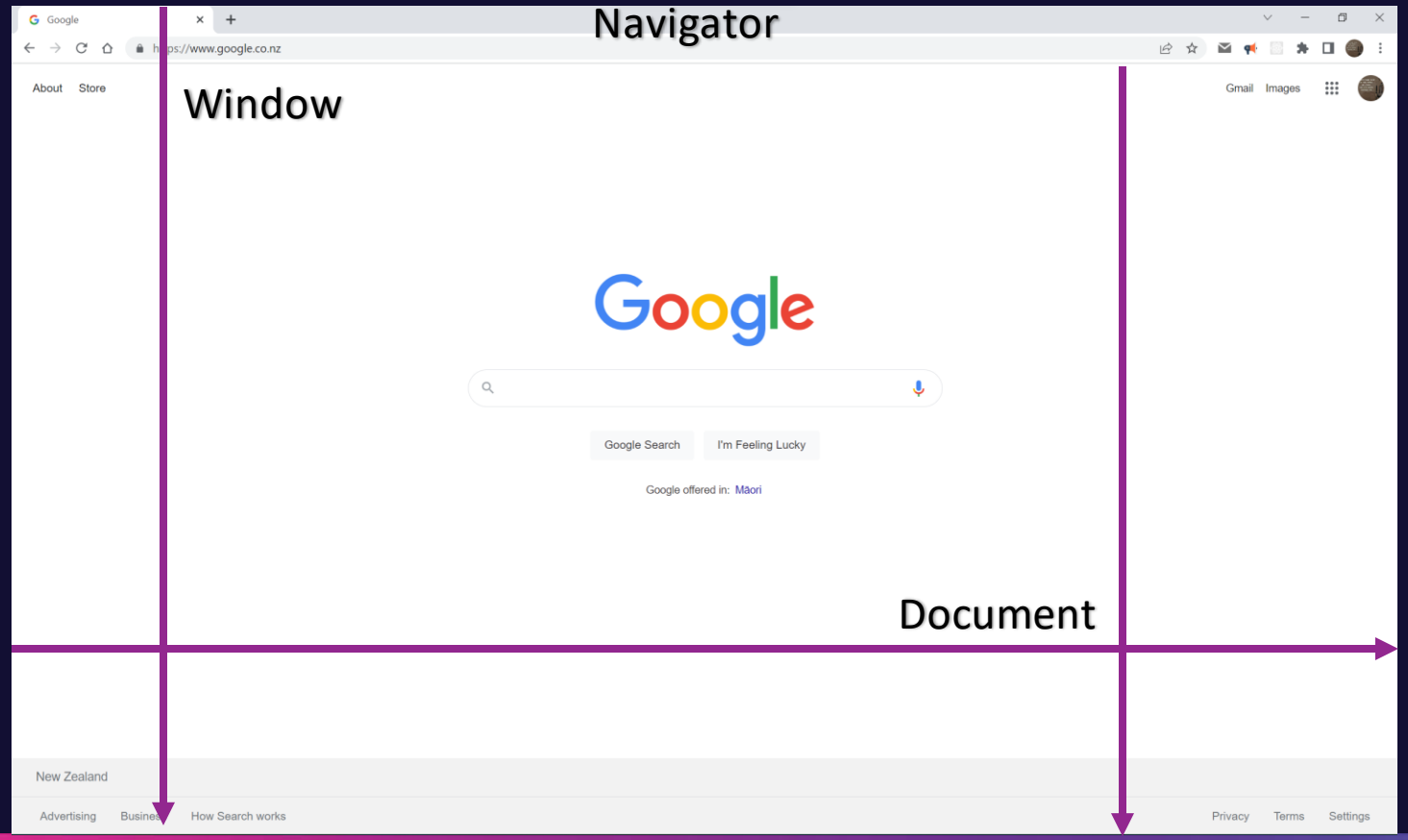
Modifying the DOM

Reuben Simpson

What is the DOM?

- The DOM is the Document *Object Model*. But what does that mean?
- Let's take a look at the structure of a page on a browser

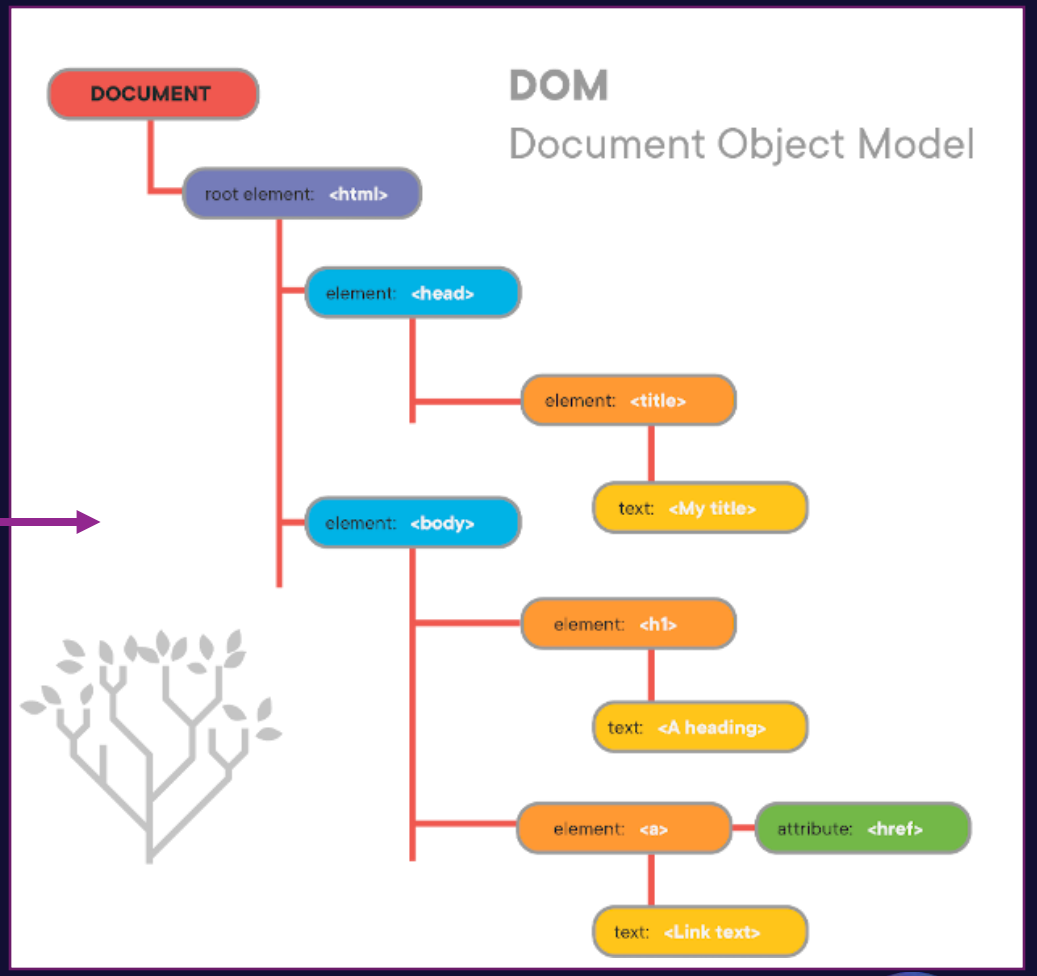


- This image shows us the different parts of the browser.
- The Window object is the browser tab that the web page is loaded into. It is the top-level object.
- The Navigator object represents the identity of the browser e.g. the user's geolocation and device type
- The Document object is the actual page that is loaded in the window

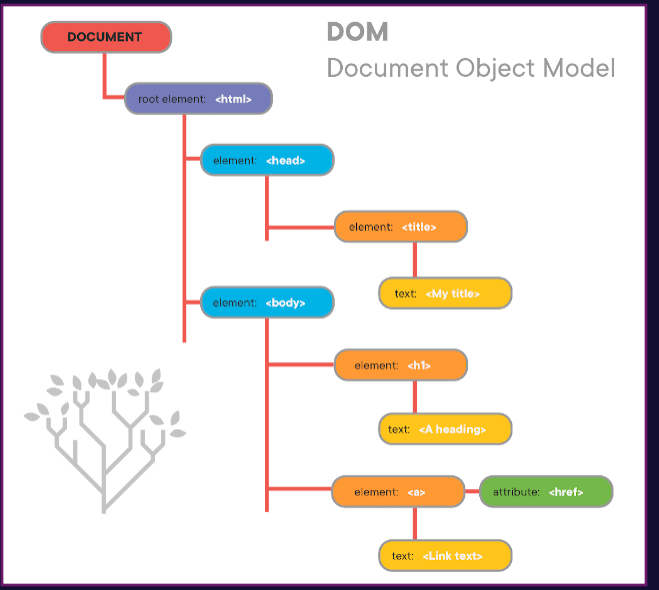
So… what is the DOM?

- The Document *Object Model* is a programming interface (API) that allows us to create, change, or remove elements from the document. We can also add events to these elements to make our page more dynamic.
- The DOM views an HTML document as a tree of nodes. A node represents an HTML element. When a web page is loaded, the browser creates a Document Object Model of the page.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My title</title>
  </head>
  <body>
    <h1>A heading</h1>
    <a href="#">Link text</a>
  </body>
</html>
```



Why do we care about the DOM?

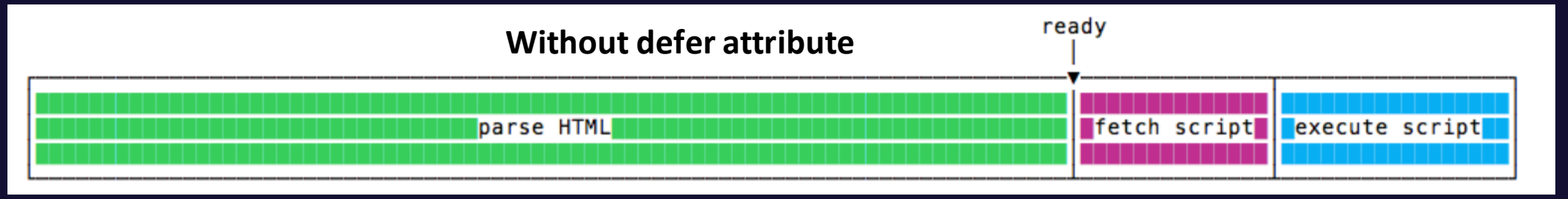


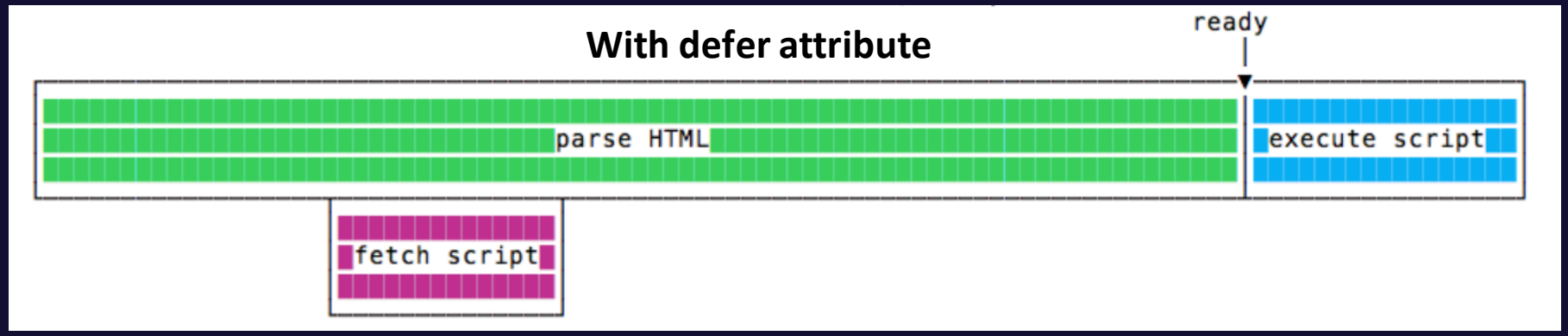
- Our HTML and CSS is rendered and shown on the DOM.
- This means that if anything on our page needs to change, this needs to be done through the DOM.
- Up until now, we have made all changes to our pages by changing the code in our IDE (VS Code). This is fine for simple static web pages, but for anything more complex than that, we need something to help us change the content in the DOM by interacting with the browser (without going back to the code).
- For example; if we wanted a button that upon being clicked, changes the colour of the page background, how would we do that right now?
- We can do this by *manipulating* the DOM with JavaScript.

Modifying the DOM

- Let's go through how to start changing the elements and properties in the DOM with JavaScript.
- 1) Let's create a new HTML and external JS file in a new folder.
- 2) This time, put your script tag in the <head>

Defer attribute





- Scripts with defer never block the page.
- Scripts with defer always execute when the DOM is ready
- We should add defer to all our script tags going forward

Accessing the body property

- To get access to our HTML's body we first need to get the document and then use .body to access the body property `document.body`
- Within the body there is a property called innerHTML. We can use this property to add some HTML to our index.html file with JavaScript. Add this script to your JS file `document.body.innerHTML = "<h1>Hello world</h1>";`
- We have access to the style property to add our own styles to the body. Try adding `document.body.style.background = "blue";` to your JS file.
- The style property allows us to access and change any valid CSS to an element on our page

Changing the page colour with JavaScript

- Let's add a button to our HTML page that allows us to change the background colour of our page. Note: Comment out the code from the previous slide.

1. Go into your html file and add a <button>Click Me</button> element in the body.

- In our JS file, we want to create a function that will allow us to change the colour of the page every time it is clicked.
- For this example, our function will change the colour of the body background from white to blue or blue to white.

Changing the page colour with JavaScript

2. In our JS file we can create the following function

```
function changeColour() {
    document.body.style.background = "blue";
}
```

- This will let us change our background from the default (white) to blue but won't allow us to change it back.

3. Let's add a conditional statement that first checks if the background is blue, if it is, then we want to change it to white, but if it isn't, then we can change it to blue

```
function changeColour() {
    if (document.body.style.background === "blue") {
        document.body.style.background = "white";
    } else {
        document.body.style.background = "blue";
    }
}
```

Changing the page colour with javascript

- Now that our function is written all we need to do is to tell our button to run the function.
- In HTML, buttons have an attribute called “onclick”, this attribute allows us to do something when we click on the button.
- This is how we could use it

```
<button onclick="changeColour()">Click Me</button>
```

- We assign the function call to our onclick value. This lets us run the function on each click of the button.
- Try it out on your page…. Success!

Bonus: changing to multiple colours

- Let’s go through how we could go about changing the background to more than just 2 colours.
- First, we need to breakdown the logic of what we want to accomplish:
  - Click button
  - Colour changes to colour1
  - Click again
  - Colour changes to colour2
  - Repeat for some number of colours
- We know that we need some list of possible colours we want to change our background to, we can use an array for that
- We also know that we need to keep track of which colour we are on right now
- Let’s implement this

Bonus: changing to multiple colours

- Let's create an array of colours

```
function changeColour() {
    const colourArray = ['white', 'blue', 'red', 'green', 'yellow', 'orange'];
}
```

- Now we need some variable to keep track of the current colour. We want this variable outside of our function because we don't want it to reset every time the function is run.

```
let currentColour = 0;

function changeColour() {
    const  colourArray = ['white', 'blue', 'red', 'green', 'yellow', 'orange'];
}
```

- We want to start with the first colour which is at position 0;

Bonus: changing to multiple colours

- Now we can start writing the logic to change the colour:

```
currentColour++;
document.body.style.background = colourArray[currentColour];
```

- Each time we run our function we move our currentColour to point at the next colour, then we change our background colour to that colour.
- Putting it all together, we have:

```
let currentColour = 0;

function changeColour() {
    let colourArray = ['white', 'blue', 'red', 'green', 'yellow', 'orange'];

    currentColour++;
    document.body.style.background = colourArray[currentColour];
}
```

Bonus: changing to multiple colours

- If we want the colour to reset back to the beginning when it gets to the end we need to add one more check.

```
function changeColour() {
    let colourArray = ['white', 'blue', 'red', 'green', 'yellow', 'orange'];

    if (currentColour === 5) {
        currentColour = 0;
    } else {
        currentColour++;
    }
    document.body.style.background = colourArray[currentColour];
}
```

- We added an if statement to check if the current colour is on the last colour and if it is then we reset it back to 0.

Accessing other elements

- Through JS we don’t just have access to the body property.
- We can also get access to any HTML element on the page through several ways.

```
document.querySelector(); //gets an element by its CSS selector but only returns the first matching element
document.querySelectorAll(); //gets all elements that match the given selector
document.getElementById(); //gets an element by its id but only returns the first matching element
document.getElementsByClassName(); //gets all elements with a certain class name
document.getElementsByTagName(); //gets all elements with a specified tag name
```

Some useful properties

- textContent (changes the text inside of an element)
- innerHTML (changes HTML inside of an element)
- style (changes the styles on an element)

Using getElementById

- To use the getElementById method we need to add an id to one of our HTML elements. Create an h1 element and give it an id:

```
<h1 id="helloWorldHeader">Playing with the DOM</h1>
```

- Go back to your JS file
- We can use document.getElementById() to get and store our html element.

```
const helloWorldHeader = document.getElementById('helloWorldHeader');
```

- Then we can modify the element

```
helloWorldHeader.style.fontSize = '10px';
```

- We can even change its content with JavaScript

```
helloWorldHeader.textContent = "Look! My text content has changed.";
```

Using querySelector

- We can also select html elements with the document.querySelector() method.
- This method takes any valid selector that we can also use in our CSS files.
- For the last example we can do

```
const helloWorldHeader = document.querySelector('h1');
```

- Or we can also do

```
const helloWorldHeader = document.querySelector('#helloWorldHeader');
```

- Then we can apply the same styles and change the content as before

```
helloWorldHeader.style.fontSize = '10px';
helloWorldHeader.textContent = 'Look! My text content has changed.';
```

Exercise 1

- Implement a counter on the page with a button that adds 1 to the count

Events

- In JavaScript there is a concept called an “Event”.
- An Event is something that happens that triggers some response, for example “onclick” is an event. When the element it is applied to is clicked, it will trigger some action and some response.
- Let’s look at some other events:

```
onchange; //when a change is detected to an element
onmouseover; //when the mouse is moved over the object
onmouseout; //when the mouse is moved out of the object
onkeydown; //when the user pushes a key on the keyboard
onload; //when the browser has finished loading the page
oninput; //each time a new character is added or taken away in an input field
onsubmit; //specific to form elements, handles the submission of a form
```

Events continued…

- These events can be applied directly to our HTML elements in our html file as we've seen with the onclick event.

```
<button onclick="changeColour()">Click Me</button>
```

- But we can also add these events directly from our JS file.
- We can do this with an EventListener.
- An event listener does exactly what it sounds like, it waits for some specified event on an html element and then executes some function we define.

The Preferred Method: Event Listeners

- Using the `.addEventListener()` method in our JS code is the preferred approach over adding an onevent inline onto a html element.
- To add an event listener to one of our elements we need to use the `addEventListener()` method. This method takes 2 main parameters:

1. The first parameter is the event it will listen for e.g. click, mouseover, mouseout (notice that with when naming an event in JS we don't need the "on" prefix like in html for onclick or onmouseover)
2. The second parameter is the function we want to run when the event is triggered, this can be defined within the parameter itself or externally.

Using Event Listeners continued…

Example of using addEventListener()

```
const myDiv = document.getElementById(divId');
myDiv.addEventListener('mouseover', function(){ myDiv.style.background = 'blue')};
```

1. We select an element with either getElementById or querySelector and save that to a variable.
2. Now we can add an event listener to that variable and allow it to do something on a certain event.
3. Here the event is "*mouseover*" which is when the mouse hovers over the element
4. The function we pass in to the second parameter is changing the background colour to blue.

Using Event Listeners continued…

5. We can then add a mouseout event that will change the background back to white when the mouse leaves the element.

```
myDiv.addEventListener('mouseout', function() { someElement.style.background = 'pink')};
```

- Alternatively, we can create an external function like this:

```
function changeToPink() {
    myDiv.style.background = 'pink'
}

myDiv.addEventListener('mouseout', changeToPink);
```

- The addEventListener's function calls on the changeToPink function when the event is triggered, which changes the background to pink.

Event Listeners with functions

```
function changeToPink() {
    myDiv.style.background = pink'
}

myDiv.addEventListener('mouseout', changeToPink);
```

- This external function we created works, but it only affects this one specific element *(myDiv).* We aren't able to call this function from another event listener for a different element … so it isn't reusable.
- But can we change that?
- Yes, we can.

Target property

- The target property relates to the element that was at the source of the event, in this case the **myDiv** that we applied the event listener to.
- By accessing this target property, we can change the styles or content of the element.

```
function changeToPink(event) {
    event.target.style.background = pink';
}

myDiv.addEventListener('mouseout', changeToPink);
```

- So now we have a dynamic or *reusable* function. The element it manipulates is the target element that the event occurred on.

Getting text from input fields

- We know that the event input or oninput will be executed whenever a new character is added to the input. Let's use that to help us log the value of our input to the console.
- First add an input element to your html

```
<label>Some input: </label><input type="text" />
```

- Then we can either add an oninput to this element, or add an event listener from our JS to the input event.

Inline function

```
<input type="text" oninput="logValue(event)" />
```

```
function logValue(e) {
    console.log(e.target.value);
}
```

Event Listener

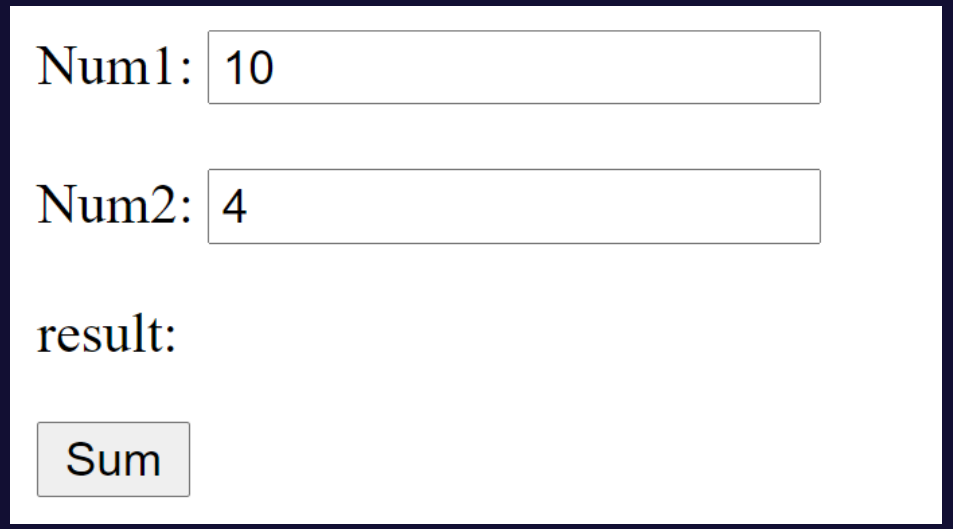
```
const inputElement = document.querySelector('input');
inputElement.addEventListener('input', logValue);
function logValue(e) {
    console.log(e.target.value);
}
```

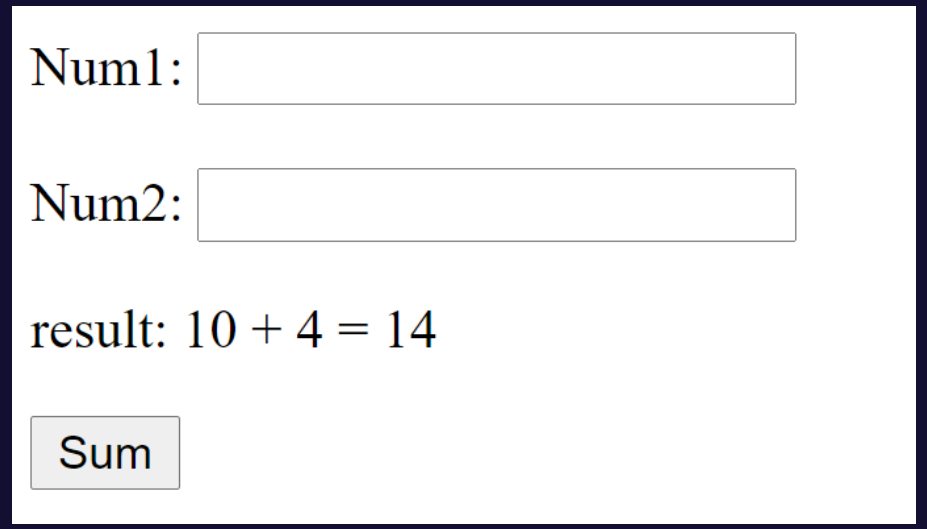
Exercise 2

- Add a character count to your *input element* that reflects the number of characters currently in the input field.

Exercise 3

- Add two input fields that accepts only numbers. Add a button that when pressed, will add the two field's number values together and then render (or display) the result on the page





MISSION READY

www.missionreadyhq.com

DARE TO DEVELOP

Thank you

Reuben Simpson