DOM Cheatsheet

The Document Object Model

The *Document Object Model*, or DOM is a representation of a document (like an HTML page) as a group of objects. While it is often used to represent HTML documents, and most web browsers use JavaScript interfaces to the DOM, it is language agnostic as a model.

DOM Element Attributes

DOM Nodes of type Element provide access to attributes of those elements declared in HTML markup. For instance, if an HTML tag has attributes like `id` or `class` or `src`, those attributes can be accessed through the DOM.

```
<p id="intro">Welcome to my website</p>
<script>
  const paragraph = document.querySelector('p');
  console.log(paragraph.id) // "intro"
</scrip>
```

`getElementById()` Method

The `document.getElementById()` method returns the element that has the `id` attribute with the specified value.

`document.getElementById()` returns `null` if no elements with the specified ID exists.

An ID should be unique within a page. However, if more than one element with the specified ID exists, the `.getElementById()` method returns the first element in the source

code.

```
const demoElement = document.getElementById('demo');
```

- Here are the methods that can be used to select an element or elements:

Method	Description
`getElementById()`	Selects an individual element within a document using a specific id
`querySelector()`	Uses CSS selector to select the first matching element within a document
`getElementsByClassName()`	Allows you to select all elements with a given class attribute
`getElementsByTagName()`	Locates all elements that match a given tag name
`querySelectorAll()`	Uses CSS selector to select one or more element
	Untitled

The JavaScript `.querySelector()` Method

The `.querySelector()` method selects the first child/descendant element that matches its selector argument.

It can be invoked on the `document` object to search the entire document or on a single element instance to search that element's descendants.

```
// Select the first <div>
const firstDiv = document.querySelector('div');

// Select the first .button element inside .main-navigation
const navMenu = document.getElementById('main-navigation');
const firstButtonChild = navMenu.querySelector('.button');
```

`element.parentNode` Property

The `.parentNode` property of an element can be used to return a reference to its direct parent node. `.parentNode` can be used on any node.

```
<div id="parent">
  <p id ="first-child">Some child text</p>
  <p id ="second-child">Some more child text</p>
</div>
<script>
  const firstChild = document.getElementById('first-child');
  firstChild.parentNode;  // reference to the parent element
</script>
```

Traversing the DOM

The process of selecting another element based on its relationship to a previously selected element.

Property	Description
`parentNode`	Locates the parent element of an initial selection
`previousSibling`	Finds the previous sibling of a selected element
`nextSibling`	Finds the next sibling of a selected element
`firstChild`	Finds the first child of a selected element

`document.createElement()` Method

The `document.createElement()` method creates and returns a reference to a new Element Node with the specified tag name.

`document.createElement()` does not actually add the new element to the DOM, it must be attached with a method such as `element.appendChild()`.

```
const newButton = document.createElement("button");
```

`element.InnerHTML`

The `element.innerHTML` property can be used to access the HTML markup that makes up an element's contents.

`element.innerHTML` can be used to access the current value of an element's contents or to reassign them.

```
<box>
  <p>Hello there!</p>
</box>

<script>
  const box = document.querySelector('p');
  // Outputs '<p>Hello there!</p>':
  console.log(box.innerHTML)
  // Reassigns the value:
  box.innerHTML = '<p>Goodbye</p>'
</script
```

Accessing and Updating Content

The `innerHTML` and `textContent` properties can be used to access or update content:

Property	Description
`innerHTML`	Get or set the HTML content of an element.
`textContent`	Get or set the text content of an element.

The syntax for getting content looks like this:

```
var firstListItem = document.querySelector('li').innerHTML;
// Remember, `querySelector()` selects the first element that matches the provided selector.
```

The syntax for updating content looks like this:

```
document.querySelector('li').innerHTML = 'Email <a href="mom@gmail.com">Mom</a>.';
```

Changing CSS in Javascript with `element.style`.

The `element.style` property can be used to access or set the CSS style rules of an element. To do so, values are assigned to the attributes of `element.style`. In the example code, `blueElement` contains the HTML elements with the ID `colorful-element`. By setting the `backgroundColor` attribute of the `style` property to blue, the CSS property `background-color` becomes blue.

```
let blueElement = document.getElementById('colorful-element');
blueElement.style.backgroundColor = 'blue';
```

.addEventListener()

The `.addEventListener()` method attaches an event handler to a specific event on an event target. The advantage of this is that you can add many events to the event target without overwriting existing events. Two arguments are passed to this method: an event name as a string, and the event handler function. Here is the syntax:

```
eventTarget.addEventListener("event", eventHandlerFunction);
```

.removeEventListener()

We can tell our code to listen for an event to fire using the `.addEventListener()` method. To tell the code to **stop** listening for that event to fire, we can use the `.removeEventListener()` method. This method takes the same two arguments that were passed to `.addEventListener()`, the event name as a string and the event handler function. See their similarities in syntax:

```
eventTarget.addEventListener("event", eventHandlerFunction);
```

```
eventTarget.removeEventListener("event", eventHandlerFunction);
```

Mouse events

A *mouse event* fires when a user interacts with the mouse, either by clicking or moving the mouse device.

- `click` events are fired when the user presses **and** releases a mouse button on an element.
- `mouseout` events are fired when the mouse leaves an element.
- `mouseover` events are fired when the mouse enters an element's content.
- `mousedown` events are fired when the user presses a mouse button.
- `mouseup` events are fired when the user releases the mouse button.

Types of Events

There are many events that can trigger a function. Here are a few:

Event	Description
`'click'`	When a button (usually a mouse button) is pressed and released on a single element.
`'keydown'`	When the user first presses a key on the keyboard.
`'keyup'`	When the user releases a key on the keyboard.
`'focus'`	When an element has received focus.
`'blur'`	When an element loses focus.
`'submit'`	When the user submits a form.
`'load'`	When the page has finished loading.
`'resize'`	When the browser window has been resized.
`'scroll'`	When the user scrolls up or down on the page.