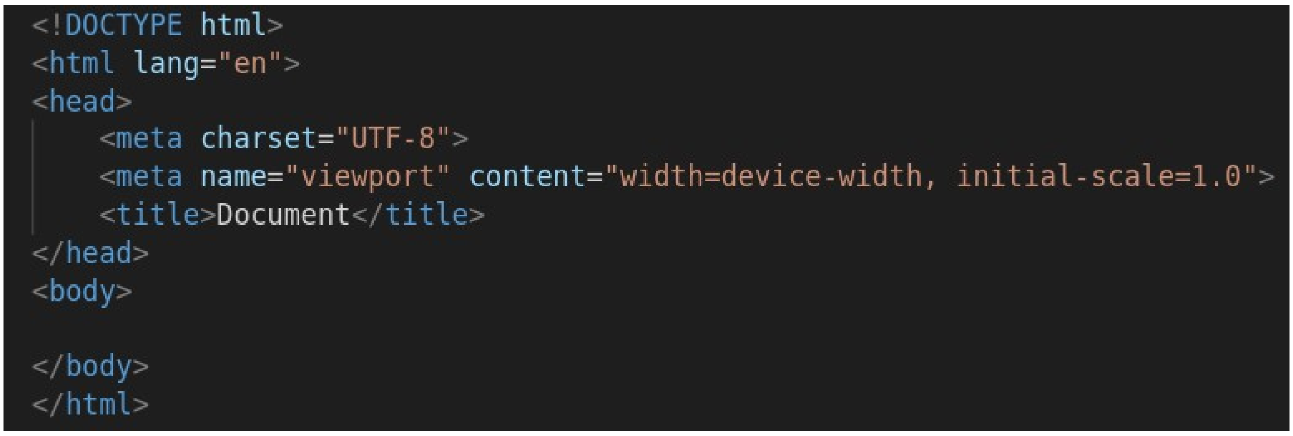
ISCG6420 Semester 1 2024

Exercise: Creating a User Form

This exercise will take you through creating an HTML form for the purposes of creating user data objects for a hypothetical application.

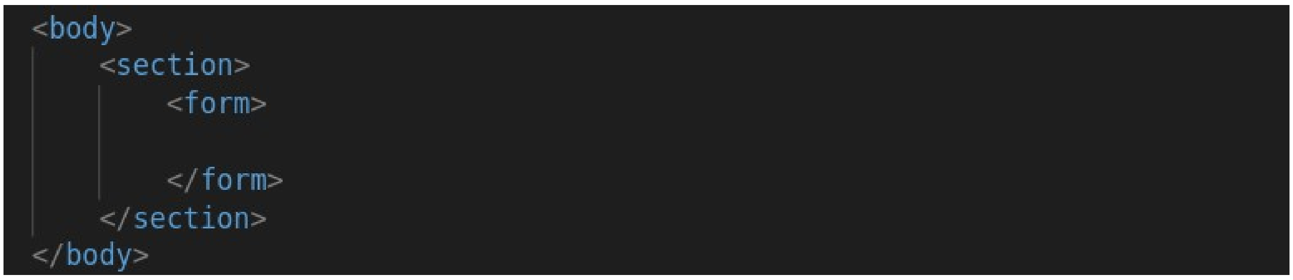
Create a new webpage

Create a new website project or open an existing project in your code editor. In the root directory create a new “userform.html” file and open it for editing.
Insert a website base template (VSCode shortcut = “html:5”).



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

In the body, create a section with a form inside.



```html
<body>
    <section>
        <form>

        </form>
    </section>
</body>
```

The app requires this form to collect the following user data:
- Full name (first name, last name)
- Email address
- Profile colour
- Phone number
- Billing address

Before adding all input fields to the form, a prototype should be created to minimise wasted effort. The user name is all we need for prototyping.

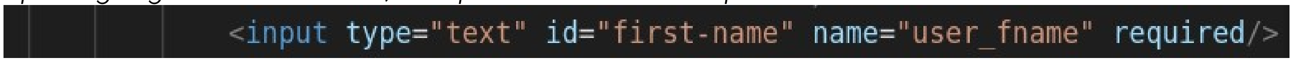
Populate the form

HTML forms are a container that can be populated with elements such as input fields. The form provides a way to group a set of inputs and link them to an action. This is typically used to send the form data to a backend application.

Assign an ID to the form element. This will provide easier access from JavaScript in a later step.

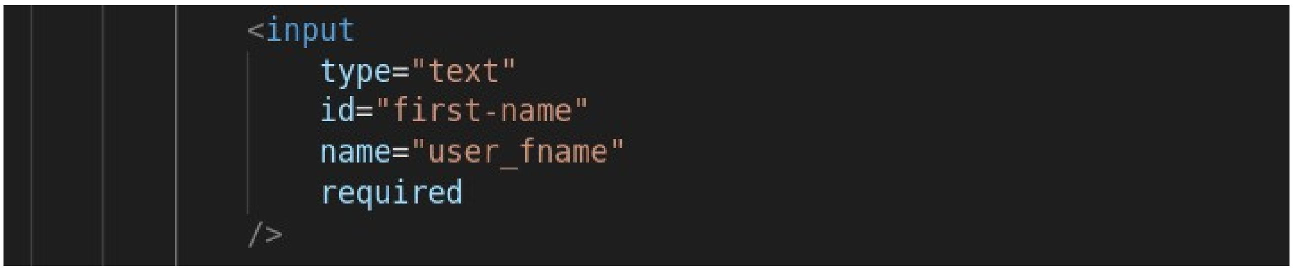
Create two pairs of label and input elements One for the first name, and one for the last name. Set the attributes of the input fields to the following:
- type = “text”
- ID = “first-name” or “last-name”
- Name = “user_” + summarised field name
- required

Note: you can make elements with many attributes easier to read by treating the element opening tag as a code block, and put attributes on separate lines.



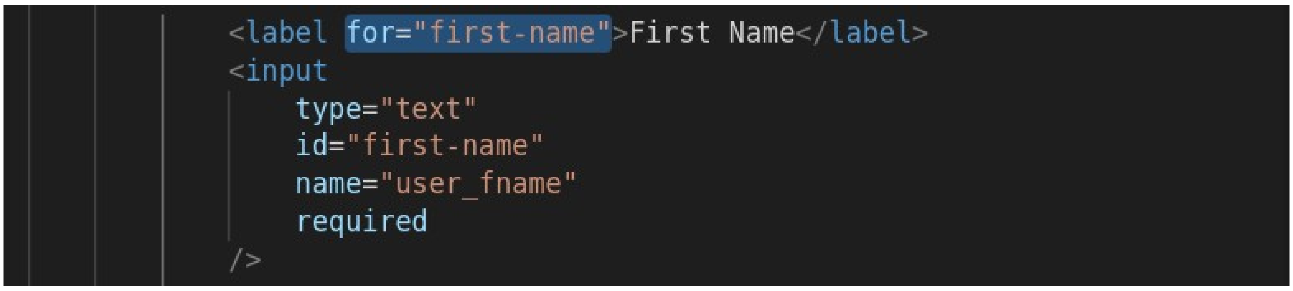
```html
<input type="text" id="first-name" name="user_fname" required/>
```

VS



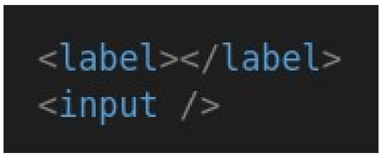
```html
<input
    type="text"
    id="first-name"
    name="user_fname"
    required
/>
```

Link the labels to the corresponding input fields by setting the “for” attribute to the input field ID:

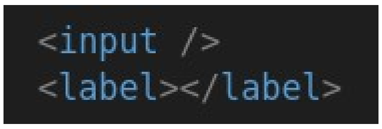


```html
<label for="first-name">First Name</label>
<input
    type="text"
    id="first-name"
    name="user_fname"
    required
/>
```

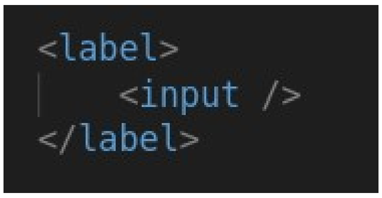
In this form the label will be displayed before the input field. The label element does not have to be above the input element in HTML. For example:



```html
<label></label>
<input />
```



```html
<input />
<label></label>
```

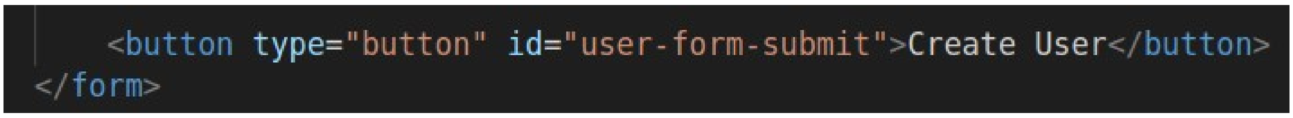


```html
<label>
    <input />
</label>
```

Submittable form

To identify the form data as ready to use, the user typically clicks a submit button.

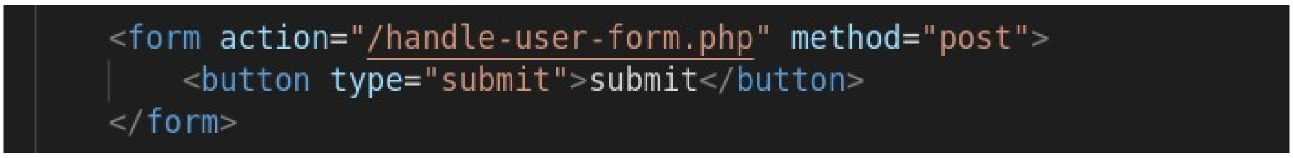
Add a button with attribute type=”button” and id=”user-form-submit” to the bottom of the form:



```
    <button type="button" id="user-form-submit">Create User</button>
</form>
```

When sending data from a form to a backend endpoint, the preferred button type is “submit”. This will send the form data to an endpoint provided in the “action” form attribute.

In this example, the form data will be sent to /handle-user-form.php when the button is clicked. <u>Do not add this to your project</u>



```
    <form action="/handle-user-form.php" method="post">
        <button type="submit">submit</button>
    </form>
```

Note: There are two ways to add a button to a form:

<button>

<input type=”button”>

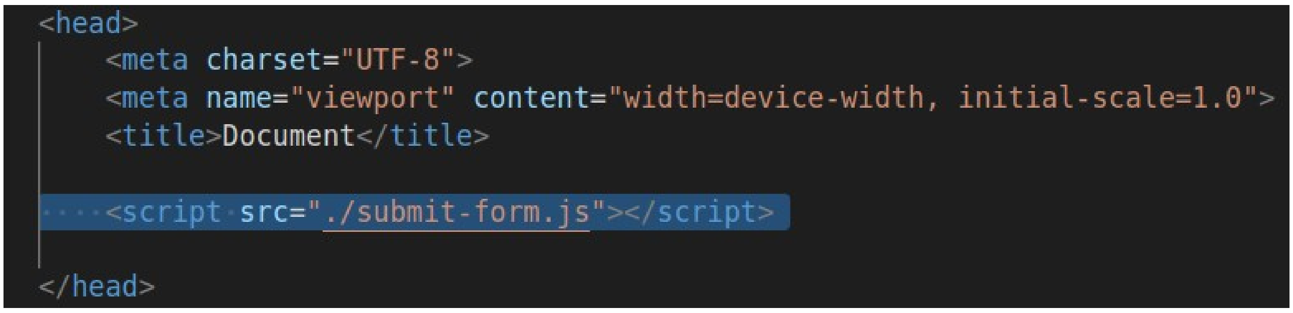
Both options are valid, but the <button> option allows us to add full HTML inside the element. The input tag option only allows text inside the element.

JavaScript

JavaScript provides functionality for calling functions from an action in the webpage. Action triggers are called events.

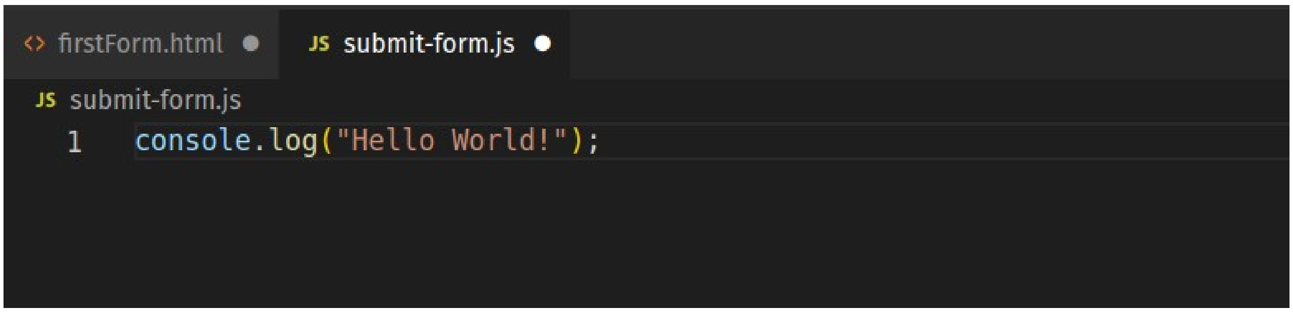
Create a new JS file in the project root directory with an appropriate name. Such as “submit-form.js”.

In the form page HTML file, link the JS file with the <script> tag in the <head>:



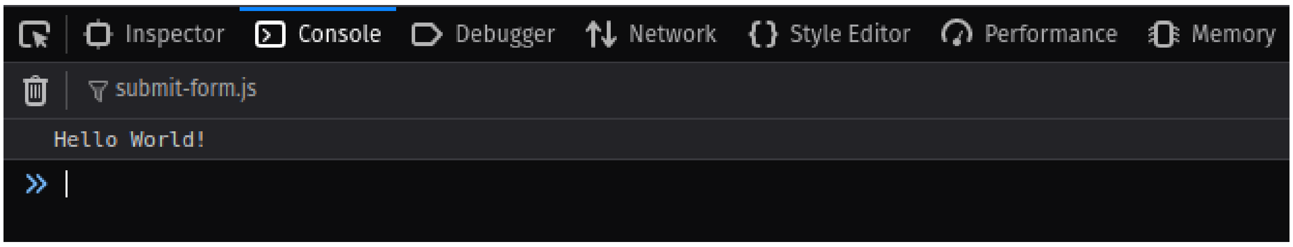
```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <script src="./submit-form.js"></script>

</head>
```

To test the JS file is linked correctly, add a Hello World! console log to the JS file:



```javascript
console.log("Hello World!");
```

Open your web browser development tools (Chrome, Firefox: F12) to access the console. The console should contain a log entry with the Hello World! Text:



```
Hello World!
>>
```

If the console log was successful, the JS file was linked correctly.

Function & Event

Create a new function and name it “submitForm”.

```
console.log("Hello World!");

function submitForm() {

}
```

To use this function we need a way of linking the button in our form to it. One way of achieving this is to utilise the “click” event of the button.

Events are hooks that trigger when certain conditions are met. For example, the “click” event for a button triggers when the button is clicked by a user. We can tell the hook to interact with a function when the event happens.

We need to reference the form button to use the “click” event. This can be done multiple ways.

```
// traditional ID selector
document.getElementById("user-form-submit");
// or
// more capable CSS selector
document.querySelector("#user-form-submit");
```

getElementById is very specific in its search scope. It only accepts IDs which must be unique in the DOM. As such it is optimised to perform its limited scope task quickly.

querySelector will accept any CSS selector (h1, .myClass, #myID, etc) and is therefore very flexible and powerful in its search scope declaration. It is not as fast as task-specific methods, but allows for diverse functionality from one method.

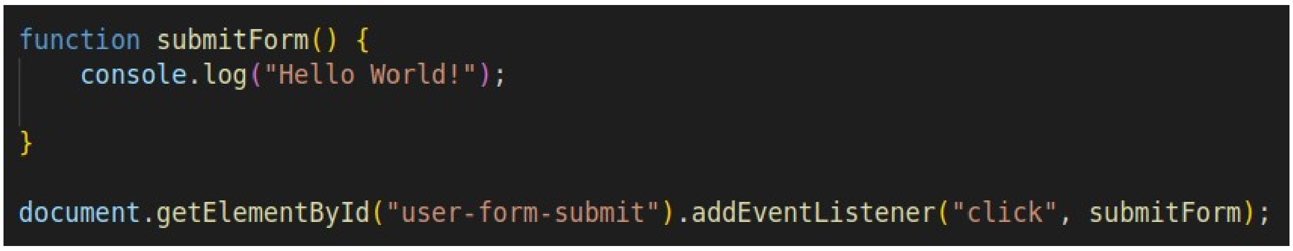
getElementById makes more sense for achieving our task. Add an event listener to the form button, and link the “click” event to the “submitForm” function:

```
document.getElementById("user-form-submit").addEventListener("click", submitForm);
```

Note: this line of code is written outside of the function. If the line was added inside the function, the function would have to run to create the event listener, which we want to happen upon the event triggering.
The event would never trigger, and thus the function would never run.

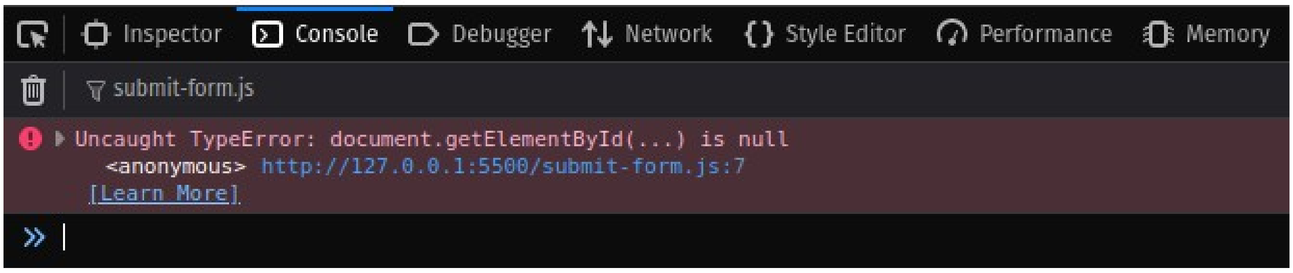
Test the event listener

Move the Hello World! Console log line into the “submitForm” function:



```
function submitForm() {
    console.log("Hello World!");

}

document.getElementById("user-form-submit").addEventListener("click", submitForm);
```

Save all files and open the web browser developer tools console:

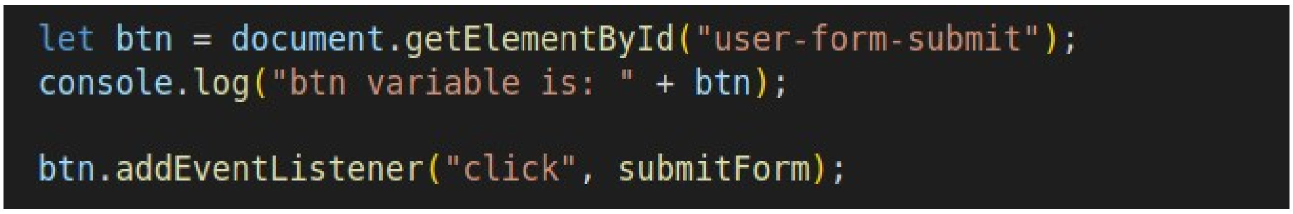


```
Uncaught TypeError: document.getElementById(...) is null
    <anonymous> http://127.0.0.1:5500/submit-form.js:7
   [Learn More]
```

An error has occured with the JavaScript code. Uncaught TypeError indicates an operation we attempted was not possible because the variable was not the correct data type. This is a common error is JavaScript due to its dynamic typing.

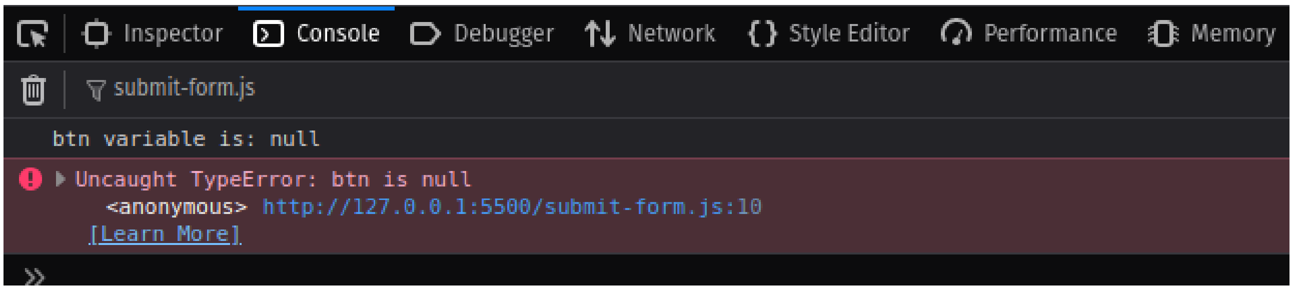
In this case it is caused by the getElementById returning null instead of the button element.

Breaking down our code can help identify the issue:



```
let btn = document.getElementById("user-form-submit");
console.log("btn variable is: " + btn);

btn.addEventListener("click", submitForm);
```

Instead of a one-line solution, this creates a variable called “btn” for the output of getElementById. We can then print this variable to the console to see what it contains:



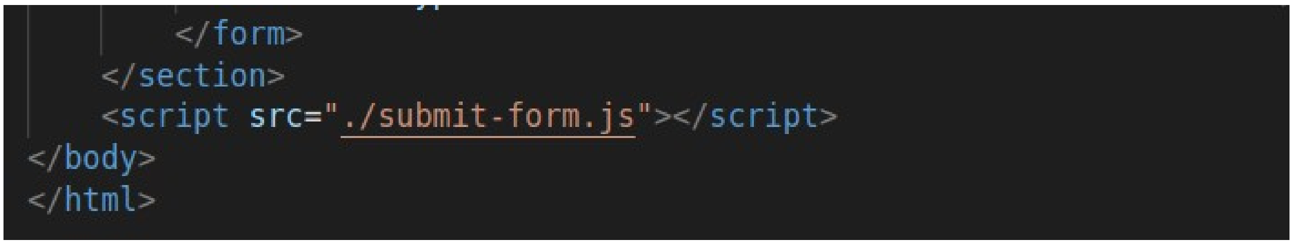
```
btn variable is: null
Uncaught TypeError: btn is null
    <anonymous> http://127.0.0.1:5500/submit-form.js:10
   [Learn More]
```

Btn is null, which means the getElementById failed to retrieve the button element...

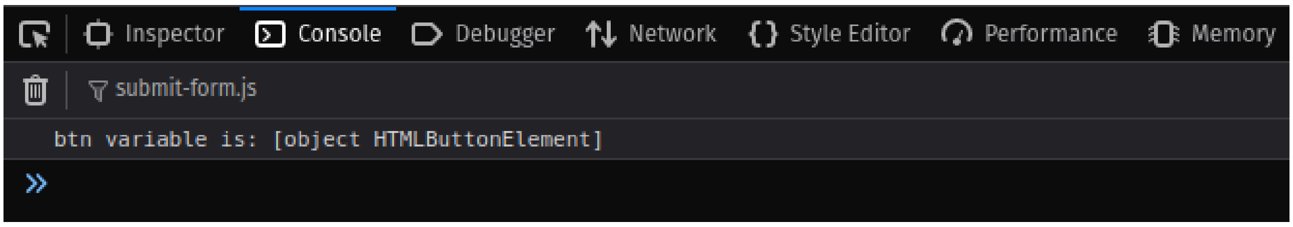
Script initialisation order

Our JS file is linked to HTML in the <head>. HTML files are interpreted by web browsers from top to bottom, so our JS code is being run before the web browser has created the body of the webpage.
The solution to our problem is to move our JS file link below the form.

Move the <script> element to the end of the <body> element, and save the file:

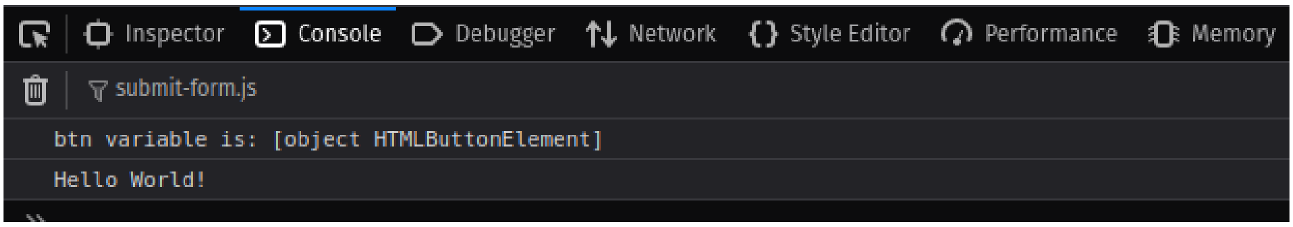


```
            </form>
        </section>
        <script src="./submit-form.js"></script>
    </body>
</html>
```



```
btn variable is: [object HTMLButtonElement]
```

And when the form submit button is clicked, the submitForm function outputs the Hello World! Message in the console:



```
btn variable is: [object HTMLButtonElement]
Hello World!
```

Form data

Remove the Hello World! console log from the submitForm function. Create a variable called "form" and set it equal to the form HTML element:

```
function submitForm() {
    let form = document.getElementById("user-form");
```

We will utilise the HTML element's data structure to access the two input fields by ID. Instead of using getElementById, we will simply access them as attributes by ID:

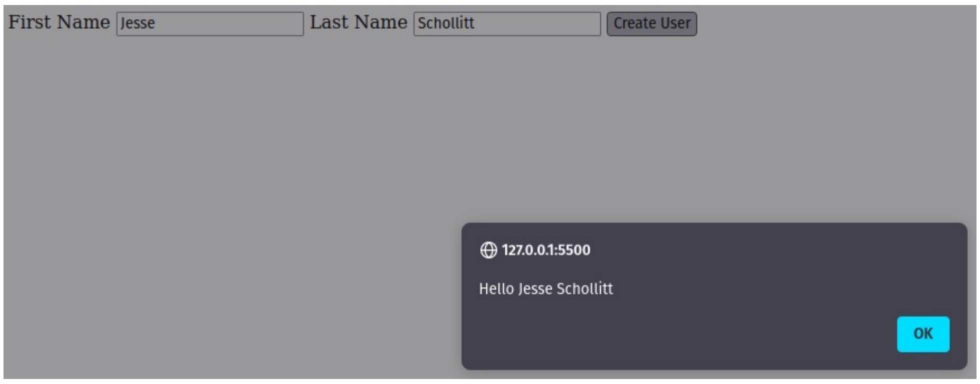
```
function submitForm() {
    let form = document.getElementById("user-form");

    let fname = form["first-name"].value;
    let lname = form["last-name"].value;
```

Note: Add ".value" to the end of an input element to access the current inputted value of the field. For text inputs, this will be the text entered by the user.

Test if this is working correctly with an alert:

```
function submitForm() {
    let form = document.getElementById("user-form");

    let fname = form["first-name"].value;
    let lname = form["last-name"].value;

    alert("Hello " + fname + " " + lname);
```

Populate the form fields and submit the form. An alert message should pop up with the supplied values:


First Name Jesse Last Name Schollitt Create User

127.0.0.1:5500

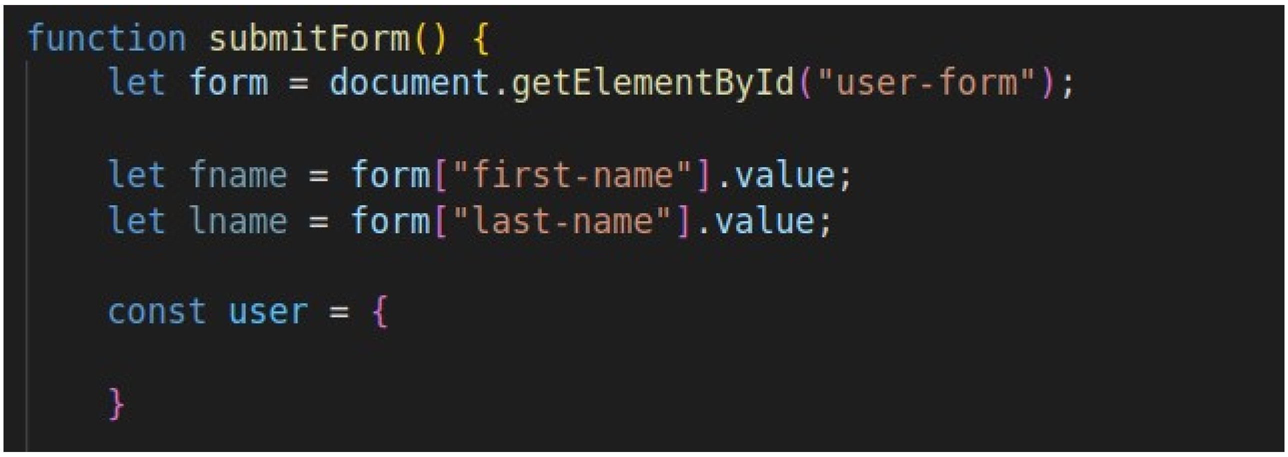
Hello Jesse Schollitt

OK

Data objects

Associated data can be stored in an object to organise and provide structure. The form data is a good candidate for storing in an object.

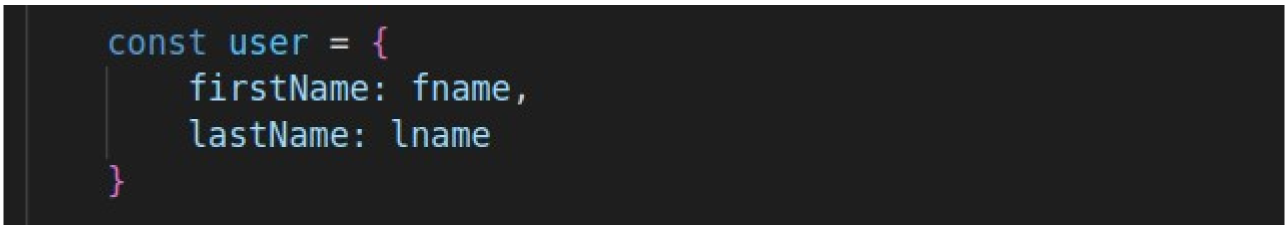
Create an object in the submitForm function and call it “user”:



```
function submitForm() {
    let form = document.getElementById("user-form");

    let fname = form["first-name"].value;
    let lname = form["last-name"].value;

    const user = {

    }
```

Note: the alert line was removed as it is no longer needed.

Inside the object we can create properties and assign values with the following syntax:

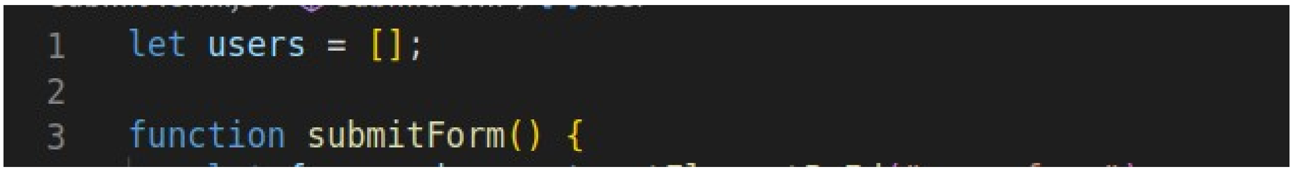


```
    const user = {
        firstName: fname,
        lastName: lname
    }
```

The name on the left is the property name of the user object, and the name on the right is the variables we created outside the object. We are assigning the value from the form data to the property of the user object.

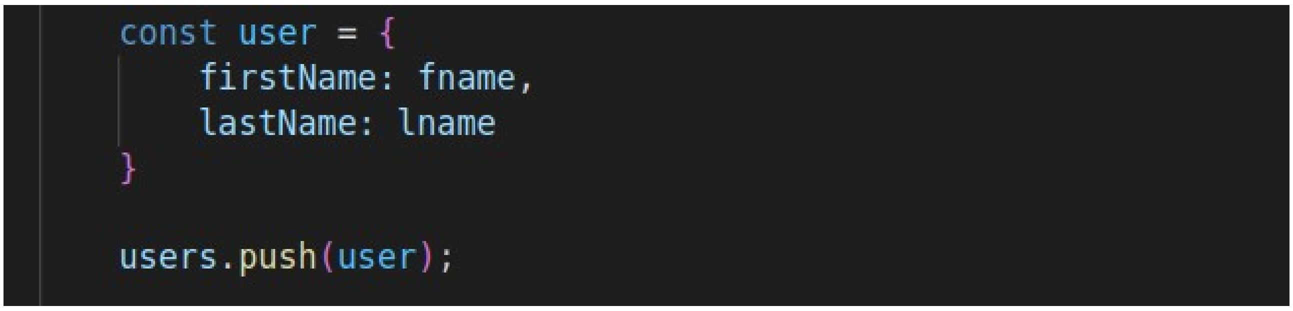
Now we have an object, we should do something with it.

Create an empty array at the top of the JS file, called users:



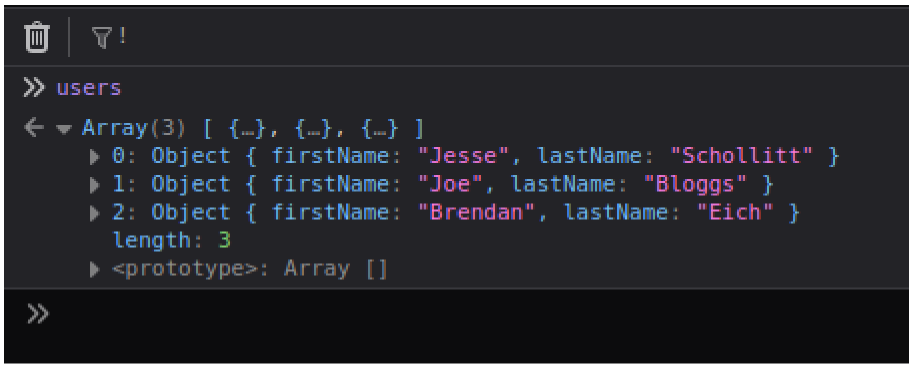
```
let users = [];

function submitForm() {
```

Inside the submitForm function and below the user object, add the user object to the array using array.push(data):



```
const user = {
    firstName: fname,
    lastName: lname
}

users.push(user);
```

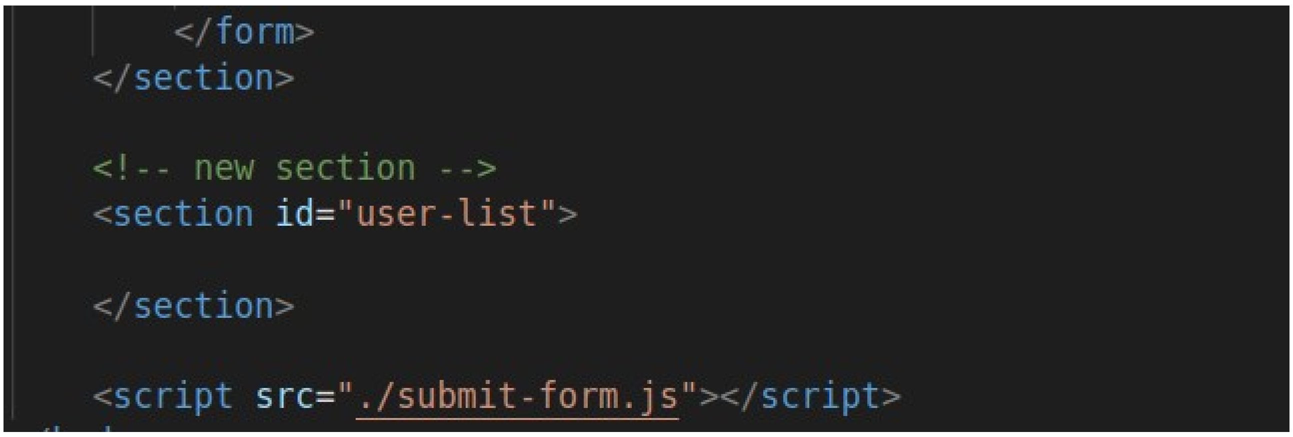
In your web browser, enter and submit a few users into the form. Open the console and type the name of the user array “users”, and press enter:



```
>> users
<- ▾ Array(3) [ {…}, {…}, {…} ]
    ▸ 0: Object { firstName: "Jesse", lastName: "Schollitt" }
    ▸ 1: Object { firstName: "Joe", lastName: "Bloggs" }
    ▸ 2: Object { firstName: "Brendan", lastName: "Eich" }
      length: 3
    ▸ <prototype>: Array []
>>
```

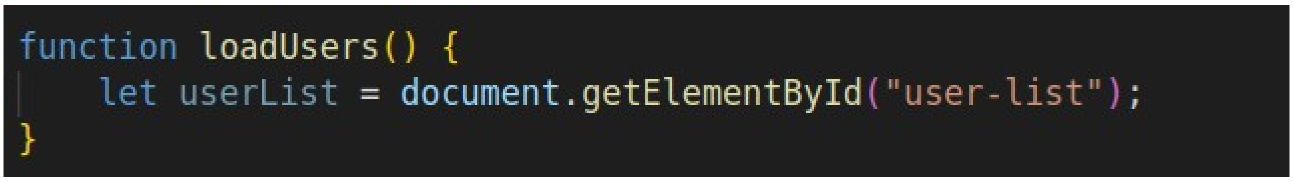
Object to HTML element

In the HTML file, create a new section below the form section and set its ID to “user-list”:



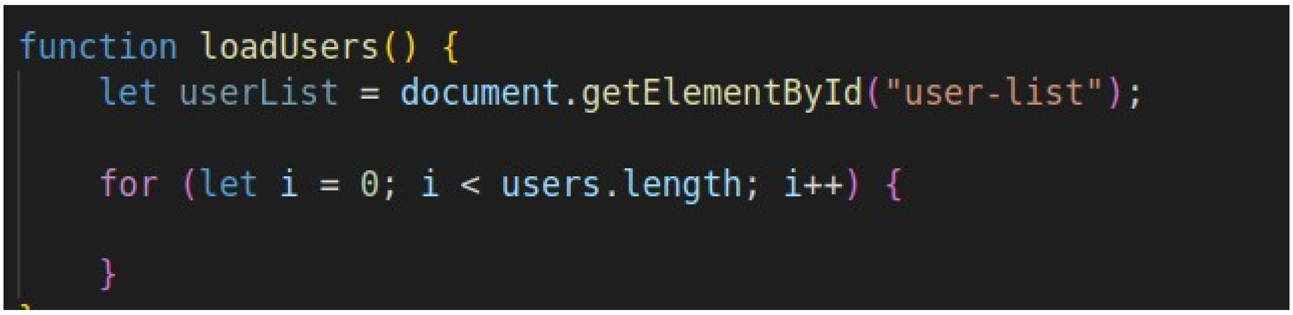
```
        </form>
    </section>

    <!-- new section -->
    <section id="user-list">

    </section>

    <script src="./submit-form.js"></script>
```

In the JS file, create a new function underneath submitForm() called “loadUsers”. Inside the function create a variable that refers to the new section in the HTML file:



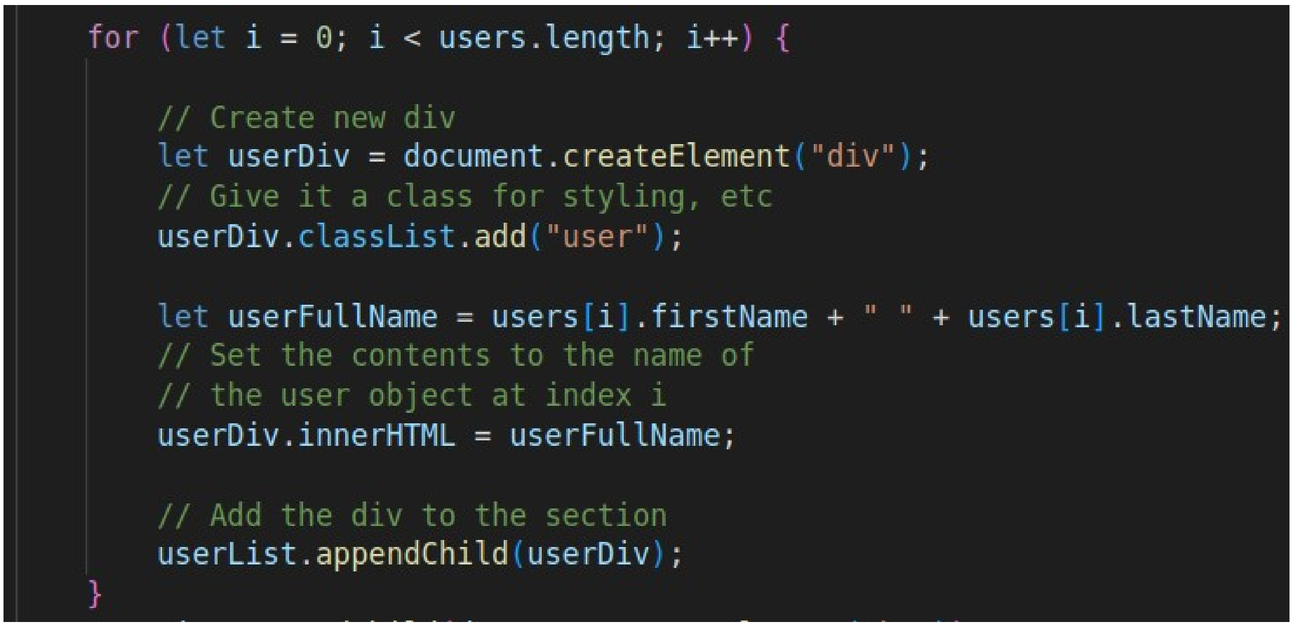
```
function loadUsers() {
    let userList = document.getElementById("user-list");
}
```

Add a for loop to the function that loops over the objects in the users array:



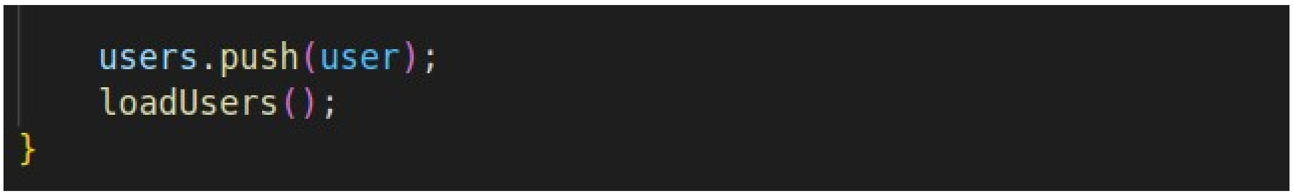
```
function loadUsers() {
    let userList = document.getElementById("user-list");

    for (let i = 0; i < users.length; i++) {

    }
```

In the for loop we are going to create a new div element, populate it with the name of a user, and add it to the section:



```
    for (let i = 0; i < users.length; i++) {

        // Create new div
        let userDiv = document.createElement("div");
        // Give it a class for styling, etc
        userDiv.classList.add("user");

        let userFullName = users[i].firstName + " " + users[i].lastName;
        // Set the contents to the name of
        // the user object at index i
        userDiv.innerHTML = userFullName;

        // Add the div to the section
        userList.appendChild(userDiv);
    }
```

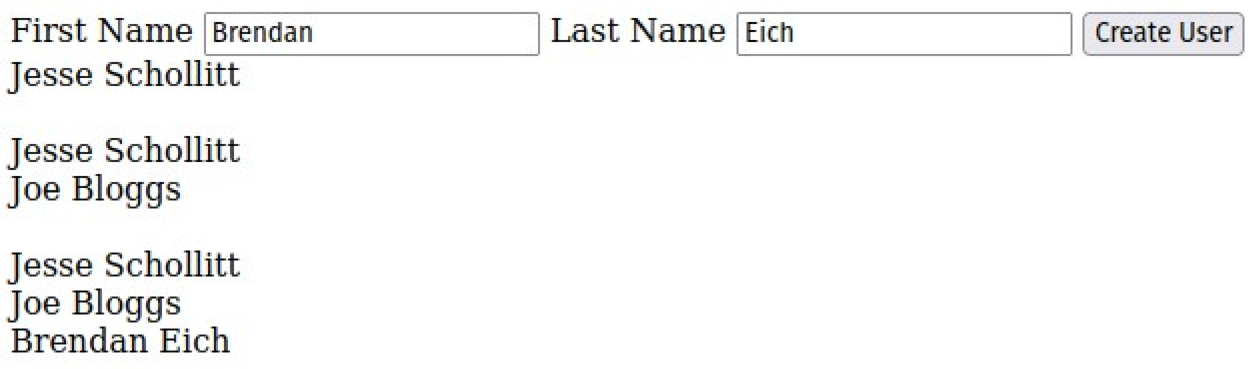
At the end of the submitForm function, add a call to the new function:



```
    users.push(user);
    loadUsers();
}
```

Test

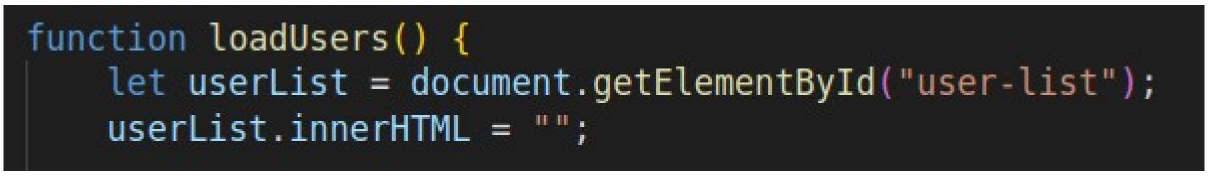
Populate and submit a few more users with the form. You'll notice each time a new user is added, all of the existing users get added again:



First Name Brendan Last Name Eich

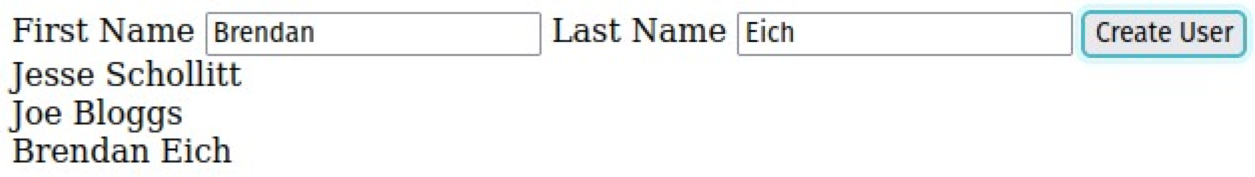
```
Jesse Schollitt

Jesse Schollitt
Joe Bloggs

Jesse Schollitt
Joe Bloggs
Brendan Eich
```

To fix this we can clear the contents of the section each time it loads. This can be achieved by looping over the child elements of the section and deleting them, or we can simply set the contents of the section to nothing, like so:



```
function loadUsers() {
    let userList = document.getElementById("user-list");
    userList.innerHTML = "";
```

Which results in each user's name only appearing once on the webpage:



First Name Brendan Last Name Eich

```
Jesse Schollitt
Joe Bloggs
Brendan Eich
```

Challenge

There are more fields of data expected by the application (see beginning of this document).

Complete the exercise:
1. Add the missing fields to the form
2. Collect and add the missing field data to the user objects
3. Add the missing data to the webpage for each user

Wrapping up

Save your work, commit and push your changes to your git repository.

Exercise complete.