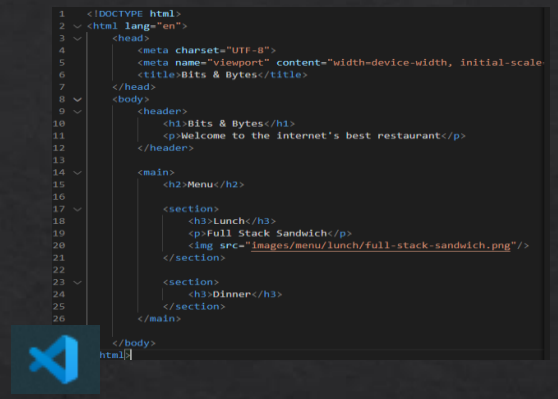
# Module 3
## DOM




### Session Objectives:

- Describe the difference between the DOM and HTML
- Select elements from the DOM using getElementById, querySelector, querySelectorAll
- Describe the DOM and how it is structured (tree)
- Set innerText on HTML elements
- Describe why innerHTML can be dangerous
- Create new DOM elements using createElement() and insertAdjacentElement()
- Traverse the DOM
- Investigate the living DOM in the browser

# Module 3
## DOM

### The DOM

DOM stands for the Document Object Model. It's the browser's internal representation of the structure of the current web page.

The DOM is an internal data structure that browsers use to represent the structure and content of a web page. When the browser loads an HTML document, it needs to translate that into something that it can use to draw a graphical representation of the page.

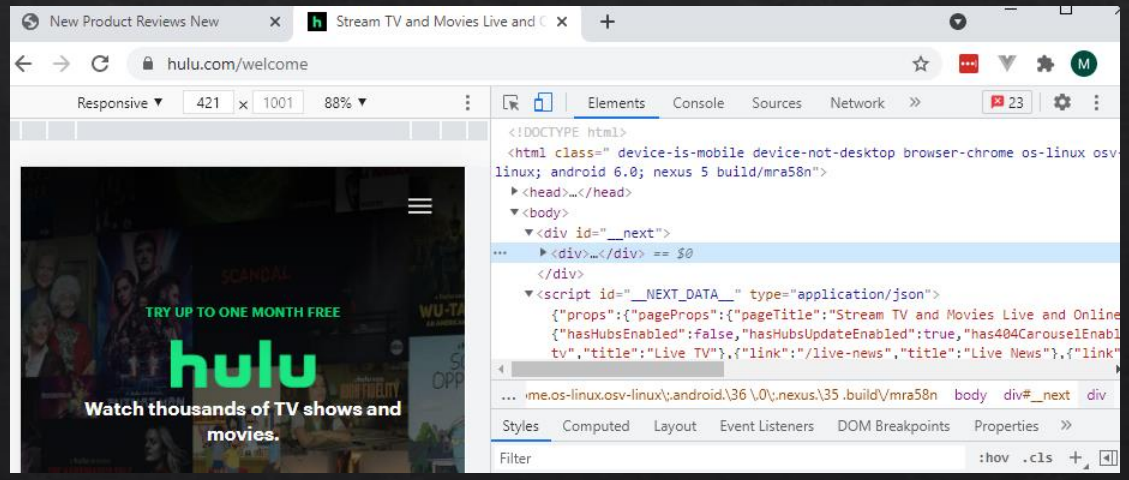
Understanding the DOM is important. HTML is static. It's only read once when the page loads, and then it's converted into a DOM. CSS and JavaScript perform their tasks using the DOM, not the static HTML.

# Module 3
## DOM

### Checking the DOM

To check the DOM, open the developer tools in Firefox or Chrome and go to the Inspector (Firefox) or the Elements (Chrome) tab. This shows an HTML-like view of the DOM as it is at that moment. If you change any JavaScript or CSS in your code, you'll see those changes happen immediately in that view.

This is the best way to understand how your browser interprets the HTML source code and how it changes due to live user interaction.



# Module 3
## DOM

### Selecting DOM elements

JavaScript has many built-in functions that you can use to manipulate the DOM. The following is a list of the most commonly used functions.

**getElementById()**
The first function you'll learn about is getElementById(). This function gets a single HTML element from the DOM and returns a reference to it.

```
<h1 id="title">Welcome</h1>
```

```
let titleElement = document.getElementById('title');
```

# Module 3
## DOM

### Selecting DOM elements

**querySelector()**
Is for selecting single elements that don't have an ID.
querySelector() takes a standard CSS selector and returns the first element it finds that matches that selector.

```
<ul id="todo-list">
    <li class="todo">Walk the dog</li>
    <li class="todo">Mow the lawn</li>
    <li class="todo">Go shopping</li>
</ul>
```

```
let firstListItem = document.querySelector('.todo');
```

# Module 3
## DOM

### Selecting DOM elements

**querySelectorAll()**
If you want to get all of the list items, you can use querySelectorAll() instead. This returns a NodeList of all the elements, which you can use as an array:

```
<ul id="todo-list">
    <li class="todo">Walk the dog</li>
    <li class="todo">Mow the lawn</li>
    <li class="todo">Go shopping</li>
</ul>
```

```
let firstListItem = document.querySelector('.todo');
```

# Module 3
## DOM

### Changing DOM elements

**Once you've selected an element, you can change it by accessing its properties.**

Changing text with **innerText**

You can change the text inside of an element using the element's innerText property:

```
// Get the list
let todoList = document.getElementById('todo-list');

// User will see '<li>Update documentation</li>',
// and will not create a new line item
todoList.innerText = '<li>Update documentation</li>';
```

Be cautious of what elements you use innerText on. Because you can use it on any element—even ones that don't normally have their own text such as ul—it'll overwrite child elements if there are any.

# Module 3
## DOM

### Changing DOM elements

**Once you've selected an element, you can change it by accessing its properties.**

Changing HTML with **innerHTML**

innerHTML acts like innerText, but it does render HTML added as DOM elements:

```
// Get the list
let todoList = document.getElementById('todo-list');

// User will see 'Update documentation'
// within a newly created list item
todoList.innerHTML = '<li>Update documentation</li>’;
```

Danger: Be careful when using innerHTML

# Module 3
## DOM

### Creating DOM elements

**createElement()**

You can create DOM elements directly using :

```
let newListItem = document.createElement('li');
newListItem.innerText = 'Update documentation’;
```

createElement() creates a new DOM element and returns it. You can then call all the normal functions on that element that you can call on any DOM element, like innerText.

This element isn't on the page yet, so it won't show up to the user until you insert it into the living DOM.

# Module 3
## DOM

### Creating DOM elements

#### insertAdjacentElement()

You add new elements to the DOM with the insertAdjacentElement() method. You call the method on another element that exists in the DOM, either a sibling element or parent element. You pass it two parameters: the location and the element to add.

The location parameter is a string that represents the location you want to insert the new element relative to the element you call insertAdjacentElement on. The string can be one of four possible values:

| Location | Meaning |
|---|---|
| `'beforebegin'` | Insert the element before this one |
| `'afterbegin'` | Insert the element inside this one as the first child |
| `'beforeend'` | Insert the element inside this one as the last child |
| `'afterend'` | Insert the element after this one |

# Module 3
## DOM

### Creating DOM elements - insertAdjacentElement()

**beforebegin**
You can add an element immediately before another one with the beforebegin:values:

```
let todoList = document.getElementById('todo-list');

let newHeader = document.createElement('h2');
newHeader.innerText = 'Todo List';

// Place the header before the list
todoList.insertAdjacentElement('beforebegin', newHeader);
```

# Module 3
## DOM

### Creating DOM elements - insertAdjacentElement()

**afterbegin**
You can add new DOM nodes to the beginning of your list with afterbegin:

```
let newListItem = document.createElement('li');
newListItem.innerText = 'Update documentation';

let todoList = document.getElementById('todo-list');
todoList.insertAdjacentElement('afterbegin', newListItem);
```

This adds the element as the first child on the element with the ID of 'todo-list', right after the list begins

# Module 3
## DOM

### Creating DOM elements - insertAdjacentElement()

**beforeend**
To add the element to the end of your list, you can use beforeend:

```
let newListItem = document.createElement('li');
newListItem.innerText = 'Update documentation';

let todoList = document.getElementById('todo-list');
todoList.insertAdjacentElement('beforeend', newListItem);
```

This puts the element as the last child in the list, right before the list ends.

# Module 3
## DOM

### Creating DOM elements - insertAdjacentElement()

**afterend**
You can add an element immediately after another one with the afterend:

```
let todoList = document.getElementById('todo-list');

let newFooter = document.createElement('h2');
newFooter.innerText = 'End of List';

// Place the footer after the list
todoList.insertAdjacentElement('afterend', newFooter);
```

# Module 3
## DOM

### Traversing the DOM

There may be times when you don't know which selector to choose an element with, or you need to work with a specific child of an element and need to loop through or walk through a list of elements. For this, you can use some element properties to get an element's children or parent.

```
let todoList = document.getElementById('todo-list');
```

You can get all of its immediate children elements through the children property:

```
let todoItems = todoList.children;
```

children returns an HTMLCollection object, which you can turn into a real array with access to map, forEach, and all the other array functions with this:

```
let todoItemsArray = Array.from(todoList.children);
```

# Module 3
## DOM

### Traversing the DOM

There may be times when you don't know which selector to choose an element with, or you need to work with a specific child of an element and need to loop through or walk through a list of elements. For this, you can use some element properties to get an element's children or parent.

You can also get children by calling childNodes:

```
let todoNodes = todoList.childNodes;
```

This returns a NodeList object that contains all the nodes inside that element. You can also pass this to Array.from() to get a normal JavaScript array.

children returns elements that are children of this element. That means that it only contains other HTML elements and not the text that might be in the element.

childNodes returns nodes that are children of this element. That includes text (including whitespace) and comments that are in the DOM.

# Module 3
## DOM

**New Tools!**