Front-End Web Development

Unit 11: Web Forms – Working with user data

Course Outline

1. Getting Started
2. HTML - Structuring the Web
3. CSS - Styling the Web
4. JavaScript - Dynamic client-side scripting
5. CSS - Making Layouts
6. Introduction to Websites/Web Applications
7. CSS - Advanced
8. Progress Review so far
9. JavaScript - Modifying the Document Object Model (DOM)
10. Dynamic HTML
11. **Web Forms - Working with user data**
12. JavaScript - Advanced
13. Building a Web Application with JavaScript
14. Introduction to CSS Frameworks – Bootstrap
15. Building a Web Application with Svelte
16. SEO, Web security, Performance
17. Walkthrough project

Course Learning Outcomes

- Competently write HTML and CSS code
- Create web page layouts according to requirements using styles
- Add interactivity to a web page with JavaScript
- Access and display third-party data on the web page
- Leverage Bootstrap and Static Site Generator

Assessment

- Final Project - 100% of the grade

  - Design and Build functioning Website using HTML5, CSS (including Bootstrap), JavaScript (browser only)

✔ Code will be managed in GitHub
✔ Website will be deployed to GitHub Pages
✔ All code to follow best practice and be documented

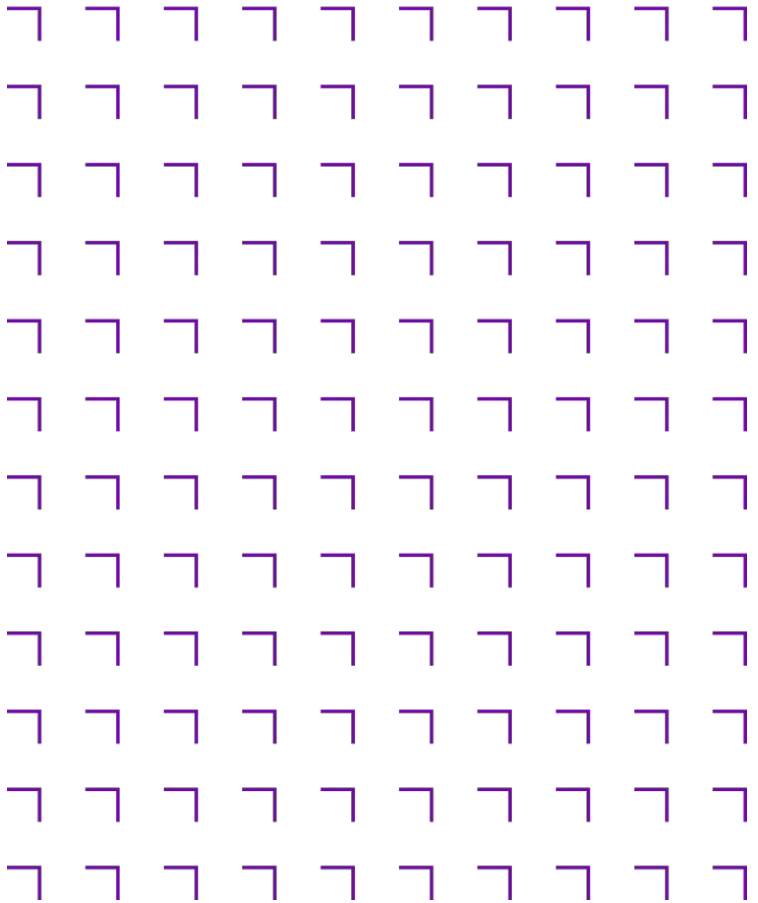
- Details and How-To-Guide are available on the course page under the section called Assessments

10. Web Forms – Working with user data

In This Unit

Title
Web forms and the common input types
Styling Web Forms
Adding JavaScript to web forms

Web forms and the common input types

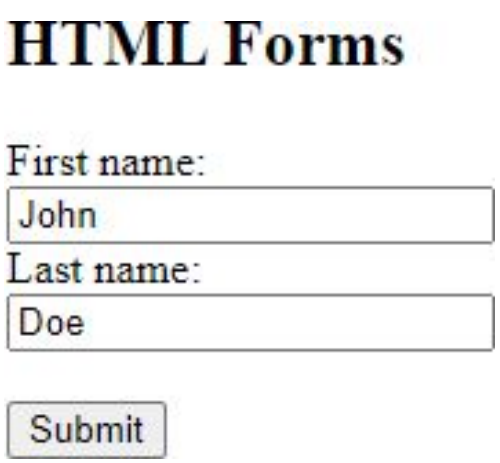


What is a web form?

- 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
- However the submission of the web form can also be intercepted using JavaScript and processed in the browser
- An HTML Form is made of one or more input elements:
  - text fields (single line or multiline)
  - select boxes
  - buttons
  - checkboxes
  - radio buttons, or textarea
  - submit button

Forms - example

```
<h2>HTML Forms</h2>
<form action="/action_page.php" method="post" name="login" autocomplete="on">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```



Form Elements - `<form>`

The HTML `<form>` element is used to create an HTML form for user input

It is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc...

Attribute	Description
action	Specifies where to send the form-data when a form is submitted
autocomplete	Specifies if the browser should display options to fill in the field, based on earlier typed values
enctype	Specifies how the form-data should be encoded when submitting it to the server (only for method="post")
method	Specifies the HTTP method to use when sending form-data, 'get' or 'post'
name	Specifies the name of the form
novalidate	Specifies that the form should not be validated when submitted
target	Specifies where to display the response that is received after submitting the form

Form elements - `<form>` method

- **GET**:
  - Appends the form data to the URL, in name/value pairs
  - NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
  - The length of a URL is limited (2048 characters)
  - Useful for form submissions where a user wants to bookmark the result
  - GET is good for non-secure data, like query strings in Google and is the default
- **POST**:
  - Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
  - POST has no size limitations, and can be used to send large amounts of data.
  - Form submissions with POST cannot be bookmarked

Form elements -`<input>`

- One of the most used form elements is the `<input>` element
- The `<input>` element can be displayed in several ways, depending on the type attribute
  - e.g. button, checkbox, color, email
- Common attributes include:
  - id - defines a unique identifir for the input
  - name - name of the input element
  - type - the type of input element to be created
  - value - value of the input element
  - title - sets a title that displays as a tooltip
  - required - sets the input to required field
  - readonly - make the input read only
  - autofocus - sets the focus on the element after page loads

Forms - `<input type="text">`

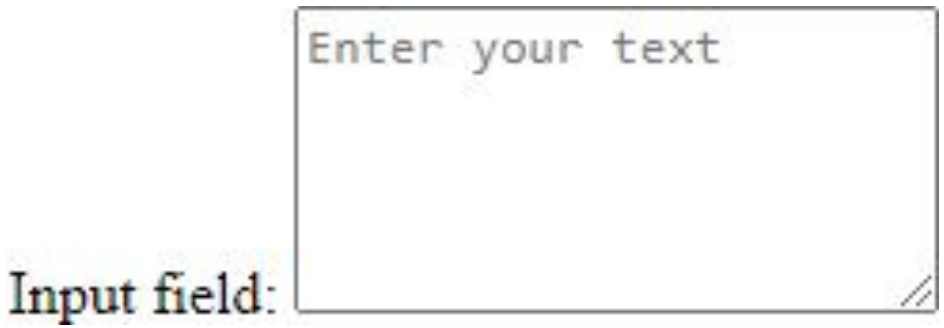
- Accepts text information
  Closely related types include: email, phone, number, search, url
-
- Input attributes:
  - type="text", name, id, size, maxlength, value, placeholder, required
- `<label>` associates a text label with a form control
  - attributes: **for** - same value as the input id to bind them together

```
<label for="input-text-field">Input field:</label>
<input id="input-text-field" name="input-text-field" type="text" placeholder="Enter your text" required>
/* or */
<label>Input field:
  <input name="input-text-field" type="text" placeholder="Enter your text" required>
</label>
```

Forms - `<textarea>`

- Scrolling text box
- Attributes: name, id, cols, rows
- browser defaults will be used if cols and rows not specified

```
<label for="input-text-field">Input field:</label>
<textarea name="input-text-field" placeholder="Enter your text" required rows="5"></textarea>
```



Forms - `<input type="submit">`

- Submits the form information
- When clicked, triggers the action method on the `<form>` tag
- Sends the form data (the name="value" pair for each form element) to the web server
- Attributes: name, id, value

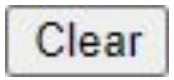
```
<input id="input-submit-field" type="submit" value="Submit"> /* defines a submit button with text */
<input type="image" src="images/img_submit.gif" alt="Submit" width="48" height="48"> /* defines a submit button with an image
*/
```

Submit

Forms - `<input type="reset">`

- Resets the form fields to their initial values
- Attributes: name, id, value

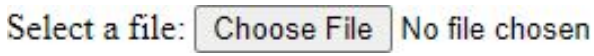
```
<input id="input-reset-field" type="reset" value="Clear">
```



Forms - `<input type="file">`

- Accepts a file upload when the form is submitted
- Requires the form element to be configured with method="post", enctype="multipart/form-data“, and the action attribute
- Attributes: name, id, accepts, capture, multiple

```
<label for="myfile">Select a file:</label>
<input type="file" id="myfile" name="myfile">
```



Forms - `<input type="password">`

- Accepts text information that needs to be hidden as it is entered, e.g. a password
- Attributes: name, id, size, maxlength, value

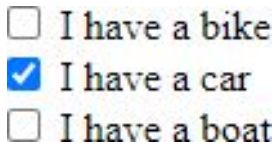
```
<label for="input-text-field">Password field:</label>
<input name="input-text-field" type="password" placeholder="Enter your password" required>
```

Password field: Enter your password

Forms - `<input type="checkbox">`

- Allows the user to select one or more of a group of predetermined items
- Attributes: name, id, checked, value

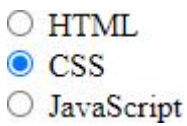
```
<input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
<label for="vehicle1"> I have a bike</label><br>
<input type="checkbox" id="vehicle2" name="vehicle2" value="Car" checked>
<label for="vehicle2"> I have a car</label><br>
<input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
<label for="vehicle3"> I have a boat</label>
```



Forms - `<input type="radio">`

- Allows the user to select exactly one from a group of predetermined items
  Each radio button in a group is given the same name and a unique value
-
- Attributes: name, id, checked, value

```
<input type="radio" id="html" name="fav_language" value="HTML">
<label for="html">HTML</label><br>
<input type="radio" id="css" name="fav_language" value="CSS" checked>
<label for="css">CSS</label><br>
<input type="radio" id="javascript" name="fav_language" value="JavaScript">
<label for="javascript">JavaScript</label>
```



Forms - `<input type="hidden">`

- This form control is not displayed on the web page
- Can be accessed by both client-side and server-side scripting
- Sometimes used to contain information needed as the visitor moves from page to page, e.g. form tracking id
- Attributes: name, id, value

```
<input type="hidden" id="custId" name="custId" value="3487">
```

Forms - `<select>`

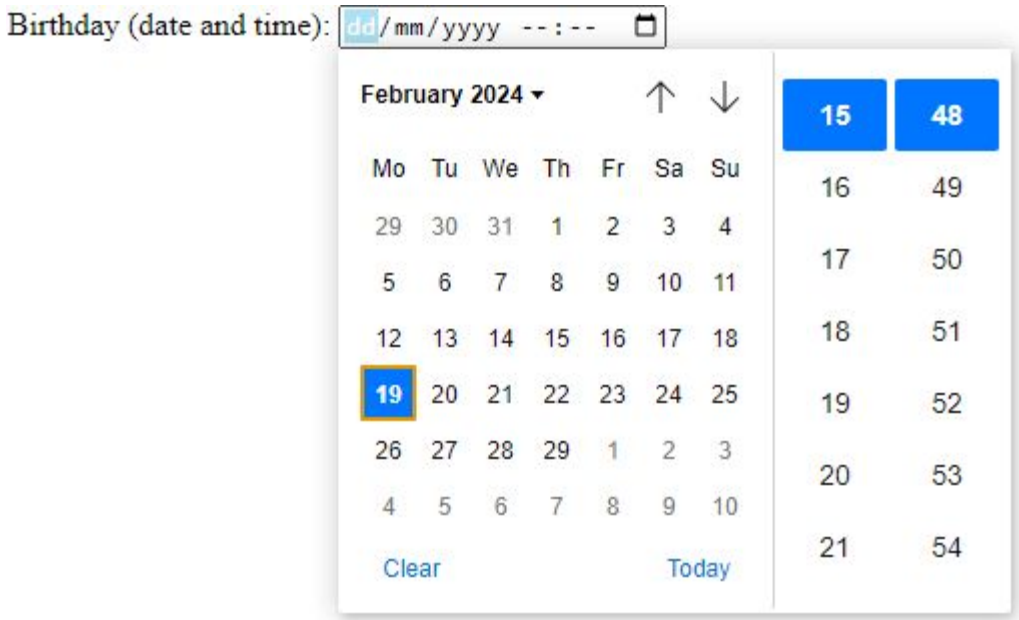
- Configures a select list (along with <option> elements)
- Also known as: Select Box, Drop-Down List, Drop-Down Box, and Option Box
- Allows the user to select one or more items from a list of predetermined choices
- Select attributes: name, id, size, multiple
- Use the <option> tag to specify the various options for the <select> list
  - Option attributes: value, selected

```
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat" selected>Fiat</option>
  <option value="audi">Audi</option>
</select>
```

Forms - `<input type="datetime-local">`

- Configures a date/time widget which depends on the browser
- Closely related types include: date, month, week, time
- Attribute: id, name

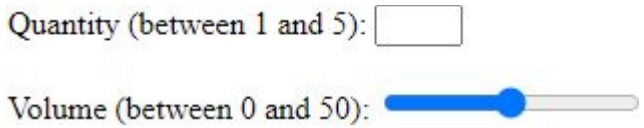
```
<label for="birthdaytime">Birthday (date and time):</label>
<input type="datetime-local" id="birthdaytime" name="birthdaytime">
```



Forms - `<input type="number">` , `<input type="range">`

- Configures a spinner or slider contral
- More usabale interface than a text input tag
- Attributes: id, name, min, max

```
<label for="quantity">Quantity (between 1 and 5):</label>
<input type="number" id="quantity" name="quantity" min="1" max="5">
<br><br>
<label for="vol">Volume (between 0 and 50):</label>
<input type="range" id="vol" name="vol" min="0" max="50">
```



Forms - `<input type="color">`

- Configures an input field that should contain a color
- Depending on browser support, a color picker can show up in the input field

```
<label for="favcolor">Select your favorite color:</label>
<input type="color" id="favcolor" name="favcolor">
```

Forms - `<input list="">` , `<datalist>`

- The input list attribute refers to a `<datalist>` element that contains pre-defined options for an `<input>` element Text
- can also be entered if there is no suitable option

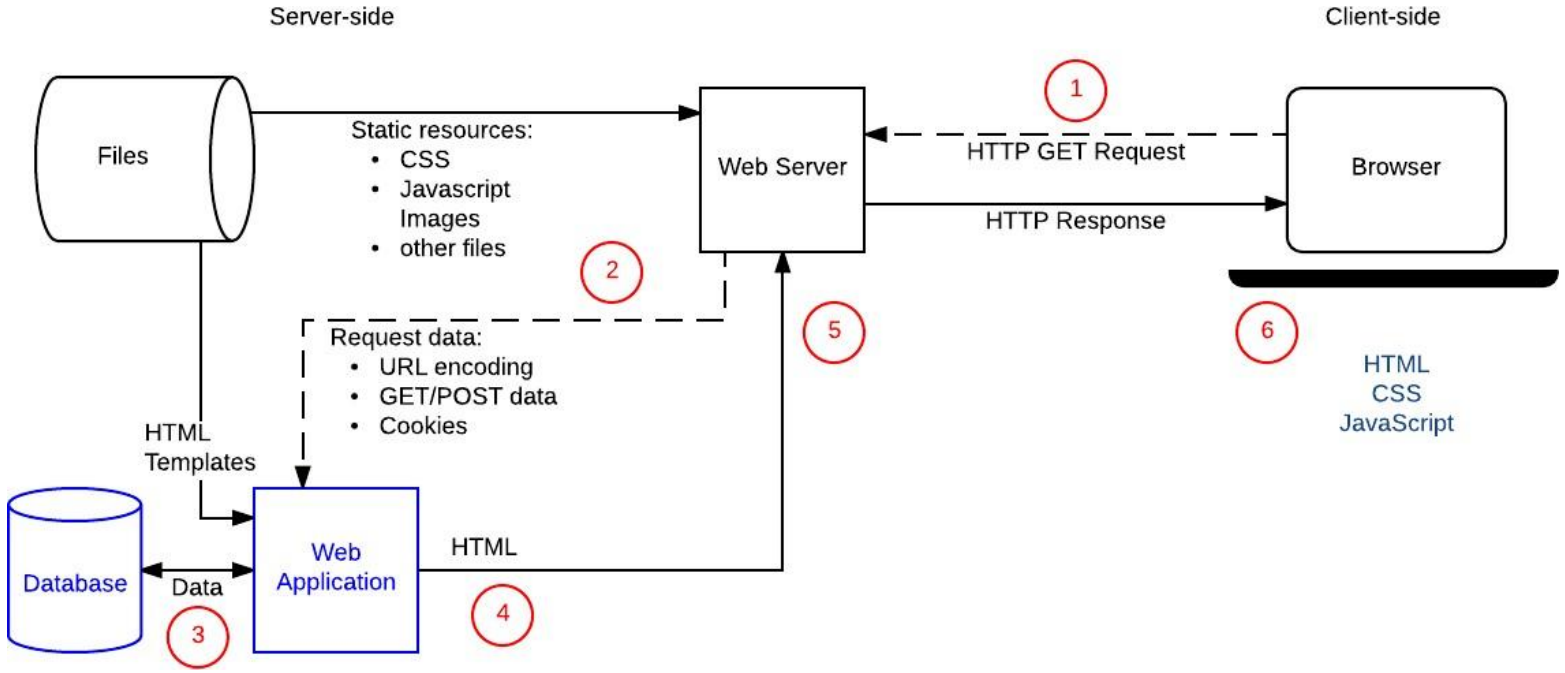
```
<label for="browser">Select a browser:</label>
<input list="browsers" name="browser" id="browser">
<datalist id="browsers">
  <option value="Edge">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

Forms - accessibility

- `<label>` element
  - Associates a text label with a form control
- `<fieldset>` element
  - Creates a visual group of form elements on a web page
- `<legend>` element
  - Creates a text label within the fieldset
- `tabindex` attribute
  - Attribute that can be used on form controls and anchor tags. Modifies the default tab order
- `accesskey` attribute
  - Attribute that can be used on form controls and anchor tags
  - Create a “hot-key” combination to place the focus on the component
  - Assign a value of a keyboard letter
  - On Windows use the CTRL and the “hot-key” to move the cursor

Forms - Processing

1. In the web form, user clicks on the "Submit" button. The form data is then sent via the http protocol to the web server
2. The web server will search for a script/endpoint that matches the URI specified in the form action
3. The script/endpoint will then process the form data
4. A reply will then be sent back to the web server
5. The web server will then send the html back to the browser
6. The html reply is then displayed in the browser



Forms - Server-Side Scripting

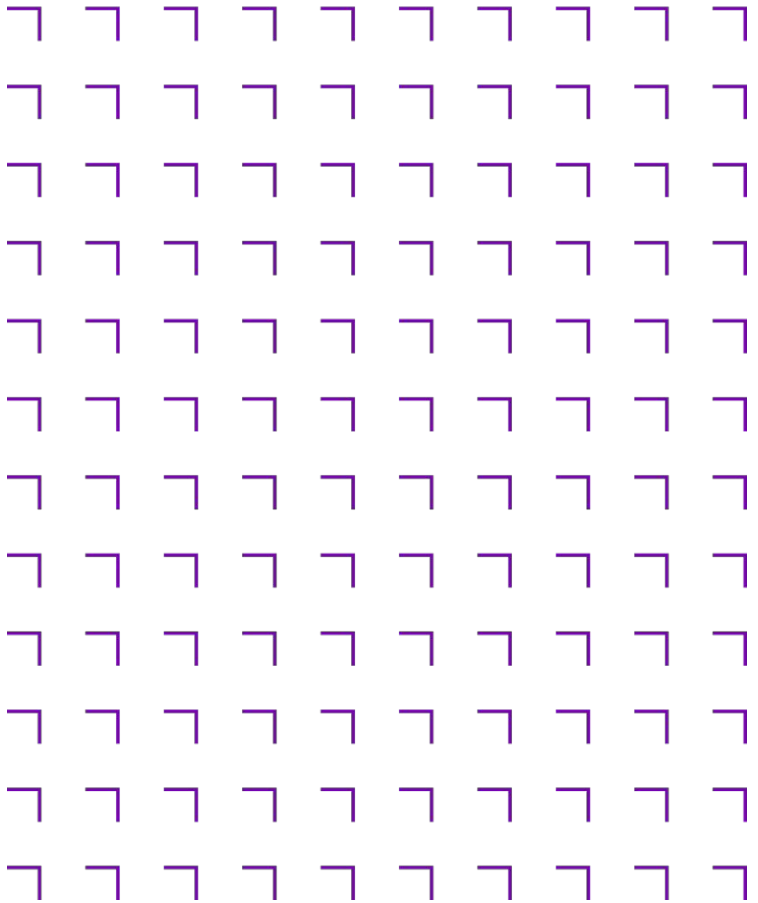
- Outside the scope of this course https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction
- A generic form processor (php) will be used for this course -> https://learning.postman.com/docs/developer/echo-api/
- Any form data sent to this URl/endpoint will be acknowledged and a reply will be sent
- Both get & post methods are supported. Reply is in json format

Activity

Breakout

- Join a breakout room
- Download the unit 10 exercises code from Moodle
- Follow the instructions and complete the exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour

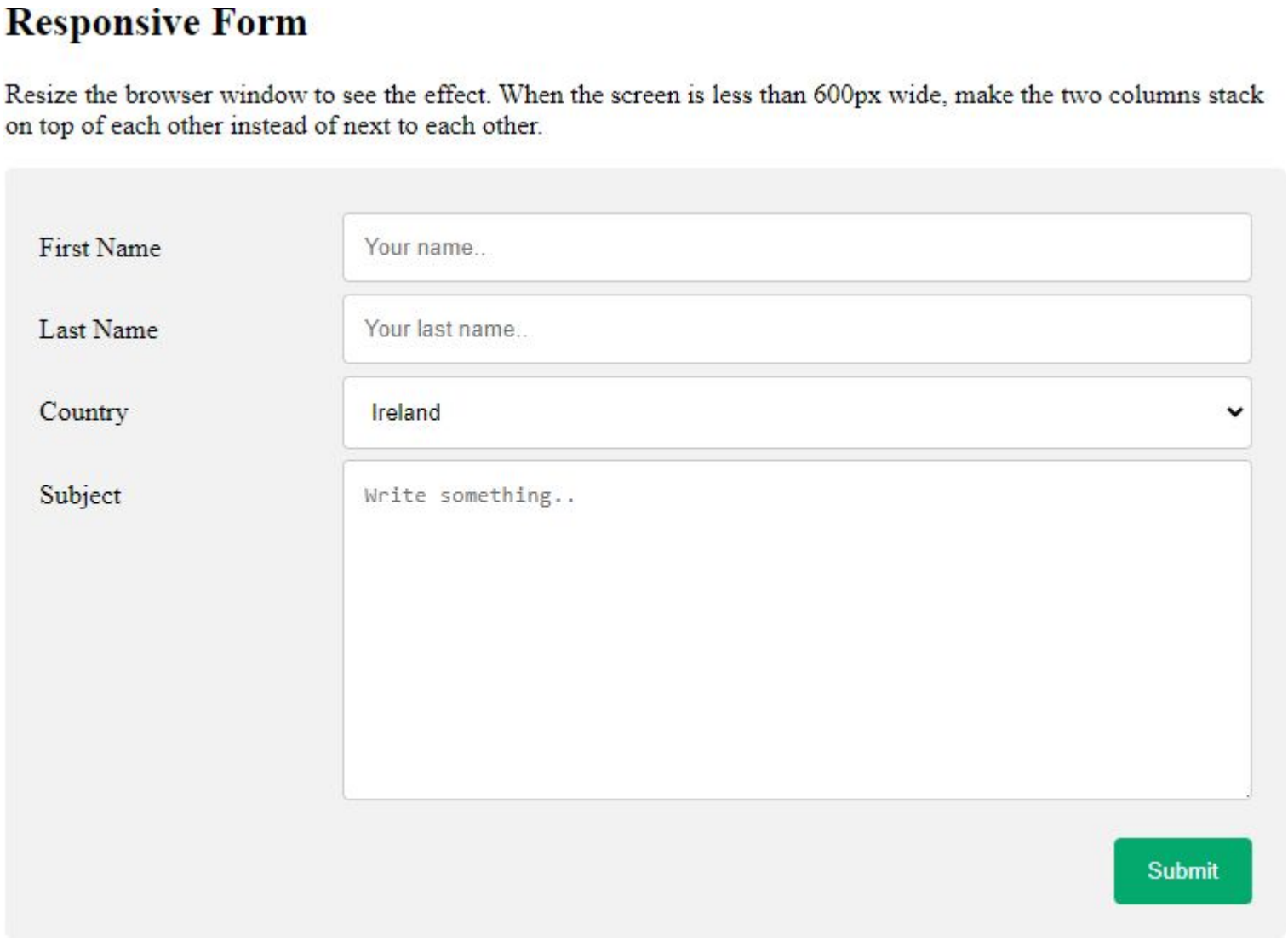
Styling Web forms



Styling Web Forms

- Initially browsers relied on underlying operating system to render form widgets
- Form widgets are now mostly stylable, with a few exceptions
- **Easy-to-style**
  `<form>`, `<fieldset>` and `<legend>`, single-line text `<input>`s (e.g. type text, url, email), except for `<input type="search">`, Multi-line `<textarea>`, buttons (both `<input>` and `<button>`), `<label>`, `<output>`
- **Harder-to-style**
  Checkboxes and radio buttons, `<input type="search">`
- **Having internals can't be styled in CSS alone**
  - `<input type="color">`, Date-related controls such as `<input type="datetime-local">`, `<input type="range">`, `<input type="file">`
  - Elements involved in creating dropdown widgets, including `<select>`, `<option>`, `<optgroup>` and `<datalist>`, `<progress>` and `<meter>`

Styling Web Forms - using CSS float



Styling Web Forms - Example

```
<section class="container">
  <form action="/action_page.php">
    <div class="row">
      <div class="col-25"><label for="fname">First Name</label></div>
      <div class="col-75">
        <input type="text" id="fname" name="firstname" placeholder="Your name.." required>
      </div>
    </div>
    <div class="row">
      <div class="col-25"><label for="lname">Last Name</label></div>
      <div class="col-75">
        <input type="text" id="lname" name="lastname" placeholder="Your last name.." required>
      </div>
    </div>
    <div class="row">
      <div class="col-25"><label for="country">Country</label></div>
      <div class="col-75">
        <select id="country" name="country">
          <option value="ireland">Ireland</option>
          <option value="england">England</option>
          <option value="spain">Spain</option>
        </select>
      </div>
    </div>
    <div class="row">
      <div class="col-25"><label for="subject">Subject</label></div>
      <div class="col-75">
        <textarea id="subject" name="subject" placeholder="Write something.." style="height:200px"></textarea>
      </div>
    </div>
```

Styling Web Forms - Layout

```
* {
  box-sizing: border-box; /* set box-sizing to make it easier to set the widths */
}
.container {   /* style the form background */  border-radius: 5px;
  background-color: #f2f2f2;  padding: 20px;
}
.col-25 { /* set the left column/label width */  float: left;
  width: 25%;
  margin-top: 6px;
}
.col-75 { / * set the right column/inputs width */  float: left;
  width: 75%;  margin-top: 6px;
}
.row::after { /* Clear floats after the columns */  content: "";
  display: table;
  clear: both;
}
```

Styling Web Forms - Labels and Controls

```
input[type=text], select, textarea {
  width: 100%;
  padding: 12px;
  border: none;
  border-radius: 4px;
  resize: vertical;
}
input[type=text]:focus, select:focus, textarea:focus {
  border: 1px solid #ccc;
  background: #f8f8f8;
  outline-color: #d3d3d3;
}
label {
  padding: 12px 12px 12px 0;
  display: inline-block;
}
input[type=submit] {
  background-color: #04AA6D;
  color: white;
  padding: 12px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  float: right;
}
input[type=submit]:hover {
```

Styling Web Forms - Responsive

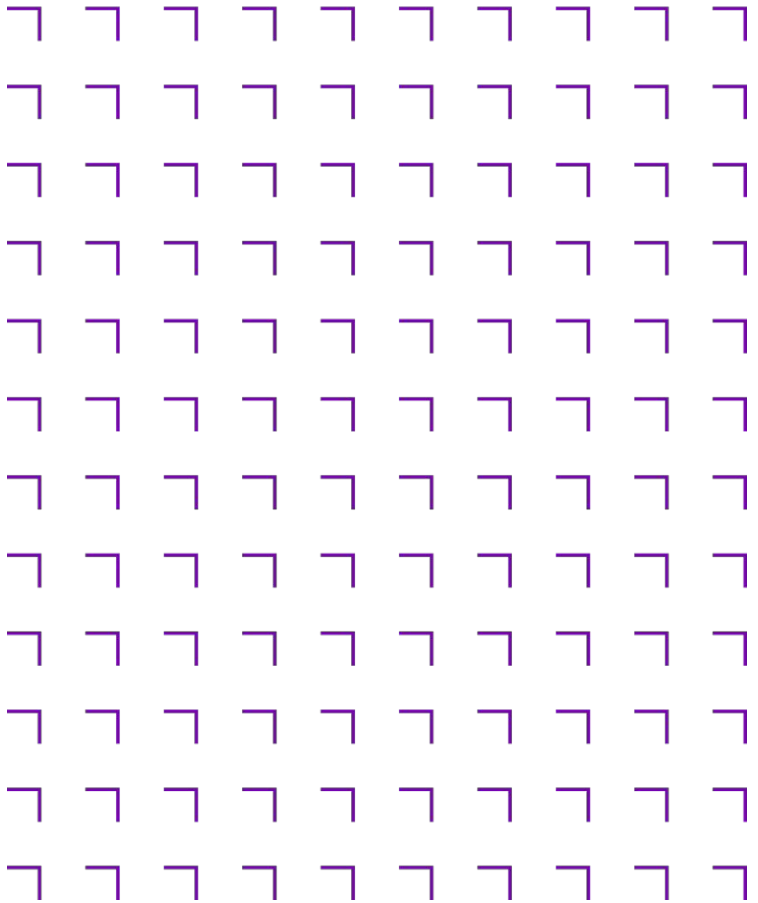
```
/* Responsive layout - when the screen is less than 600px wide,
make the two columns stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .col-25, .col-75, input[type=submit] {
    width: 100%;
    margin-top: 0;
  }
}
```

Activity

Breakout

- Join a breakout room
- Continue working on the unit 10 exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...
- Will start next topic on the hour

Add JavaScript to web forms



Adding Javascript to web forms

There are three main use cases for using JavaScript with Web Forms:

- Form validation
- Submitting a web form
- Client side processing

Form Validation using JavaScript

- Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format
- HTML 5 provide native support for form validation using attributes such as `required` and new input types, e.g. `<input type="email">`
- is an important feature of good user experience as issues can be caught and fixed straight away
- Service-side validation is still required as client-side validation can be bypassed, e.g. by modifying the html

Examples

- "This field is required" (You can't leave this field blank)
- "Please enter your phone number in the format xxx-xxxx" (A specific data format is required for it to be considered valid)
- "Please enter a valid email address" (the data you entered is not in the right format)
- "Your password needs to be between 8 and 30 characters long and contain one uppercase letter, one symbol, and a number." (A very specific data format is required for your data)

Form Validation - Example 1

```
<label for="inputBox">Enter something in the box: </label>
<input type="text" name="inputbox" id="inputBox" value="" />
<p id="msg"></p>
```

```
let inputBox = document.querySelector('#inputBox');
// 'input' event is fired when the value of the input field changes
inputBox.addEventListener('input', (event) => {
  if (["frodo","bilbo","sam"].includes(event.target.value)){
    document.querySelector('#msg').innerHTML="Already have that Hobbit";
  } else {
    document.querySelector('#msg').innerHTML="";
  }
})
```

Form Validation - Example 2

```
<p>Please input a number between 1 and 10:</p>
<input id="numb">
<button id="button" type="button" >Submit</button>
<p id="msg"></p>
```

```
const button = document.querySelector('#button');
// <button> does not generate a 'submit' event. Use 'click' instead
button.addEventListener('click', myFunction );

function myFunction() {
  // Get the value of the input field with id="numb"
  let x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  let text;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("msg").innerHTML = text;
}
```

Form Validation - Example 3

```
<p>Please input a password between 8 and 15:</p>
<input id="password" type="password">
<button id="button" type="button" >Submit</button>
<p id="msg"></p>
```

```
const button = document.querySelector('#button');
button.addEventListener('click', (event) => {
  let password = document.getElementById("password").value;
  document.querySelector('#msg').innerHTML= checkStrength(password);
} );

function checkStrength(pass) {
  const strength = { 1: "very Weak", 2: "Weak", 3: "Meduim", 4: "Strong",};
  if (pass.length > 15) { return pass + " Password is too lengthy"
  } else if (pass.length < 8) { return pass + " Password is too short" }

  let regex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@.#$!%*?&])[A-Za-z\d@.#$!^%*?&]{8,15}$/;
  if (regex.test(pass)) { return pass + " Password is strong"; }
  if (/[a-z]/.test(pass)) count++;
  if (/[A-Z]/.test(pass)) count++;
  if (/[\d]/.test(pass)) count++;
}
  if  (/[!@#$%^&*.?]/.test(pass))  count++;
  return pass + "Pasword is " + strength[count];
```

Forms - Submit using JavaScript

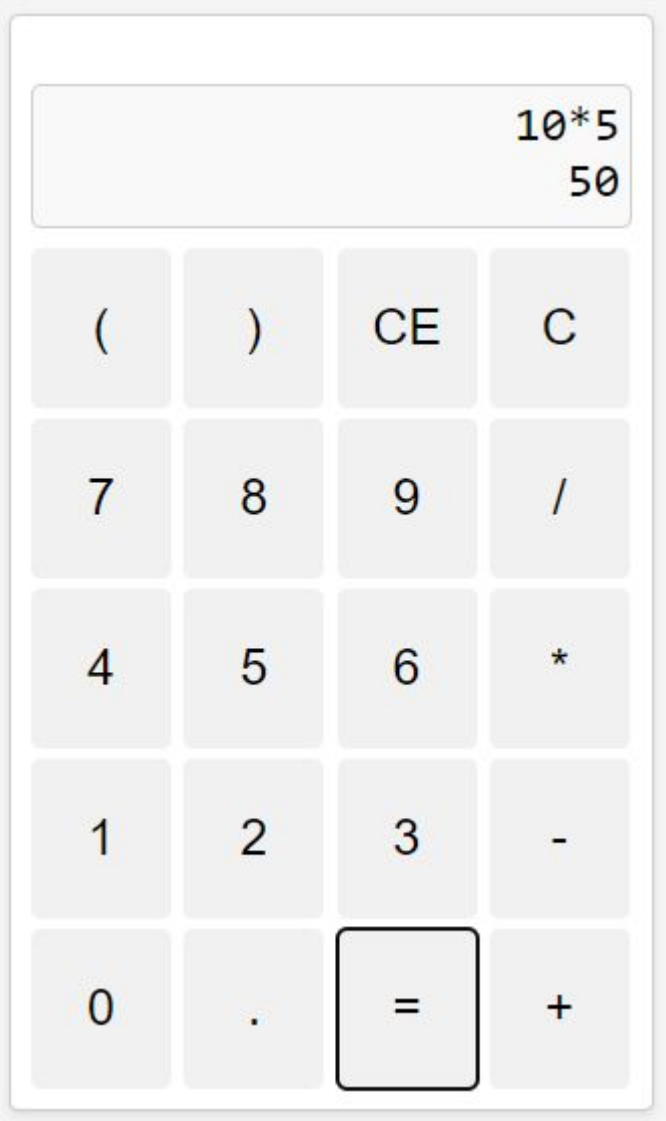
- When a user submits an HTML form, for example by clicking the submit button, the browser makes an HTTP request to send the data in the form
- Sometimes it's useful to intercept the submit, for example form validation
- Also possible to use JavaScript APIs such as `fetch()` to send data programmatically to an endpoint but we'll look at that in a future lecture

Form - Submit example

```
<form id="form" action="#">
  <label for="number">Please input a number between 1 and 10:</label><br>
  <input id="number" name="number"><br>
  <p id="msg"></p>
  <input id="submit" type="submit" value="Submit"></input>
</form>
```

```
const form = document.querySelector('#form');
form.addEventListener('submit', (event) => {
  myFunction(event, form);
});
function myFunction(event, form) {
  let x = form.querySelector("#number").value;
  let text;
  // If x is Not a Number or less than one or greater than 10
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
    event.preventDefault();
  } else {
    text = "Input OK";
  }
  form.querySelector("#msg").innerHTML = text;
}
```

Form - Client-Side processing - Calculator



Form - Client-Side processing - Calculator

- Use of `load` event on `window` object to ensure page has fully loaded
- Event delegation (single event handler)
- Use of `event` object to get user input
- Use of `Function()` to execute JS code

Form - Client-Side processing - Calculator

```
window.addEventListener("load", (event => {
  calcButtons.addEventListener('click', (event) => {
    if (event.target.tagName.toLowerCase() === 'button') {
      switch (event.target.textContent) {
        case 'CE':
          clearLastElement();
          break;
        case 'C':
          clearDisplay();
          break;
        case '=':
          calculateResult();
          break;
        default:
          // pass value of the clicked button to appendToDisplay()
          appendToDisplay(event.target.textContent);
      }
    }
  });
}));
```

Form - Client-Side processing - Calculator

```
function calculateResult() {
  try {
    // execute the currentDisplay string as JS code (recommended). Alternative: eval()
    const result = Function("return " + currentDisplay)();
    currentDisplay += `\n${result.toString()}`;
    updateDisplay();
  } catch (error) {
    currentDisplay += "\nError";
    updateDisplay();
  }
  resultDisplay=true;
}
```

Form - Client-Side processing - User Profile

Account Setting

Email: joe.bloggs@gmail.com

Username: Joe Bloggs

Password: ••••••••

Confirm Password: ••••••••

Profile Setting

First Name: Joe

Last Name: Bloggs

Gender: (•) Male ( ) Female

Birthday: 06/01/2010

Form - Client-Side processing - User Profile

Aspects of JS demonstrated in the code

- IIFE (Immediately Invoked Function Expression)
  - Immediate Execution
  - Encapsulation
  - Anonymous or Named. Example shown is anonymous
- Caching of DOM objects, e.g. userProfile
- Two options for getting form data
  - `value` property
  - `FormData` - provides a way to construct a set of key/value pairs representing form fields and their values
- `localStorage` API - example of storing user data locally

Form - Client-Side processing - User Profile

```
// code is wrapped in an IIFE (Immediately Invoked Function Expression).
//See https://developer.mozilla.org/en-US/docs/Glossary/IIFE for more details
//
(() => {
  // globals
  const userProfile = document.querySelector('.user-profile');
  const userProfileForm = userProfile.querySelector('form');  const
  userProfileSubmit = userProfile.querySelector('#submit');  const
  userProfileLSName = 'userProfileFormData';

  let userProfileFormData = new FormData(userProfileForm, userProfileSubmit);
  let userProfileLocalFormData = {};

  // rest of the code
  ...
})();
```

Form - Client-Side processing - User Profile

```
function getFormData() {
  // option 1 - using form elements
  const formElements = userProfileForm.elements;
  for (const element of formElements) {
    const elementName = element.getAttribute('name');
    if (elementName) {
      console.log(element.getAttribute('name'), element.value);
    }
  }

  // option 2 - FormData - https://developer.mozilla.org/en-US/docs/Web/API/FormData
  userProfileFormData = new FormData(userProfileForm, userProfileSubmit); // update with latest from data
  userProfileLocalFormData = {};
  for (let [key, value] of userProfileFormData) {
    if (userProfileLocalFormData.hasOwnProperty(key)) {
      value += ',' + userProfileLocalFormData[key];
    }
    userProfileLocalFormData[key] = value;
  }
}
```

Form - Client-Side processing - User Profile

```
function updateFormData()
  {
  if(userProfileFormData)
  {
    Object.entries(userProfileLocalFormData).forEach(([key, value]) => {
      // certain input types such as radio contain more than one element
      const fieldElements =
      userProfileForm.querySelectorAll(`[name=${key}]`);  for (const
      fieldElement of Array.from(fieldElements) ) {
        const tagName = fieldElement.tagName.toLowerCase();
        //for (const value of values.split(',')) { // field element may contain multiple values, e.g. select
          multiple.  switch (tagName) {
            case 'input':
              if (fieldElement.getAttribute('type').toLowerCase() === 'radio' )
                {  fieldElement.setAttribute('checked', '');
              } else {
                fieldElement.value = value;
              }
              break;
            case 'textarea':
              fieldElement.innerHTML =
              value;  break;
            case 'select':
              fieldElement.value =
              value;
            default:
              // do nothing
          }
```

Form - Client-Side processing - User Profile

```
function init(){
  // check localstorage and load form data if present
  const lsFormData = localStorage.getItem(userProfileLSName);
  if (lsFormData) {
    userProfileLocalFormData = JSON.parse(lsFormData);
  }
  updateFormData();
}

window.addEventListener("load", (event => {
  init();
  // submit event handler
  userProfileForm.addEventListener('submit', (event) => {
    event.preventDefault();
    getFormData();
    localStorage.setItem(userProfileLSName, JSON.stringify(userProfileLocalFormData)); // store form data to localStorage
  });
  // reset event handler
  userProfileForm.addEventListener('reset', (event) => {
    userProfileLocalFormData = {}; // clear local form data object
    localStorage.removeItem(userProfileLSName); // clear localStorage
    resetLocalFormData(); // clear local form data
    updateFormData(); // update html form
  });
}));
```

Activity

Breakout

- Join a breakout room
- Continue working on the unit 10 exercises
- You have 35 minutes
- Lecturer will visit each room in turn, etc...

Summary

Completed this Week

- Web forms and the common input types
- Styling Web Forms
- Adding JavaScript to web forms

For Next Week

- Complete the remaining exercises for unit 10 before next class
- Review the slides and examples for unit 11