
focus
mousedown
mousemove
mouseup
mouseover

And many more...
https://developer.mozilla.org/en-US/docs/Web/Events

JSFiddle:
https://jsfiddle.net/gocode/wtgjd03m/

Forms - Form element



HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own.

```
<form action="/my-handling-form-page" method="post">
... Form elements ...
</form>
```

The **action** attribute defines the location (URL) where the form's collected data should be sent when it is submitted.

The **method** attribute defines which HTTP method to send the data with (it can be "get" or "post").

MDN: https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data

Forms - Form element - Action attribute

This attribute defines where the data gets sent. Its value must be a valid URL. If this attribute isn't provided, the data will be sent to the URL of the page containing the form.

In this example, the data is sent to an absolute URL – http://foo.com:

```
<form action="http://foo.com">
```

Here, we use a relative URL – the data is sent to a different URL on the server:

```
<form action="/contact">
```

When specified with no attributes, as below, the <form> data is sent to the same page that the form is present on:

```
<form>
```

Forms - Form element - Method attribute - GET

The GET method is the method used by the browser to ask the server to send back a given resource:
"Hey server, I want to get this resource."
In this case, the browser sends an **empty body**.
Because the body is empty, if a form is sent using this method the data sent to the server is **appended to the URL**.

After the URL web address has ended, we include a question mark (?) followed by the name/value pairs, each one separated by an ampersand (&). In this case we are passing two pieces of data to the server

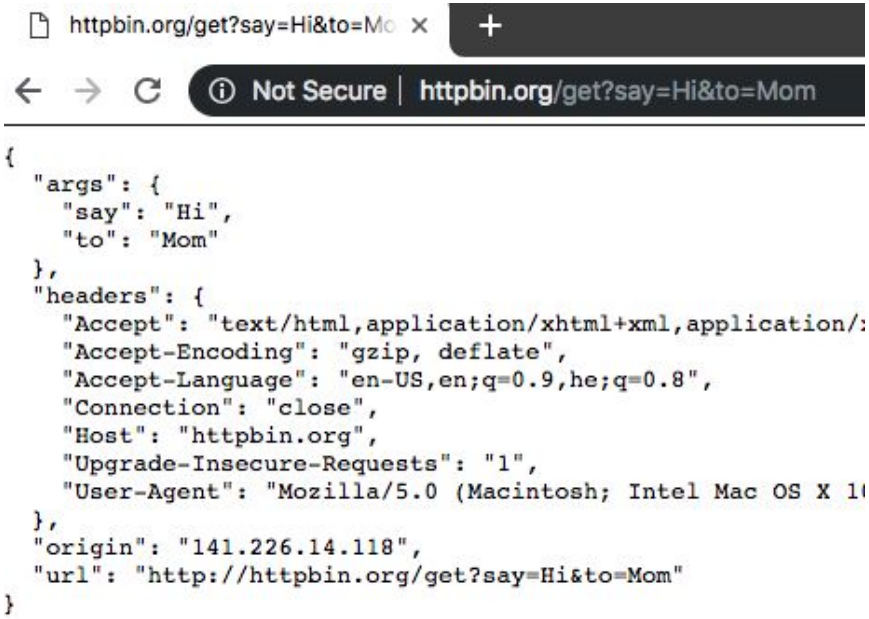
```
urlFormAction?formInputName1=val1&formInputName2=val2&formInputName3=val3 ....
```

The request structure:

```
GET /get?say=Hi&to=Mom HTTP/2.0
Host: httpbin.org
```

Forms - Form element - GET Method example

```html
<form action="http://httpbin.org/get" method="get">
  <div>
    <label for="say">What greeting do you want to say?</label>
    <input name="say" id="say" value="Hi" />
  </div>
  <div>
    <label for="to">Who do you want to say it to?</label>
    <input name="to" id="to" value="Mom" />
  </div>
  <div><button>Send my greetings</button></div>
</form>
```



Forms - Form element - Method attribute - POST

The POST method is a little different.
It's the method the browser uses to talk to the server when asking for a response that takes into account the data provided in the body of the HTTP request:
"Hey server, take a look at this data and send me back an appropriate result."

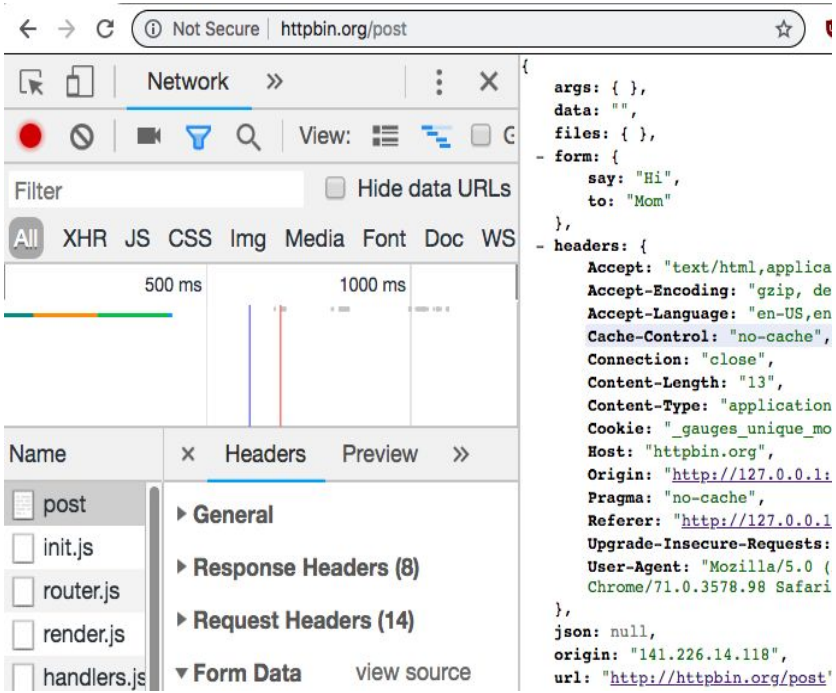
If a form is sent using this method, the data is appended to the **body** of the HTTP request.

urlFormAction (The form data isn't exist the URL)

```
POST /post HTTP/2.0
Host: httpbin.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 13
say=Hi&to=Mom
```

Forms - Form element - POST Method example

```
<form action="http://httpbin.org/post" method="post">
  <div>
    <label for="say">What greeting do you want to say?</label>
    <input name="say" id="say" value="Hi" />
  </div>
  <div>
    <label for="to">Who do you want to say it to?</label>
    <input name="to" id="to" value="Mom" />
  </div>
  <div><button>Send my greetings</button></div>
</form>
```



Forms - Full form example

```
<form action="/my-handling-form-page" method="post">
  <div>
    <label for="name">Name:</label>
    <input type="text" id="name" name="user_name">
  </div>
  <div>
    <label for="mail">E-mail:</label>
    <input type="email" id="mail" name="user_mail">
  </div>
  <div>
    <label for="msg">Message:</label>
    <textarea id="msg" name="user_message"></textarea>
  </div>
  <button>Submit</button>
</form>
```

Forms - Input element

The HTML <input> element is used to create interactive controls for web-based forms in order to accept data from the user

```
<label for="name">Name (4 to 8 characters):</label>

<input type="text" id="name" name="name" required
       minlength="4" maxlength="8" size="10">
```

Forms - Input element

autofocus	A Boolean which, if present, makes the input take focus when the form is presented
disabled	A Boolean attribute which is present if the input should be disabled
name	The input's name, to identify the input in the data submitted with the form's data
readonly	A Boolean attribute which, if true, indicates that the input cannot be edited
required	A Boolean which, if true, indicates that the input must have a value before the form can be submitted
tabindex	A numeric value providing guidance to the user agent as to the order in which controls receive focus when the user presses the Tab key
type	A string indicating which input type the <input> element represents
value	The input's current value

Forms - Input element types

button	A push button with no default behavior.
checkbox	A check box allowing single values to be selected/deselected.
number	A control for entering a number.
password	A single-line text field whose value is obscured. Use the **maxlength** and **minlength** attributes to specify the maximum length of the value that can be entered.
radio	A radio button, allowing a single value to be selected out of multiple choices.
reset	A button that resets the contents of the form to default values.
submit	A button that submits the form.
text	A single-line text field. Line-breaks are automatically removed from the input value.

Forms - Input element - checkbox & radio

Checkbox

```
<input type="checkbox" checked id="carrots" name="carrots" value="carrots">
```

Radio

```
<input type="radio" checked id="soup" name="meal">
```

Forms - Input element - Radio group

```html
<fieldset>
  <legend>What is your favorite meal?</legend>
  <ul>
    <li>
      <label for="soup">Soup</label>
      <input type="radio" checked id="soup" name="meal" value="soup">
    </li>
    <li>
      <label for="curry">Toast</label>
      <input type="radio" id="toast" name="meal" value="toast">
    </li>
    <li>
      <label for="pizza">Pizza</label>
      <input type="radio" id="pizza" name="meal" value="pizza">
    </li>
  </ul>
</fieldset>
```

Forms - Textarea

```
<label for="story">Tell us your story:</label>

<textarea id="story" name="story" rows="5" cols="33" maxlength="100">
It was a dark and stormy night...
</textarea>
```

The rows and cols attributes allow you to specify an exact size for the <textarea> to take. Setting these is a good idea for consistency, as browser defaults can differ.

maxlength specifies a maximum number of characters that the <textarea> is allowed to contain. You can also set a minimum length that is considered valid using the minlength attribute, and specify that the <textarea> will not submit (and is invalid) if it is empty, using the required attribute

Forms - Select element

```
<label for="pet-select">Choose a pet:</label>

<select id="pet-select">
    <option value="">--Please choose an option--</option>
    <option value="dog">Dog</option>
    <option value="cat">Cat</option>
    <option value="hamster">Hamster</option>
    <option value="parrot">Parrot</option>
    <option value="spider">Spider</option>
    <option value="goldfish">Goldfish</option>
</select>
```

JSFiddle for multiple select:
https://jsfiddle.net/gocode/hd4Lc5w8/

Forms - Buttons

Within HTML forms, there are three kinds of button:

Submit

> **Sends the form data to the server**. For <button> elements, omitting the type attribute (or an invalid value of type) results in a submit button.

Reset

> Resets all form widgets to their default values.

Anonymous

> Buttons that have **no automatic effect** but can be customized using JavaScript code.

Forms - Submit button

As input:

```
<input type="submit" value="This is a submit input">
```

As button:

```
<button type="submit">This is a submit button</button>
```

Button element is more powerful, because it allows to put a HTML element inside it

If the button is under a form element, it'll be a submit button by default (without using type attribute).

```
<button>This is a submit button</button>
```

When the user clicks/triggers a submit button, the form data is sent to the <form> action url by using the <form> method

```
<form action="/my-handling-form-page" method="post"> ... </form>
```

Forms Validation - Intro

Form validation helps us to ensure that users fill out forms in the correct format, making sure that submitted data will work successfully with our applications.

Validations Examples:

- "This field is required" (you can't leave this field blank)
- "Please enter your phone number in the format xxx-xxxx" (it enforces three numbers followed by a dash, followed by four numbers)
- "Please enter a valid e-mail address"
  (if your entry is not in the format of "somebody@example.com")
- "Your password needs to be between 8 and 30 characters long, and contain one uppercase letter, one symbol, and a number"

Forms Validation - Client-Side & Server-Side

- Client-side validation is validation that occurs in the browser before the data has been submitted to the server. This is more user-friendly than server-side validation as it gives an instant response. This can be further subdivided:
  - JavaScript validation is coded using JavaScript. It is completely **customizable.**
  - Built-in form validation using HTML5 form validation features. This generally **does not require JavaScript**. Built-in form validation has **better performance**, but it is **not as customizable** as JavaScript.

- Server-side validation is validation which occurs on the server after the data has been submitted. Server-side code is used to validate the data before it is saved into the database. If the data fails authentication, a response is sent back to the client to tell the user what corrections to make.

Forms Validation - "required" attribute

```html
<form>
  <label for="choose">Would you prefer a banana or cherry?</label>
  <input id="choose" name="i_like" required>
  <button>Submit</button>
</form>
```

And change the style using :invalid pseudo-class

```css
input:invalid {
  border: 2px dashed red;
}

input:valid {
  border: 2px solid black;
}
```

Tip: There're also :in-range :out-of-range pseudo classes

Forms Validation - lengths attributes

Add lengths constraints using
"**min**", "**max**" on number and "**minlength**", "**maxlength**" on text

```
<form>
  <div>
    <label for="choose">Would you prefer a banana or a cherry?</label>
    <input id="choose" name="i_like" required minlength="6" maxlength="6">
  </div>
  <div>
    <label for="number">How many would you like?</label>
    <input type="number" id="number" name="amount" value="1" min="1" max="10">
  </div>
  <div>
    <button>Submit</button>
  </div>
</form>
```

Forms Validation - JS Validation Using valid. API

Add "novalidate" to disable the native validate API

```html
<form novalidate>
  <p>
    <label for="mail">
      <span>Please enter an email address:</span>
      <input type="email" id="mail" name="mail">
      <span class="error"></span>
    </label>
  </p>
  <button>Submit</button>
</form>
```

```css
/* This is our style for
the invalid fields */
input:invalid{
  border-color: #900;
  background-color: #FDD;
}

input:focus:invalid {
  outline: none;
}
```

```css
/* This is the style of our error
messages */
.error {
  width  : 100%;
  padding: 0;

  font-size: 80%;
  color: white;
  background-color: #900;
  border-radius: 0 0 5px 5px;

  -moz-box-sizing: border-box;
  box-sizing: border-box;
}

.error.active {
  padding: 0.3em;
}
```

Forms Validation - JS Validation Using valid. API

```
var form  = document.querySelector('form');
var email = document.querySelector('#mail');
var error = document.querySelector('.error');

email.addEventListener("input", function (event) {
  // Each time the user types something, we check if the
  // email field is valid.
  if (email.validity.valid) {
    // In case there is an error message visible, if the field
    // is valid, we remove the error message.
    error.innerHTML = ""; // Reset the content of the message
    error.className = "error"; // Reset the visual state of the message
  }
});
```

```
form.addEventListener("submit", function (event) {
  // Each time the user tries to send the data, we check
  // if the email field is valid.
  if (!email.validity.valid) {

    // If the field is not valid, we display a custom
    // error message.
    error.innerHTML = "I expect an e-mail, darling!";
    error.className = "error active";
    // And we prevent the form from being sent by
canceling the event
    event.preventDefault();
  }
});
```

Forms Validation - Pure JS Validation

To check if a field is valid, like email, we use **Regular Expressions** (Regex): patterns used to match character combinations in strings.

We init regular expression this way:

```
var re1 = new RegExp("abc");
```

Or this way:

```
var re2 = /abc/;
```

And use test function on regex to check if it match the string passed as parameter:

```
console.log(re1.test("abcde"));
// → true
console.log(re1.test("abxde"));
// → false
```

Regular expressions can contain many special characters for checking any pattern... but we'll discuss it on another time...

Forms Validation - Pure JS Validation

Same code here!

```html
<form novalidate>
  <p>
    <label for="mail">
      <span>Please enter an email address:</span>
      <input type="email" id="mail" name="mail">
      <span class="error"></span>
    </label>
  </p>
  <button>Submit</button>
</form>
```

```css
/* This is our style for
the invalid fields */
input:invalid{
  border-color: #900;
  background-color: #FDD;
}

input:focus:invalid {
  outline: none;
}
```

```css
/* This is the style of our error
messages */
.error {
  width  : 100%;
  padding: 0;

  font-size: 80%;
  color: white;
  background-color: #900;
  border-radius: 0 0 5px 5px;

  -moz-box-sizing: border-box;
  box-sizing: border-box;
}

.error.active {
  padding: 0.3em;
}
```

Forms Validation - Pure JS Validation - Part 1

```
var form = document.querySelector('form');
var email = document.querySelector('#mail');
var error = document.querySelector('.error');

function isExistAndValidEmail(email) {
  // As per the HTML5 Specification
  var emailRegExp =
/^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;
  return email.length !== 0 && emailRegExp.test(email);
}
```

```
// This defines what happens when the user types in the field
email.addEventListener('input', function() {
  var isValidEmail = isExistAndValidEmail(email.value);

  if (isValidEmail) {
    email.className = 'valid';
    error.innerHTML = '';
    error.className = 'error';
  } else {
    email.className = 'invalid';
  }
});
```

Forms Validation - Pure JS Validation - Part 2

```
// This defines what happens when the user tries to submit the data
form.addEventListener('submit', function(e) {
  var isValidEmail = isExistAndValidEmail(email.value);

  if (isValidEmail) {
    email.className = 'valid';
    error.innerHTML = '';
    error.className = 'error';
  } else {
    email.className = 'invalid';
    error.innerHTML = 'I expect an e-mail, darling!';
    error.className = 'error active';

    e.preventDefault();
  }
});
```

JSON - The data format for client & server

- JSON = **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

JSON file is like a JS object. Just pay attention that all properties should be wrapped with " (Quotation mark). Example for data.json file:

```
{
  "employees": [
    { "firstName": "John", "lastName": "Doe", "age": 20, "isMarried": false },
    { "firstName": "Anna", "lastName": "Smith", "age": 34, "isMarried": true },
    { "firstName": "Peter", "lastName": "Jones", "age": 5, "isMarried": false }
  ]
}
```

JSON - JSON.parse() & JSON.stringify

```
var json = JSON.parse(str)   Convert a string into a JS object
```

```
var str = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
var obj = JSON.parse(str);
```

```
var str = JSON.stringify(json)   Convert a JS object into a string
```

```
var json = {
    "employees": [
    { "firstName": "John", "lastName": "Doe"},
    { "firstName": "Anna", "lastName": "Smith"},
    { "firstName": "Peter", "lastName": "Jones"}
  ]
}
var str = JSON.stringify(json);
```

JSON - Examples from servers with JSON response

- Currencies: http://api.openrates.io/latest
- Full cat API: https://docs.thecatapi.com/
- Random cat file: https://aws.random.cat/meow
- Random dog file: https://dog.ceo/api/breeds/image/random
- Random fox file: https://randomfox.ca/floof/
- Books details: https://openlibrary.org/api/books?bibkeys=ISBN:0385472579,LCCN:62019420&format=json&jscmd=data
- Open Trivia: https://opentdb.com/api.php?amount=10 (https://opentdb.com/api_config.php)

API: **A**pplication **P**rogramming **I**nterface - Helps the enterprises expose individual components in well-documented services that the internal developers and partners can use to rapidly iterate new features.

Public APIs Lists:
https://github.com/toddmotto/public-apis#currency-exchange
https://github.com/n0shake/Public-APIs

Install JSON Viewer Chrome extension to view JSON responses:
https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh

XMLHttpRequest() - AJAX

Make requests to the server without reloading the page!

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
It is the use of the XMLHttpRequest object to communicate with servers.

It can send and receive information in various formats, including JSON, XML, HTML, and text files.

This communication is in an "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without **having to refresh the page** along with allowing other JavaScript code to execute.

Examples:

- Get the current Dollar currency in Shekels in order to convert an item price from Dollar to Shekels.
- Check if the username exists in the database while the user write on the username field

Cover Image by Christopher Keefer

XMLHttpRequest() - Make Request

```
<button onclick="makeRequest()">Make Request</button>
<script>
  function makeRequest() {
    var httpRequest = new XMLHttpRequest();
    httpRequest.onreadystatechange = function() {
      if (httpRequest.readyState === XMLHttpRequest.DONE) {
        if (httpRequest.status === 200) {
          alert(httpRequest.responseText);
        } else {
          alert('There was a problem with the request.');
        }
      }
    };
    httpRequest.open('GET', 'test.html');
    httpRequest.send();
  }
</script>
```

An example to get test.html content into current page

Tell the XMLHttpRequest object which JavaScript function will handle the response, by setting the **onreadystatechange** property of the object and naming it after the function to call.

StatusCode: **200 OK** - The request has succeeded.

XMLHttpRequest.open(**method, url**)

XMLHttpRequest() - Make Request with JSON Response

An example to get a JSON response

```
<button onclick="makeRequest()">Make Request</button>
<script>
  function makeRequest() {
    var httpRequest = new XMLHttpRequest();
    httpRequest.onreadystatechange = function() {
      if (httpRequest.readyState === XMLHttpRequest.DONE) {
        if (httpRequest.status === 200) {
          console.log(JSON.parse(httpRequest.responseText).rates.ILS);
        } else {
          alert('There was a problem with the request.');
        }
      }
    };
    httpRequest.open('GET', 'https://api.exchangeratesapi.io/latest');
    httpRequest.send();
  }
</script>
```

Convert the response text to JS Object and take the related ILS currency rate

XMLHttpRequest() - onload event

An example to get a JSON response in a more modern way

```
<button onclick="makeRequest()">Make Request</button>
<script>
  function makeRequest() {
    var httpRequest = new XMLHttpRequest();
    httpRequest.responseType = 'json';
    httpRequest.onload = function() {
      if (httpRequest.status === 200) {
        console.log(httpRequest.response.rates.ILS);
      } else {
        alert('There was a problem with the request.');
      }
    };
    httpRequest.open('GET', 'https://api.exchangeratesapi.io/latestsa');
    httpRequest.send();
  }
</script>
```

The onload occured when the readyState is DONE

You don't need any convert here because the response type is already in a JSON format

XMLHttpRequest() - onloadstart event

```
<div class="rate-label">ILS for 1 Dollar:</div>
    <div class="rate"></div>
    <button onclick="makeRequest()">Make Request</button>
    <div class="loading"></div>
    <script>
      var loading = document.querySelector('.loading');
      var rate = document.querySelector('.rate');
      function makeRequest() {
        var httpRequest = new XMLHttpRequest();
        httpRequest.responseType = 'json';
        httpRequest.onload = function() {
          if (httpRequest.status === 200) {
            rate.innerHTML = httpRequest.response.rates.ILS;
            loading.innerHTML = 'Done!';
          } else {
            alert('There was a problem with the request.');
          }
        };
```

Adds a Loading/Done text
(Can be replaced with a nice icon)

```
        httpRequest.onloadstart = function() {
            loading.innerHTML = 'Loading...';
        };
        httpRequest.open('GET',
'https://api.exchangeratesapi.io/latest');
        httpRequest.send();
      }
    </script>
```

SetTimeout

Schedules **callback** to be called in **ms** milliseconds.

```
var timeoutID = setTimeout(callback, ms)
```

Example:

```
var timeoutID = setTimeout(function() { alert('That was really slow!'); }, 2000);
```

To cancel the timeout:

```
clearTimeout(timeoutID);
```

SetInterval

Repeatedly calls a function or executes a code snippet, with a fixed time delay between each call

```
var intervalID = setInterval(callback, ms)
```

Example:

```
var intervalID = setInterval(function() { console.log(new Date()) }, 2000);
```

To cancel the interval:

```
clearInterval(intervalID);
```

SetTimeout examples

SetTimeout callback with arguments:

```
function sayHi(phrase, who) {
  alert( phrase + ', ' + who );
}
setTimeout(sayHi, 1000, "Hello", "John"); // Hello, John
```

SetTimeout recursive calls with 1 sec between each call:

```
function sayHi(phrase, who) {
  console.log(phrase + ', ' + who);
  setTimeoutToSayHi();
}
function setTimeoutToSayHi() {
  setTimeout(sayHi, 1000, 'Hello', 'John'); // Hello, John
}
setTimeoutToSayHi();
```

SetTimeout - asynchronous function running separately

Try this code and see what happens...

```
<script>
  console.log(1);
  setTimeout(function() {
    console.log(2);
  }, 0);
  console.log(3);
</script>
```

setTimeout doesn't run immediately even with 0 milliseconds...

A detailed explanation can be found here: https://hackernoon.com/understanding-js-the-event-loop-959beae3ac40

SetTimeout - asynchronous function advantage

```
<table border=1>
    <tr><td><button id='do'>Do long calc - bad status!</button></td>
        <td><div id='status'>Not Calculating yet.</div></td>
    </tr>
    <tr><td><button id='do_ok'>Do long calc - good status!</button></td>
        <td><div id='status_ok'>Not Calculating yet.</div></td>
    </tr>
</table>
```

setTimeout doesn't block the drawing of document!

SetTimeout - asynchronous function advantage

JSFIDDLE: http://jsfiddle.net/gocode/v9erxkt0/

```
<script>
function longRunning(status_div) {

    var result = 0;
    for (var i = 0; i < 5000; i++) {
        for (var j = 0; j < 700; j++) {
            for (var k = 0; k < 300; k++) {
                result = result + i + j + k;
            }
        }
    }

document.querySelector(status_div).innerHTML='cal
clation done';
}
```

```
document.querySelector('#do').addEventListener('click', function ()
{
    document.querySelector('#status').innerHTML = 'calculating....';
    longRunning('#status');
});
document.querySelector('#do_ok').addEventListener('click', function
() {
    document.querySelector('#status_ok').innerHTML =
'calculating....';
    setTimeout(function (){ longRunning('#status_ok') }, 0);
});
</script>
```

setTimeout doesn't block the drawing of document!

SetTimeout recursive vs. SetInterval

Credits: https://javascript.info/settimeout-setinterval

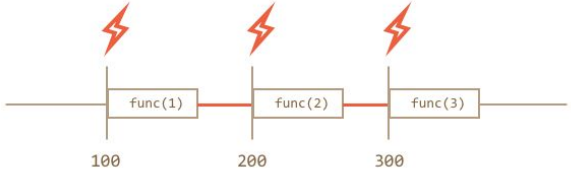
Let's compare two code fragments. The first one uses `setInterval`:

```
let i = 1;
setInterval(function() {
  func(i);
}, 100);
```

The second one uses recursive `setTimeout`:

```
let i = 1;
setTimeout(function run() {
  func(i);
  setTimeout(run, 100);
}, 100);
```

For `setInterval` the internal scheduler will run `func(i)` every 100ms:



Did you notice?

The real delay between `func` calls for `setInterval` is less than in the code!

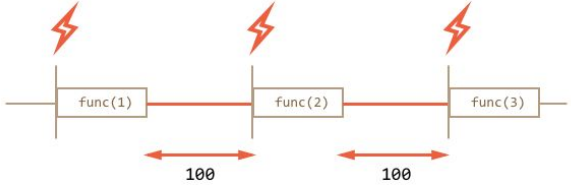
That's normal, because the time taken by `func`'s execution "consumes" a part of the interval.

It is possible that `func`'s execution turns out to be longer than we expected and takes more than 100ms.

In this case the engine waits for `func` to complete, then checks the scheduler and if the time is up, runs it again *immediately*.

In the edge case, if the function always executes longer than `delay` ms, then the calls will happen without a pause at all.

And here is the picture for the recursive `setTimeout`:



XMLHttpRequest() - Ajax with setInterval

```
<button onclick="clearRequestInterval()">Clear Interval</button><br />
<div class="counter"></div>
<div class="loading"></div>
<img class="cat-img" src="" width="100" />
<script>
  var INTERVAL_TIME_MS = 3000;
  var catImg = document.querySelector('.cat-img');
  var loading = document.querySelector('.loading');

  makeRequest();

  var requestInterval = setInterval(function() {
    makeRequest();
   }, INTERVAL_TIME_MS);

  function clearRequestInterval() {
    clearInterval(requestInterval);
   }
```

```
function makeRequest() {
    var httpRequest = new XMLHttpRequest();
    httpRequest.responseType = 'json';
    httpRequest.onloadstart = function() {
      loading.innerHTML = 'Loading...';
    };
    httpRequest.onload = function() {
      if (httpRequest.status === 200) {
        catImg.src = httpRequest.response.file;
        loading.innerHTML = 'Done!';
      } else {
        alert('There was a problem with the request.');
      }
    };

    httpRequest.open('GET', 'https://aws.random.cat/meow');
    httpRequest.send();
  }
</script>
```

Exercises - DOM Manipulation & Events

1. Create a list with 30 items, each item with a different color & background-color.
2. Create buttons to show/hide images on the page.
   What's the difference between display and visibility CSS properties? Check and see!
3. Try to write exercise 1 again but with choosing a random background-color
4. https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_dom_html1
5. https://github.com/eladcandroid/js-exercises/blob/master/week-5/A-dom-manipulation/exercise.js
6. https://github.com/eladcandroid/js-exercises/blob/master/week-5/B-callbacks/exercise.js - Only exercise 1

Exercises - DOM Manipulation & Events

---

7. Count and show the number of clicks on the page.
8. Use an event to show the x, y current coordinates of mouse movement.
9. Put some elements on a page (div, inputs, img etc..) and when the user clicks on a "garbage" icon, he will able to click on each element and remove it.
10. Create a table structure by prompting the rows & columns number (2 prompts). Use a dark border for table styling to show it.
11. Put a box on a div and 4 buttons that moves this box up, down, right and left by 10px for each click.

Exercises - DOM Manipulation & Events

Write a page that displays a balloon (using the balloon emoji, 🎈). When you press the up arrow, it should inflate (grow) 10 percent, and when you press the down arrow, it should deflate (shrink) 10 percent.
You can control the size of text (emoji are text) by setting the font-size CSS property (style.fontSize) on its parent element. Remember to include a unit in the value—for example, pixels (10px).
The key names of the arrow keys are "ArrowUp" and "ArrowDown". Make sure the keys change only the balloon, without scrolling the page.
When that works, add a feature where, if you blow up the balloon past a certain size, it explodes. In this case, exploding means that it is replaced with an 💥 emoji, and the event handler is removed (so that you can't inflate or deflate the explosion).

From: https://eloquentjavascript.net/15_event.html#i_ZPJB9UFdQA

Emojis list: https://unicode.org/emoji/charts/full-emoji-list.html

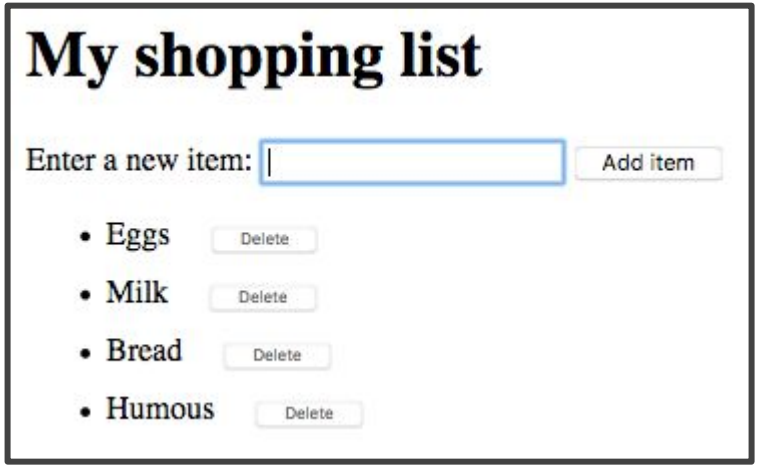
Exercise - DOM Manipulation Final - Shopping list

We want to make a simple shopping list example that allows you to dynamically add items to the list using a form input and button.

When you add an item to the input and press the button:

- The item should appear in the list.
- Each item should be given a button that can be pressed to delete that item off the list.
- The input should be emptied and focused ready for you to enter another item.

The finished demo will look something like this:



Exercise - DOM Manipulation Final - Shopping list

To complete the exercise, follow the steps below, and make sure that the list behaves as described above.

To start with, download a copy of shopping-list.html starting file and make a copy of it somewhere. You'll see that it has some minimal CSS, a list with a label, input, and button, and an empty list and <script> element. You'll be making all your additions inside the script.

- Create three variables that hold references to the list (<ul>), <input>, and <button> elements.
- Create a function that will run in response to the button being clicked.
- Inside the function body, start off by storing the current value of the input element in a variable.
- Next, empty the input element by setting its value to an empty string – ''.
- Create three new elements – a list item (<li>), , and <button>, and store them in variables.
- Append the span and the button as children of the list item.
- Set the text content of the span to the input element value you saved earlier, and the text content of the button to 'Delete'.
- Append the list item as a child of the list.
- Attach an event handler to the delete button, so that when clicked it will delete the entire list item it is inside.
- Finally, use the focus() method to focus the input element ready for entering the next shopping list item.

Forms - Exercise

Create a register form that will send a post request to http://httpbin.org/post with the following fields:

Username - input with min 3 characters and max 10 characters

Email - input with an email type

Password - input with a password type and should contain numbers and letters

Repeat Password - input with a password type

Hobbies - checkboxes with like 5 options to choose from

Gender - radio buttons with "Male/Female"

About me - textarea with maxlength of 150

Phone - 2 inputs - 1st with select with providers options (050, 052 etc..) and 2nd with the phone itself.

Validations - show them near each field and check it on input and on submit events:

All fields are required.

Passwords should match each other.

The user have to check at least one hobbie

Style this form the best you can.

Bonus: Add a text near the textarea to show how many characters the user wrote.. for example: "60/150"

Another bonus: Show a icon with "V" when the input passed and "X" when there's a validation error.

Ajax - Exercises

Exercise 1 - Use API

Write a solution to this exercise using only JavaScript (without JQuery)
https://www.rithmschool.com/courses/intermediate-javascript-part-2/ajax-exercises

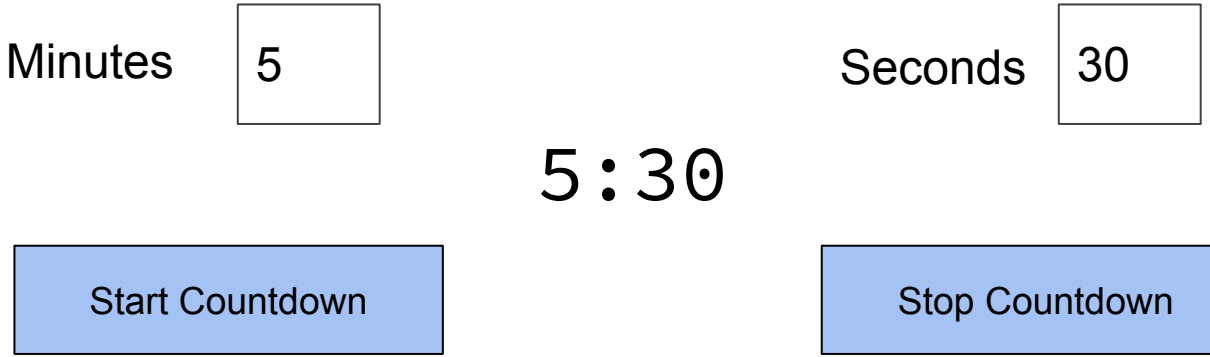
Exercise 2 - Your own data file

1. Add to your forms exercise project a file "users.json" that will simulate 5 registered users with all of their details (array of objects).
   Check in every character input if the username in the input is already exist in users.json.
   Add this check to the validation process - show an error if the user exist.
2. Create a file "users.html" that makes a request to users.json to show all the exist users in a nice HTML table.
3. Combine users.html code in index.html (form) so that the form will be on top and the list of users in the bottom of this page.

setTimeout & setInterval - Exercises

Ajax Countdown Timer

1. Build a html page with a number input that takes seconds.
2. Add a button "Start Countdown"
3. Add a button "Stop Countdown"
4. When the user clicks on the "Start Countdown" button, the countdown will start until it'll reach zero seconds.
5. When the user clicks on the "Stop Countdown" button, the countdown will stop immediately.
6. Add minutes input for the countdown for counting with minutes & seconds.
7. When the countdown ends - make an AJAX call to https://aws.random.cat/meow and show the relevant image.



More Info

1. https://youtu.be/QyUFheng6J0
2. https://plainjs.com
3. https://javascript.info
4. https://eloquentjavascript.net
5.