Full Stack JavaScript

DOM

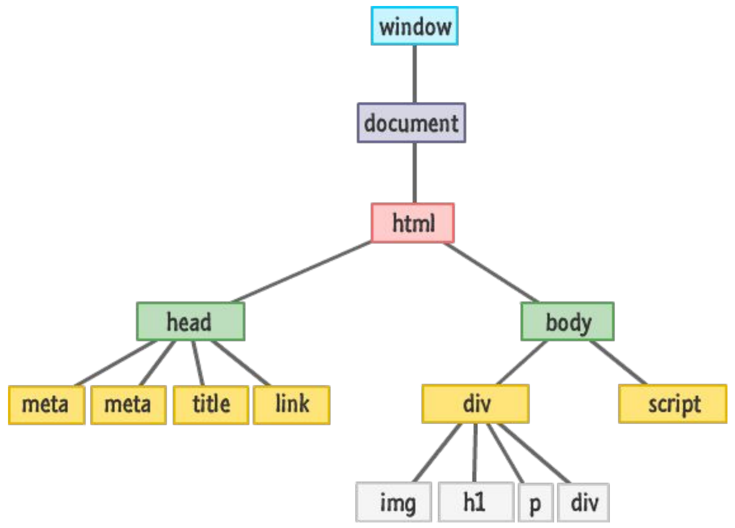
Download PDF Copy

DOM

- Selecting & Manipulating Elements
- Events
- Event Bubbling & Delegation
- Inputs & Forms

The Document Object Model (DOM) is an interface which allows dynamic access to the HTML.

The DOM is a logical tree structure where each element in HTML represents a node (object).



The DOM exposes methods that are used to manipulate the structure of a page.

Each web page loaded in the browser has a unique `document` object.

Open any web page in a browser, open the developer tools, and run this command in the console.

```
document.write("JavaScript is pretty cool.");
```

Accessing Elements

There are numerous ways to access various nodes within the DOM.

- Node.childNodes - all direct children of this node
- Node.parentNode - the parent of this node
- Node.firstChild - all direct children of this node

```
// body element
const bodyNode = document.body;

// html element
const htmlNode = document.body.parentNode;

// Array of all body's children
const childNodes = document.body.childNodes;
```

To access an element on the page using its ID...

- **document.getElementById()**
- returns a Node or null if none found
- WARNING: a common error is to capitalize “ID”.

```
let nav = document.getElementById("top-nav");
```

There are two other common methods that can be used to do the selecting which are:

- document.querySelector() - returns a Node or null
- document.querySelectorAll() - returns a NodeList
- CSS selectors are used as arguments to the methods.

```
document.querySelector("#intro"); // one with ID intro
document.querySelectorAll("p"); // all <p> tags
document.querySelectorAll(".row"); // all with class row
```

Manipulating Elements

The objects returned from these selectors contain a large amount of keys/values.

Remembering dot and bracket notation is going to be quite substantial.

JavaScript provides the power to change anything and everything about an HTML element.

```
const el = document.getElementById("myLink");
el.innerText = "Hello World";
el.classList.add("selected");
el.setAttribute("href", "/hello.html");
```

Manipulating: CSS Classes

Add, remove, toggle classes. And check whether the element has a certain class.

```
el.classList.add("some-class");
el.classList.remove("some-class");
el.classList.toggle("some-class");
if (el.classList.contains("some-class")) {
 ...  }
```

Manipulating: Attributes

Set attributes and read their values.

```
el.setAttribute("href", "http://example.com");
el.setAttribute("title", "I'm a tooltip!");
let imgSource = el.getAttribute("src");
```

Manipulating: Style Properties

Set individual inline styles.

NOTE: Names are camelCase and values are always strings.

```
el.style.width = "200px";
el.style.backgroundColor = "red";
el.style.boxShadow = "2px 2px 5px #ccc";
```

Manipulating: Text

Get and set the text inside an element.

NOTE: Any symbols will be displayed as-is rather than interpreted as HTML code.

```
el.innerText = "Hello World";

// User will see the <em>s

el.innerText = "Hello <em>World</em>";
```

Manipulating: HTML

Get and set HTML code inside an element.

NOTE: Interpreted as HTML code.

```
el.innerHTML = "Hello World";
// User will see the “World” emphasized
el.innerHTML = "Hello <em>World</em>";
```

Adding & Removing Elements

Creating elements is quite simple.

Use the document.createElement() method.

The tag name of the element is the parameter of document.createElement().

```
const header = document.createElement("div");
const headerPara = document.createElement("p");
```

Adjust the new element by defining properties.

```
header.style.width = "500px";
header.style.height = "200px";
header.style.backgroundColor = "pink";
headerPara.innerText = "Welcome to the Our
Site!";
headerPara.style.fontSize = "30px";
```

To place these objects into the DOM, use the ParentNode.append() method.

This method will add the argument as the last child of the said parent.

```
document.body.append(header);
header.append(headerPara);
```

Remove an element

Two options:

- remove() - easy, but doesn’t work in IE11
- parent.removeChild(el) - universal

```
el.remove();

el.parentElement.removeChild(el);
```

References

Memorizing all of the properties and methods that the DOM provides is nearly impossible.

Each DOM node has a massive list of properties.

Do not stress trying to memorize everything.

Document methods/properties

Element methods/properties

Events

Events

The browser can register user interactions or events.

If a registered event fires it will execute a function.

These functions can be bound to elements through event handlers.

To attach an event to a node the EventTarget.addEventListener() method is used.

The first argument is the event name.

The second argument is a function that executes when the event fires.

```
element.addEventListener("event_type", handlerFunction);
```

The handler function takes a parameter which has all of the information about the event.

"e" or "event" is a common name for the parameter.

```
function clickHandler(event) {
  console.log("Clicked", event);
}

element.addEventListener("click", clickHandler);
```

Event listeners can also be removed.

If there is a need to remove an event listener it is vital to use a function reference as the event handler and not an anonymous function.

```
function clickHandler(event) {
  console.log("Clicked", event);
}

element.removeEventListener("click",
clickHandler);
```

Event	Description
`load`	When a page finishes loading.
`unload`	When a page is unloading.
`error`	When a JavaScript or asset error occurs.
`resize`	When the browser window resizes.
`scroll`	When the user scrolls.
`keydown`	Pushing down on a key.
`keyup`	Releasing the key.
`keypress`	When a key is pressed.

Event	Description
`click`	A standard mouse click.
`dblclick`	A standard double click.
`mousedown`	Initial press of the mouse.
`mouseup`	Releasing the mouse.
`mouseover`	Mouse moved over an element (not on touch screen).
`mouseout`	Mouse moved off of an element (not on touch screen).

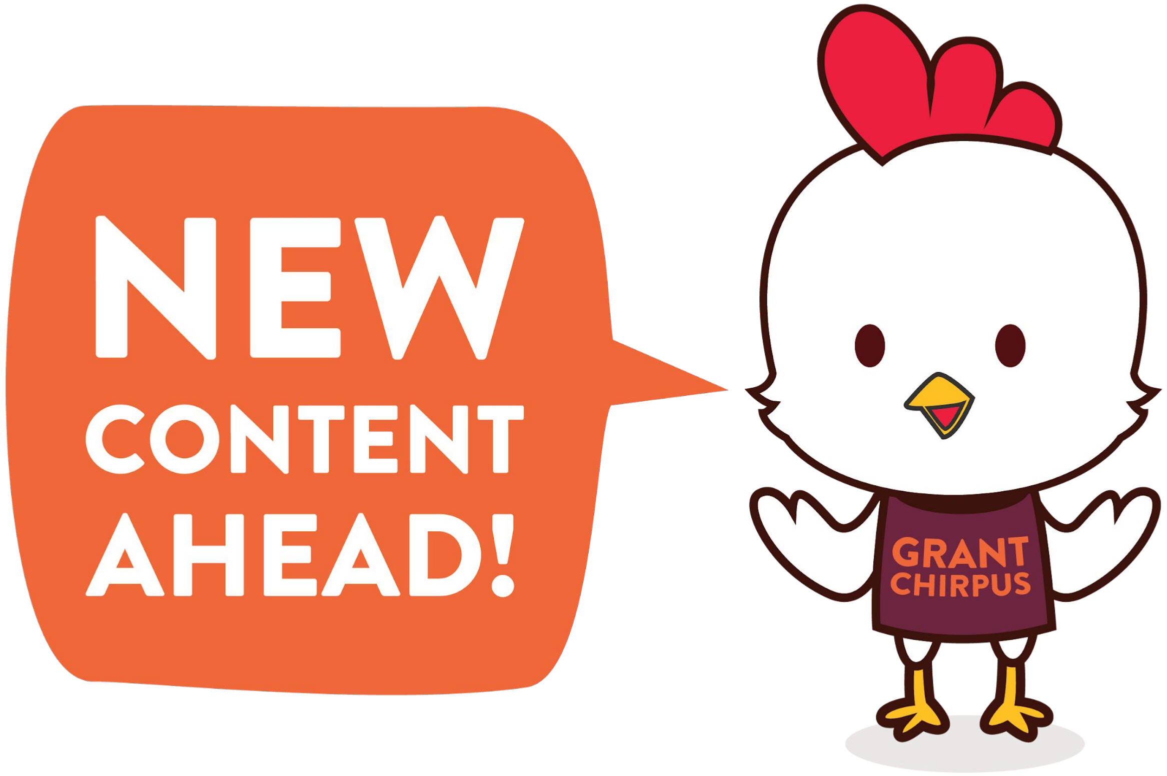
More Events

There are also:

- focus events when the focus is on or leaves an object
- form events for form interactions
- also, many others

Demos

1. Click Demo
2. Mouse Over Demo


NEW CONTENT AHEAD!
GRANT CHIRPUS

Event Delegation

The process of adding events to nodes gets complicated rather quickly.

Consider the following situation:

- When the script executes, it adds a click event to all paragraphs that exist in the DOM.
- A button click adds a new paragraph to the DOM.
- Clicking on the newly appended paragraph does nothing.

The click event mentioned above does not execute.

Why not?

When elements are dynamically inserted into the DOM (after the script executes), elements have no event listeners.

A common fix for this is to add the event listener to a common ancestor.

By doing so, all children of the ancestor or parent will inherit the click event.

Now the event handler requires some additional logic to check if we clicked the correct element.

For example, rather than adding a click event to all listed items, the click event can be added to the unordered list.

Following this process allows for any listed item dynamically inserted into the unordered list to inherit the click event.

Event Bubbling

When an event fires, the event will "bubble" up to the highest level node in the DOM.

If any ancestors have a click event, that click event is fired during the "bubbling" phase.

```
<div>
  <button>Click Me</button>
</div>
```

```html
<div>
  <button>Click Me</button>
</div>
```

```javascript
document.querySelector("button").addEventListener("click",
function(event) {
  event.stopPropagation();
  console.log("button was clicked");
});

document.querySelector("div").addEventListener("click",
function(e) {
  console.log("div was clicked");
});
```

Inputs & Forms

Access input value

The “value” is what is entered or selected in an HTML input, textarea, or select. Get and set the value using **.value**

```
inputEl.value = "Some Value";
inputEl.value = ""; // Clear the input
let userInput = inputEl.value;
```

Access checkboxes & radio buttons

For checkboxes & radio buttons, use **.checked**

```
checkboxEl.checked = true;
if (checkboxEl.checked) {
  ...
}
```

FormData

Another option is to collect all the values from within a form using FormData. Then access them by their “name” attribute.

```
const data = new FormData(formEl);
let firstName = data.get('firstName');
let age = data.get('age');
```

```html
<form id="theForm">
  <input name="firstName" />
  <input name="age" type="number" />
</form>
```

```javascript
const form = document.getElementById('theForm');
const data = new FormData(form);
let firstName = data.get('firstName');
let age = data.get('age');
```

Handling form submission

It is wise to handle the submit event on the form rather than the click event on the submit button.

- The submit event can come from ENTER key as well.
- HTML5 validation is triggered on submit

By default, form submit reloads the page.

We often want to prevent this and handle the form submit in JavaScript. Use **event.preventDefault()**.

```
formEl.addEventListener('submit', e => {
  e.preventDefault();
  // ... handle in JavaScript here.
});
```