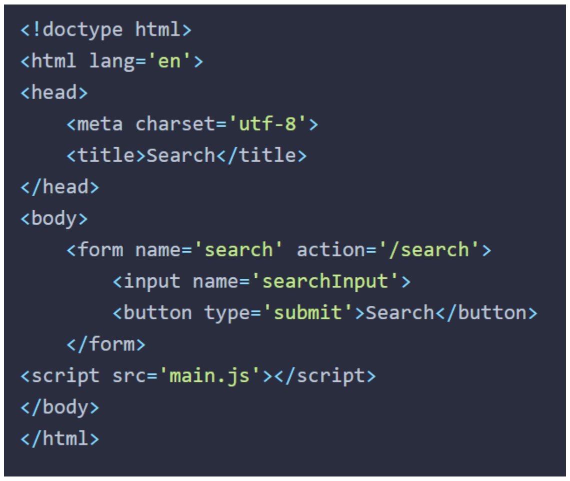
1.- CH8: FORMS

A **form** is the main component of Google's home page.

- ♥ A very common method of interacting with a web page
- ♥ Are made up of a **<form>** element
- ♥ A form contains "form controls" (input fields, select menus, buttons)
- ♥ Information is processed once the form has been submitted.

1)- A searching example

Simple example of a form that contains one input field



```
<!doctype html>
<html lang='en'>
<head>
    <meta charset='utf-8'>
    <title>Search</title>
</head>
<body>
    <form name='search' action='/search'>
        <input name='searchInput'>
        <button type='submit'>Search</button>
    </form>
<script src='main.js'></script>
</body>
</html>
```

Where:

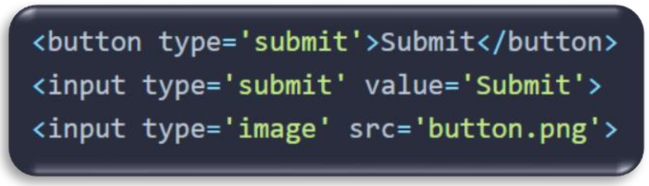
- ♥ Has a **name** attribute of **search**

- ♥ Contains two controls: an input field where a user can enter a search phrase, and a button to submit the form
  - The input field also has a name attribute of **searchInput** that is used to access the information inside it.
- ♥ The **action** attribute is the URL that the form will be submitted to (so it can be processed on the server side)

2)- FORM properties and methods

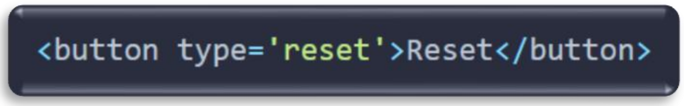
- The **form.submit()** method will submit the form automatically.

-A **form** can be submitted manually by the user employing a button or input element with a type of attribute of submit



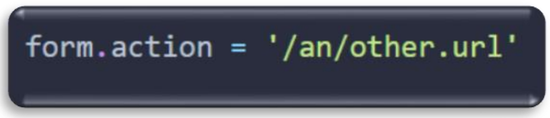
```
<button type='submit'>Submit</button>
<input type='submit' value='Submit'>
<input type='image' src='button.png'>
```

-The **form.reset()** method will reset all the form controls back to their initial values specified in the HTML.



```
<button type='reset'>Reset</button>
```

Note: A **button** with a **type** attribute of reset can also be used to do this without the need for additional scripting.



```
form.action = '/an/other.url'
```

-The **form.action** property can be used to set the action attribute of a form

3)- FORM events

Forms trigger a number of **events.**

The focus event occurs when an element is focused on.

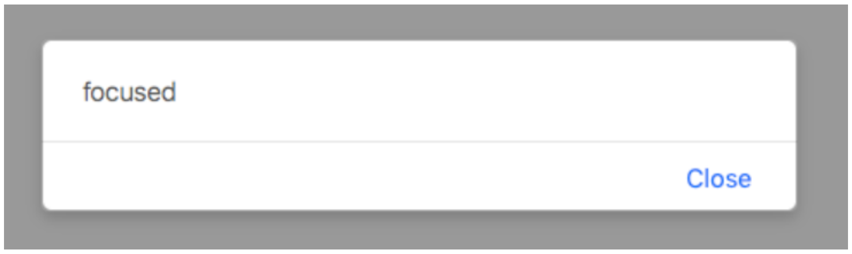
- In the case of an **<input>** element, this is when the cursor is placed inside the element (either by clicking or tapping on it or navigating to it using the keyboard).

EXAMPLE:

```
const input = form.elements.searchInput;

input.addEventListener('focus', () => alert('focused'), false);
```

Open **search.html** in your browser and place the cursor inside the input field. You should see an alert dialog similar to the one in the screenshot below.


focused

4)- Submitting a FORM

Possibly the most important form event is the submit event, occurring when the form is submitted.

Usually this will send the content of the form to the server to be processed.... but we can use JavaScript to intercept the form before it's sent by adding a submit event listener.

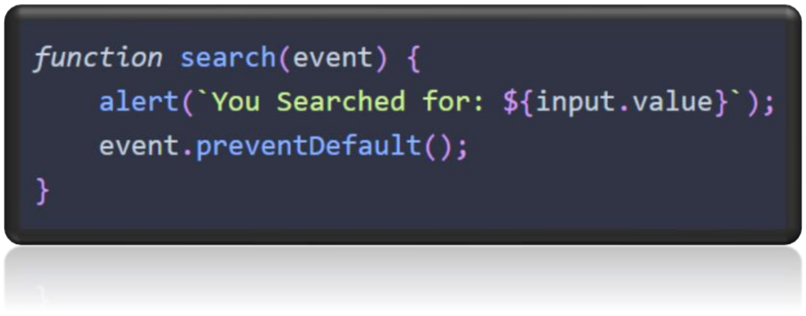
```
const form = document.forms['search'];
form.addEventListener ('submit', search, false);

function search() {
    alert(' Form Submitted');
}
```

5)- Retrieving and Changing Values From a FORM

Text input element objects have a value property that can be used to retrieve the text inside the field.

NOTE: We can use this to report back what the user has searched for



```
function search(event) {
    alert(`You Searched for: ${input.value}`);
    event.preventDefault();
}
```

6)- FORM Controls

Common types of **form control** are:

- <input> fields, including text, passwords, check boxes, radio buttons, and file uploads
- <select> menus for drop-down lists of options
- <textarea> elements for longer text entry
- <button> elements for submitting and resetting forms

7)- FORM Validation

Form validation is the process of checking whether a user has entered the information into a form correctly.

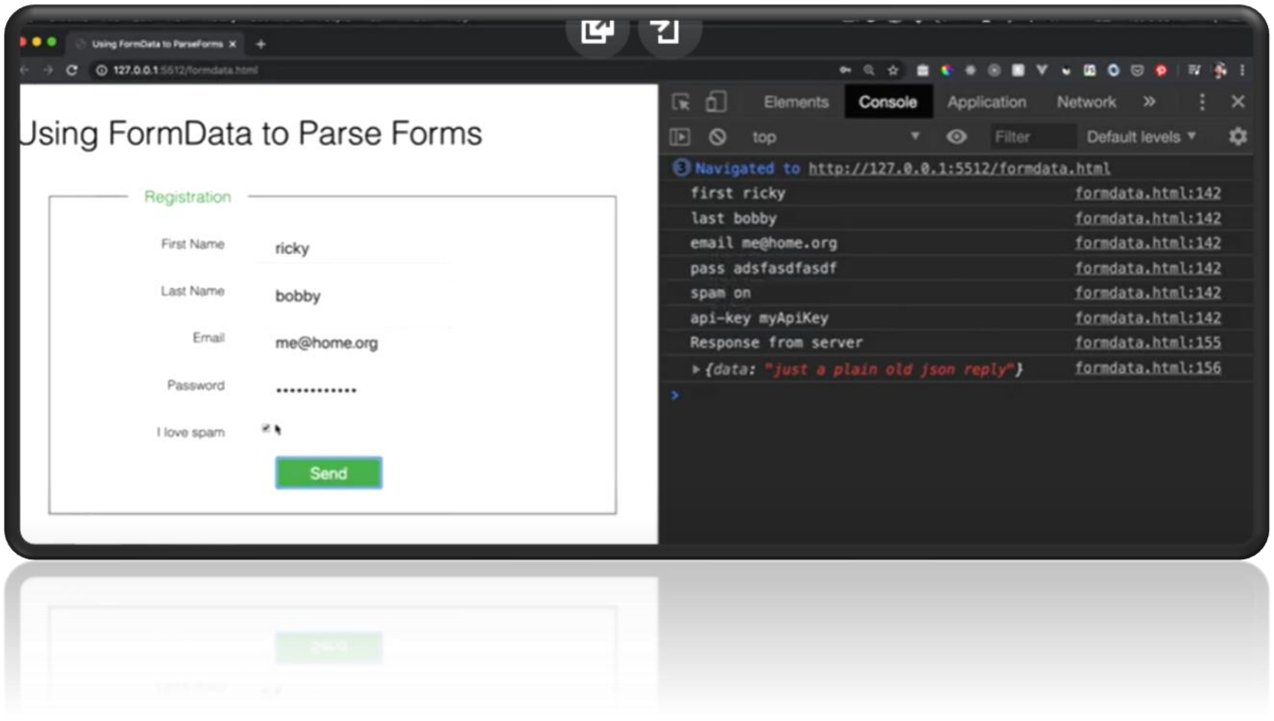
EXAMPLES:

- A required field is completed
- An email address is valid

-A number is entered when numerical data is required

-A password is at least a minimum number of characters

2.- Using FormData Objects Effectively



Using FormData to Parse Forms

Registration

First Name	ricky
Last Name	bobby
Email	me@home.org
Password	·············
I love spam	✓

```
Navigated to http://127.0.0.1:5512/formdata.html
first ricky                                   formdata.html:142
last bobby                                    formdata.html:142
email me@home.org                             formdata.html:142
pass adsfasdfasdf                             formdata.html:142
spam on                                       formdata.html:142
api-key myApiKey                              formdata.html:142
Response from server                          formdata.html:155
▸ {data: "just a plain old json reply"}       formdata.html:156
>
```

Any time you're going to work with JavaScript and you want to use JavaScript to handle your FORM, and you don't want the page to just automatically reload… We need to add prevent default on to the event.

- This is going to stop the FORM from actually trying to go off and get that new page and reload the page and replace the current page
- Always put ev.preventdefault()

To add more things that were not in the FORM:

- append()

3.- MDN: Client-Side Form Validation

Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format.

client-side form validation helps ensure data submitted matches the requirements set forth in the various form controls.

Form Validation –

When you enter data, the browser and/or the web server will check to see that the data is in the correct format and within the constraints set by the application.

If the information is correctly formatted, the application allows the data to be submitted to the server and (usually) saved in a database.

WHY DO WE INSIST ON VALIDATING OUR FORMS?

- We want to get the right data, in the right format
- We want to protect our users' data
- We want to protect ourselves

Types –

There are two different types of client-side validation that you'll encounter on the web: **Built-in** and **JavaScript**

Validating using BUILT-IN

This is done by using validation attributes on form elements:

- **required**: Specifies whether a form field needs to be filled in before the form can be submitted.
- **minlength** and **maxlength**: Specifies the minimum and maximum length of textual data (strings).
- **min** and **max**: Specifies the minimum and maximum values of numerical input types.
- **type**: Specifies whether the data needs to be a number, an email address, or some other specific preset type.
- **pattern:** Specifies a regular expression that defines a pattern the entered data needs to follow.

VALID ELEMENT

- The element matches the :valid CSS pseudo-class, which lets you apply a specific style to valid elements.
- If the user tries to send the data, the **browser will submit the form, provided there is nothing else stopping it from doing so**

INVALID ELEMENT

- The element matches the **:invalid** CSS pseudo-class, and sometimes other UI pseudo-classes (e.g., :out-of-range) depending on the error, which lets you apply a specific style to invalid elements.
- If the user tries to send the data, the **browser will block the form and display an error message.**

Validating using JavaScript

Most browsers support the Constraint Validation API, which consists of a set of methods and properties available on the following form element DOM interfaces