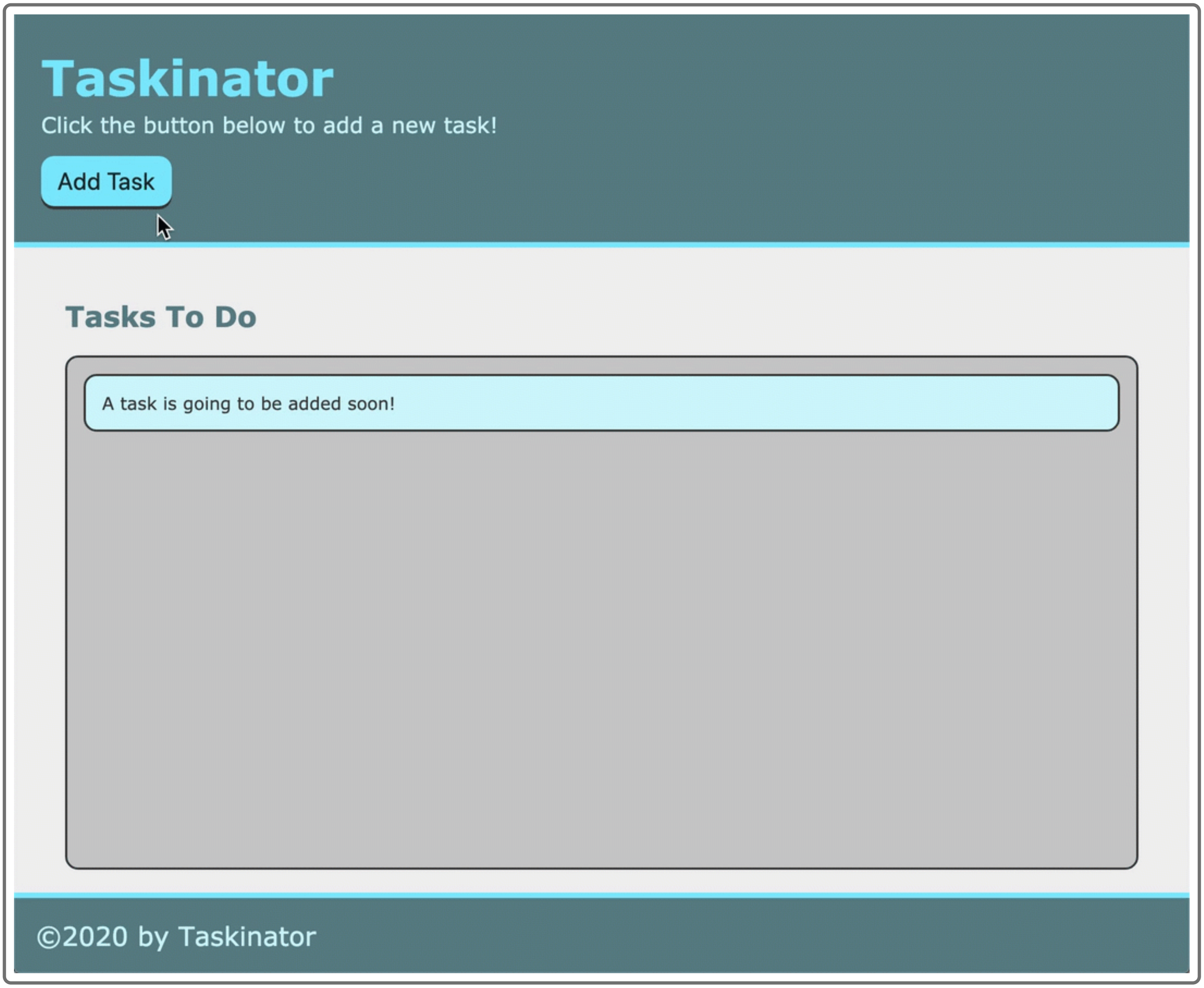
4.1.6

Create a DOM Element

The app looks great so far, but nothing actually happens if you click the "Add Task" button. This button should add a task to the task list, as the following animation shows:


Taskinator

Click the button below to add a new task!

Tasks To Do

A task is going to be added soon!

©2020 by Taskinator

So how do we get the button to work? We can probably guess that JavaScript is involved. But how does that execute and how does it interact with the rest of the webpage's content?

In order to make the button work, you'll use JavaScript to access the browser's built-in properties and methods. The script runs right in the browser, which can interpret and execute JavaScript. In fact, this is one of the benefits of using

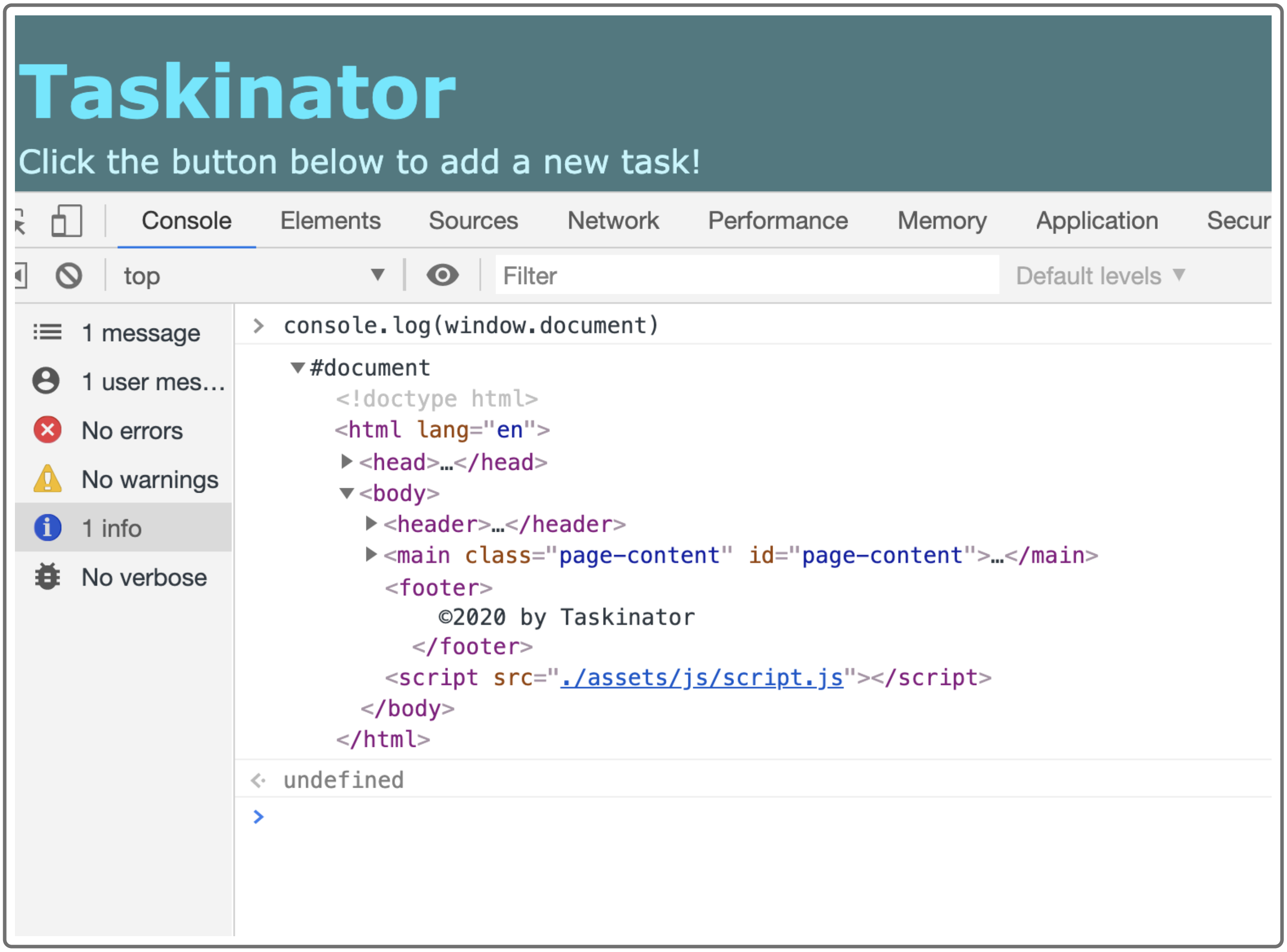
JavaScript. Let's explore how that works.

Browsers consider the HTML code in a webpage as a **document**. That document is actually an object (also known as the `document` object), which is contained inside the `window` object.

Let's use our Chrome DevTools Console detective skills to investigate this. Type the following expression into the browser's console:

```
console.log(window.document);
```

Running this command in the DevTools Console tab should display something like the following image:



```
> console.log(window.document)
    ▼#document
        <!doctype html>
        <html lang="en">
        ▶<head>…</head>
        ▼<body>
          ▶<header>…</header>
          ▶<main class="page-content" id="page-content">…</main>
           <footer>
                ©2020 by Taskinator
             </footer>
           <script src="./assets/js/script.js"></script>
          </body>
        </html>
< undefined
>
```

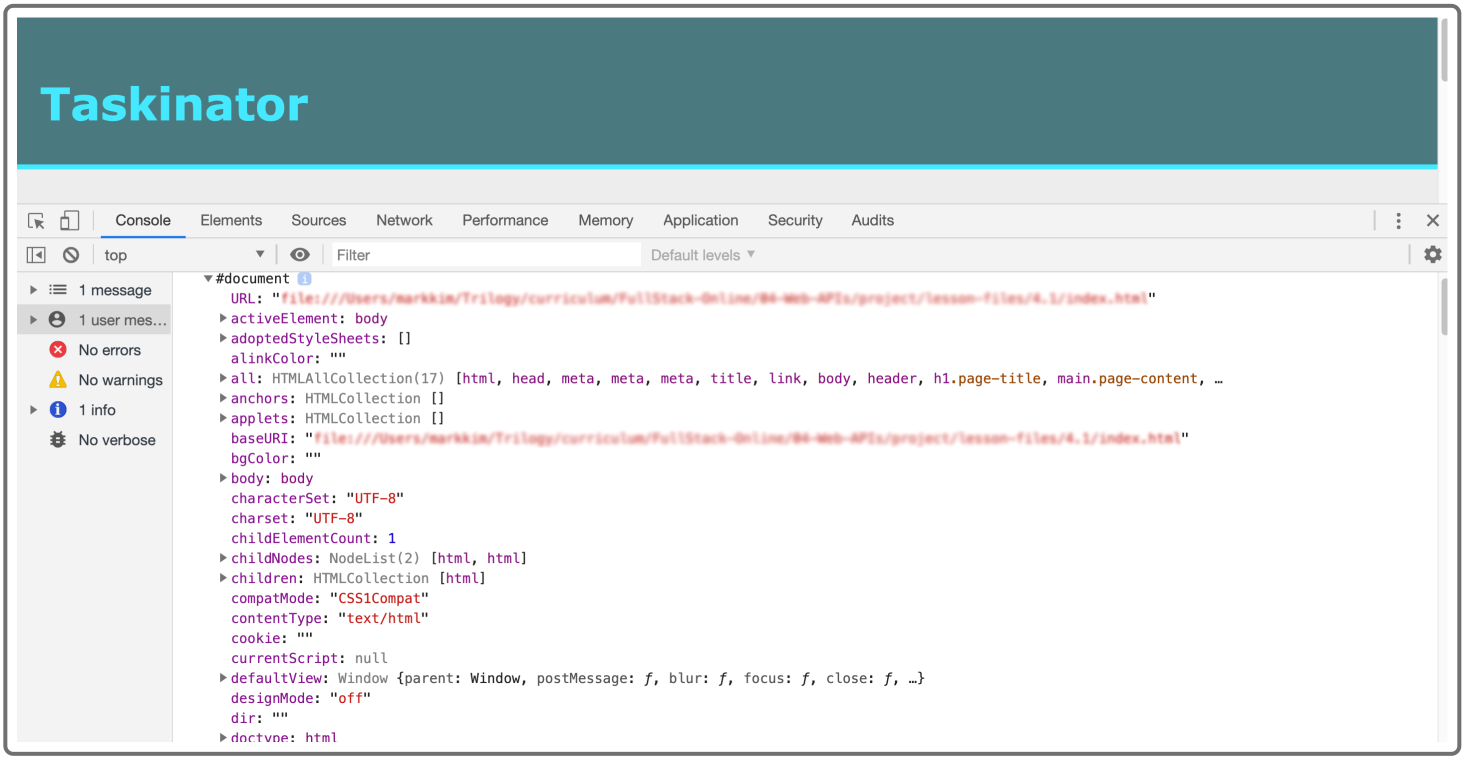
Although this result may look like HTML, it's actually the object representation of the webpage. We call this the **Document Object Model**, or **DOM**. The `document` object represents the root element or the highest-level element of the webpage, which is the `<html>` element in the `index.html` file. The rest of the elements are the descendant elements of the `document` object.

Using the DOM

To better illustrate the object representation of the elements, type the following code into your `script.js` file:

```
console.dir(window.document);
```

Save the file and refresh the `index.html` file in the browser. Now in Chrome DevTools, expand the `document` object, which is shown in the following image:



```
▼#document
  URL: "[redacted]"
 ▶activeElement: body
 ▶adoptedStyleSheets: []
  alinkColor: ""
 ▶all: HTMLAllCollection(17) [html, head, meta, meta, meta, title, link, body, header, h1.page-title, main.page-content, …
 ▶anchors: HTMLCollection []
 ▶applets: HTMLCollection []
  baseURI: "[redacted]"
  bgColor: ""
 ▶body: body
  characterSet: "UTF-8"
  charset: "UTF-8"
  childElementCount: 1
 ▶childNodes: NodeList(2) [html, html]
 ▶children: HTMLCollection [html]
  compatMode: "CSS1Compat"
  contentType: "text/html"
  cookie: ""
  currentScript: null
 ▶defaultView: Window {parent: Window, postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, …}
  designMode: "off"
  dir: ""
 ▶doctype: html
```

Unlike `console.log`, which displays the element's HTML, `console.dir()` displays the HTML element as an object, known as a **DOM element**. And because it is an object, that means we can access its properties and methods using dot notation. In fact, the `document` object allows us to access everything on the webpage, including all of the elements and their attributes, text content, metadata, and much more.

DEEP DIVE ▲

DEEP DIVE

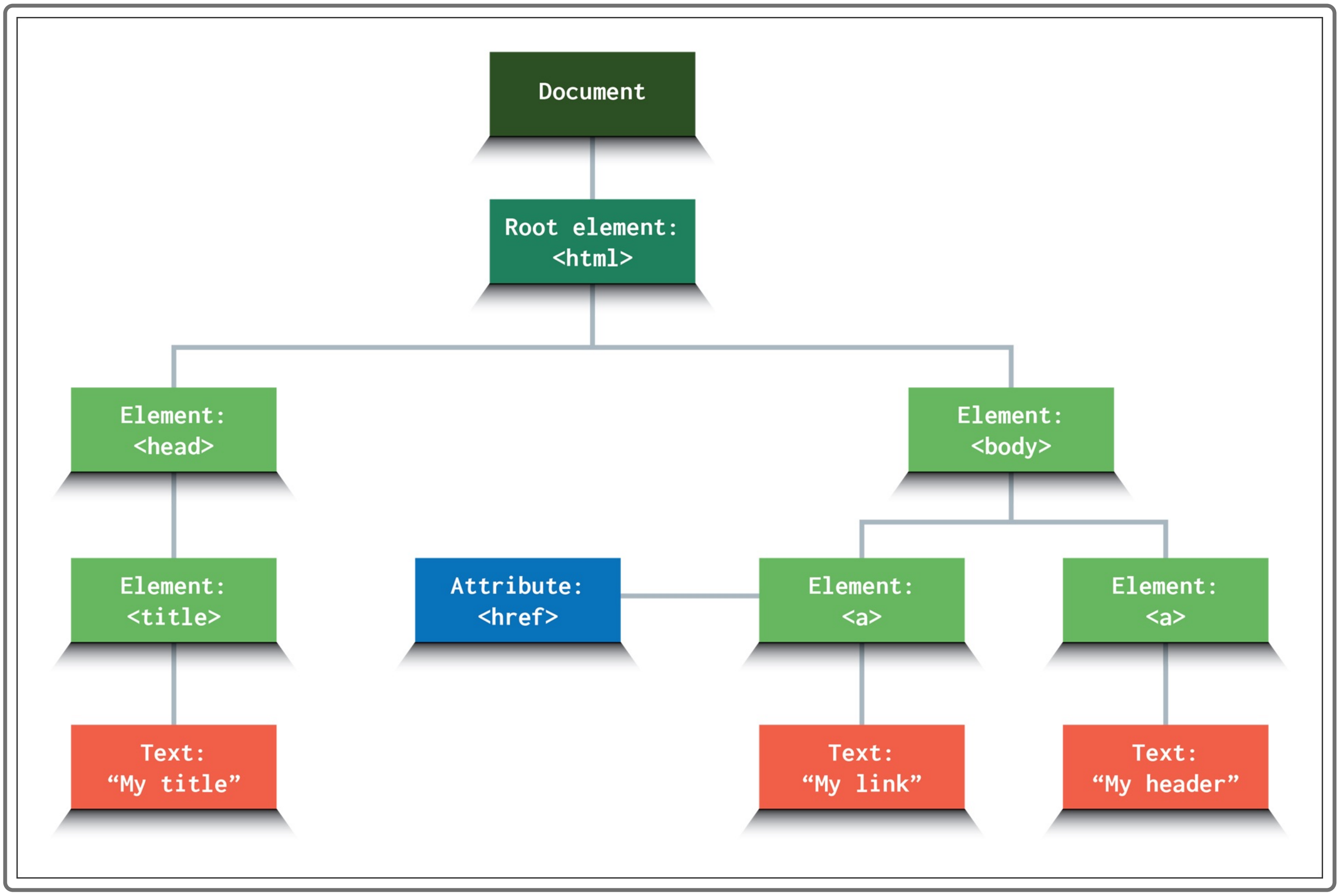
For more information, refer to the **MDN Web Docs on console.dir() (https://developer.mozilla.org/en-US/docs/Web/API/Console/dir)** .

Now that we have an object model of the webpage, we can select existing DOM elements, create new elements, use various built-in DOM methods, or create our own functions to provide rich interactive features like dropdown menus, slideshows, and more.

DEEP DIVE ▲

DEEP DIVE

The DOM is an API for HTML documents, which is organized by the **DOM node tree**, or **DOM tree**. This hierarchy is based on parent-child relationships, as you can see in the following image:



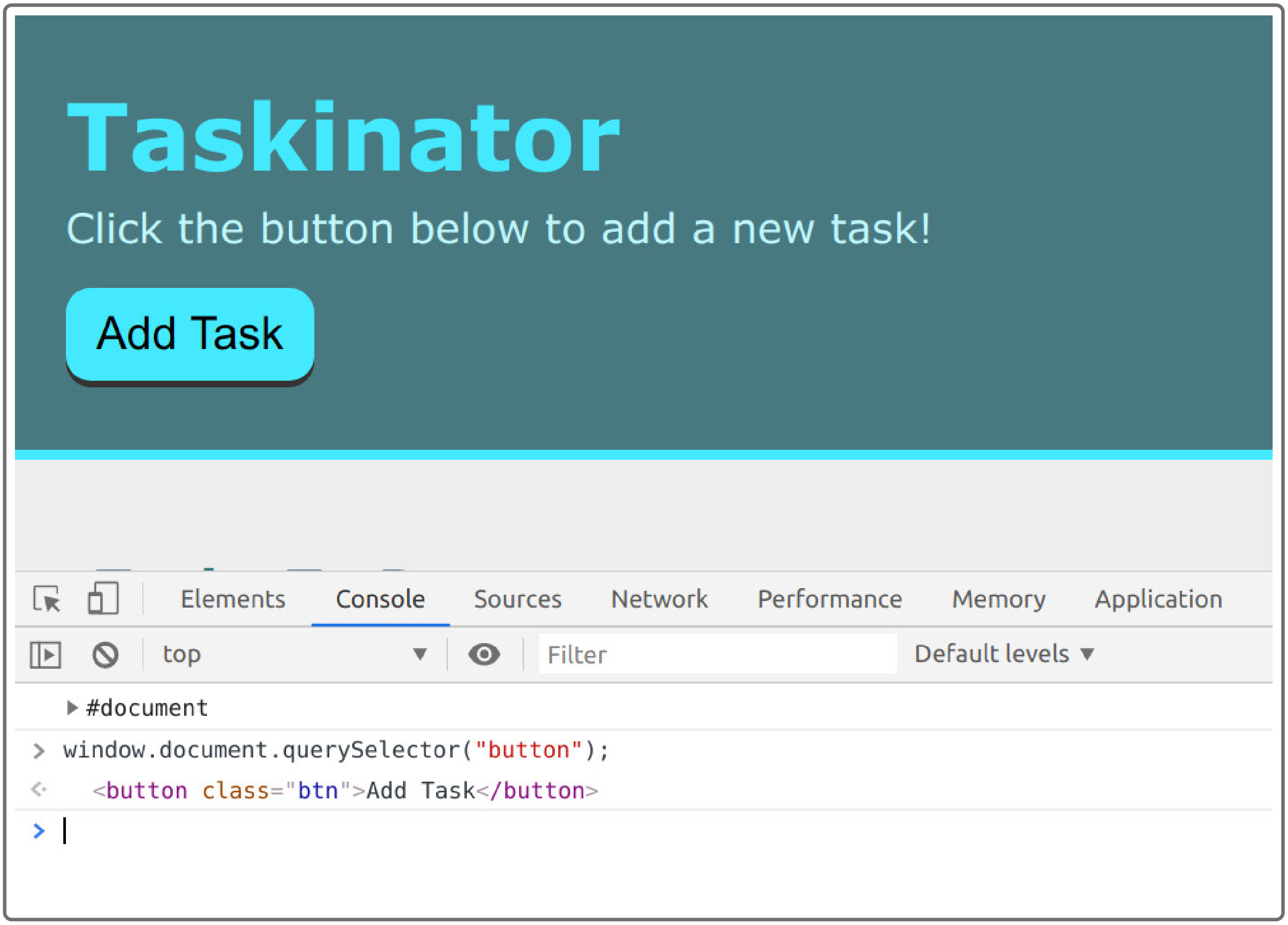
To learn more, refer to the **MDN Web Docs on the DOM** (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) .

The DOM has various methods that allow us to use JavaScript to find specific elements within it. Let's use one of those methods, `querySelector()`, to find the `<button>` and log the results in the console.

Type the following code in the browser console for the `index.html` file:

```
window.document.querySelector("button");
```

You should see something like the following image in the console:

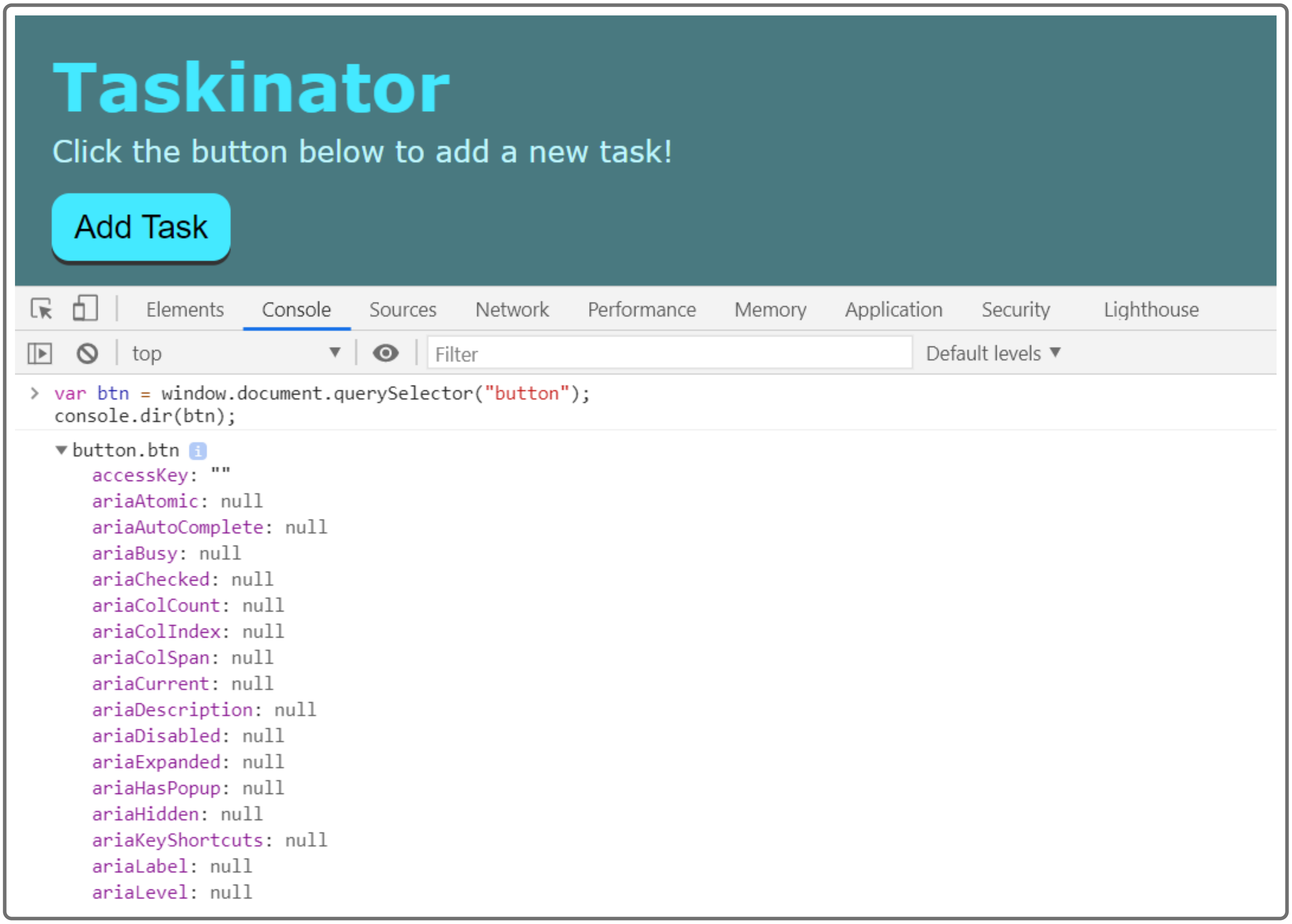


```
▶ #document
> window.document.querySelector("button");
<·  <button class="btn">Add Task</button>
> |
```

This looks like HTML, but it's actually an object representation of the `<button>` element. We can verify that this is actually a DOM element by typing the following into the browser console:

```
var btn = window.document.querySelector("button");
console.dir(btn);
```

This will display an object in the console that looks like the following image:



```
> var btn = window.document.querySelector("button");
  console.dir(btn);
  ▼button.btn
      accessKey: ""
      ariaAtomic: null
      ariaAutoComplete: null
      ariaBusy: null
      ariaChecked: null
      ariaColCount: null
      ariaColIndex: null
      ariaColSpan: null
      ariaCurrent: null
      ariaDescription: null
      ariaDisabled: null
      ariaExpanded: null
      ariaHasPopup: null
      ariaHidden: null
      ariaKeyShortcuts: null
      ariaLabel: null
      ariaLevel: null
```

Did you notice the `document` prefix on the `querySelector()` method? The `document` is the root DOM element, which represents the `index.html` file that's open in the browser. We can use `querySelector()` to select any element in the HTML, including the `<button>` that we need here.

The `querySelector()` method searches down from the `document` object through all the descendant elements. In addition to element types—like `<button>`—the versatile `querySelector()` method can also search selectors and attributes.

Experiment in the console by selecting different elements. Try to target the `<body>` or `<main>`. You'll find that all the elements in HTML are available to target.

Can you select an HTML element by its class attribute value? Doing so requires a little trick.

SHOW HINT

To select a class attribute, you need to add a dot (`.`) prefix, as shown in the following code snippet:

```
document.querySelector(".btn");
```

Here, we chose the class attribute `.btn` on the `<button>` attribute.

REWIND

This is the same syntax we used for CSS class selectors!

The same object will be displayed in the console for the `<button>` element, even though we chose a different selector.

Notice that we didn't add `window` before `document` in the expression above. A `window` prefix is unnecessary because we opened `index.html` in the browser, where `window` is a global object.

REWIND

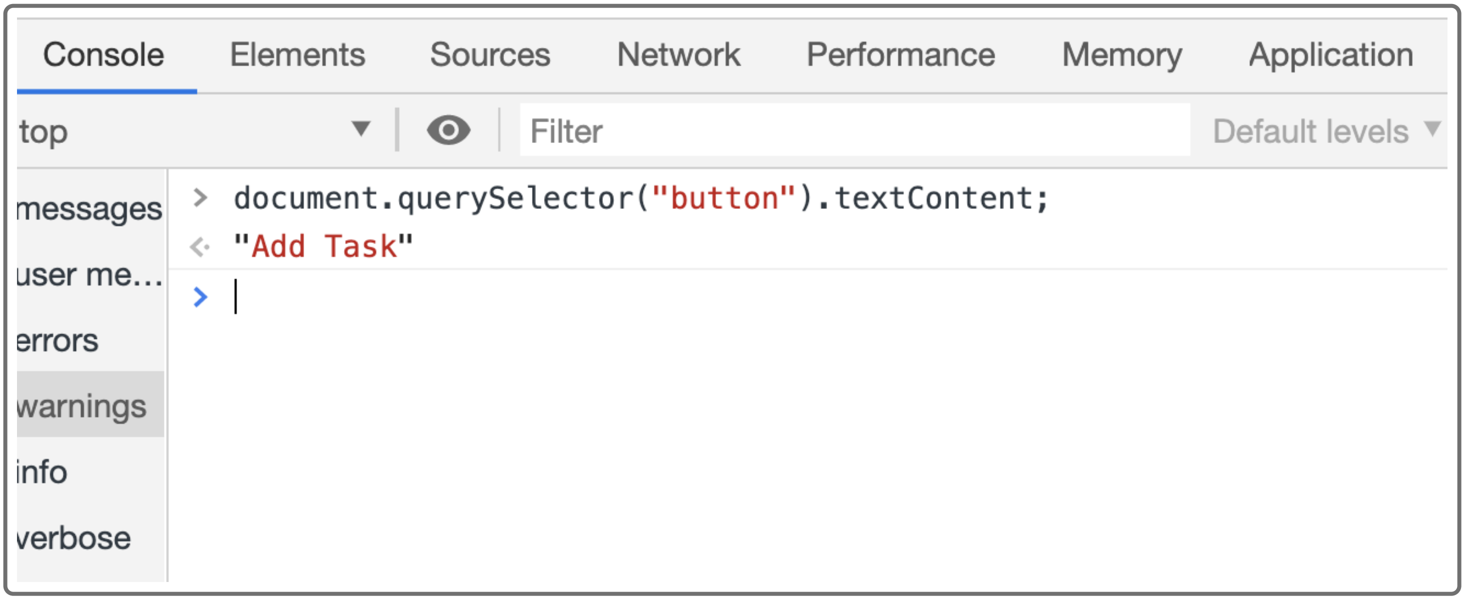
You might recall similarly dropping the `window` prefix from the `Math` object when using `Math.random()`. Incidentally, `alert()` and `prompt()` can also function without the `window` prefix. We'll drop it from here going forward, because shortcuts make a developer's life much sweeter!

Even though the display in the console for the `<button>` query looks like raw HTML, it's actually an object representation of this element. And because it's an object, we have access to its built-in properties and methods.

Let's investigate one of these properties—`textContent`—by typing the following code into the console:

```
document.querySelector("button").textContent;
```

The following image shows what this looks like in the console:



```
> document.querySelector("button").textContent;
<· "Add Task"
>
```

As you can see, the console displays "Add Task". Thus we can conclude that the aptly named `textContent` property returns the text content of the element!

You were able to select and get the text content of the button, but if you added more buttons to the page, how would you distinguish this button from the rest? To do this, we can use an `id` attribute and assign it an id name. Let's try that now.

In the `index.html` file, add the `id="save-task"` attribute to the `button` element so that it looks like the following code:

```
<button class="btn" id="save-task">Add Task</button>
```

Save the file and refresh the browser so that the DOM has the `id` attribute on the element. Then update the `querySelector()` call to look for the `id` attribute instead of the generic element.

Type the following code into the console to see how the result looks:

```
document.querySelector("#save-task");
```

The result should look like the `<button class="btn" id="save-task">` element that you just updated in the HTML.

Having successfully targeted the HTML element, now we can add this code into our JavaScript file, `script.js`. To do that, delete `console.dir(window.document);` and then add the following code to assign the button element object representation to a variable in the file:

```
var buttonEl = document.querySelector("#save-task");
console.log(buttonEl);
```

The name of the button element is `buttonEl`. The use of camelCase marks the element as a JavaScript variable. The `El` suffix identifies this as a DOM element; this naming convention will help us keep track of which variables store DOM elements.

SHOW PRO TIP

To see the expression in action, save the JavaScript file and refresh `index.html` in the browser.

IMPORTANT

> Have you tried opening the `script.js` file in the browser and noticed that you can't? Why is that? Think about the HTML file as the subject matter and the style sheet and script files as the modifiers or enhancements. The HTML file provides a canvas to apply styles and behaviors and will always be the connection to the web browser. The HTML file acts as the hub, connecting the supplemental files with relative file paths.

As we can see in the console, the `buttonEl` variable now represents the same `button` element we displayed earlier. Now that we've selected the correct element in the DOM and preserved the element reference in the script, how can we add a task to the task list by using a button click?