100 XP

Exercise - Add interactivity with JavaScript

10 minutes

JavaScript (or *ECMAScript*) is a programming language that helps you add interactivity to your web pages.

For example, you can use JavaScript to define the behavior that happens when a user selects a button; for example, open a pop-up window. Using JavaScript, you can add or remove content from a web page without reloading it.

In this unit, you set up an example JavaScript file for your web page. In it, you create a button to switch between light and dark themes. Then, you attach the button to JavaScript code that performs the actual theme switching. Finally, you check the finished project using your browser's developer tools.

Link to JavaScript

Like CSS, you could add JavaScript directly to the HTML file, but a recommended best practice is to save your JavaScript in a separate file. Adding your JavaScript code to a separate file makes it easier to reuse it across several web pages. For example, you could create a pop-up alert by adding the following code anywhere within the body of your web pages:

HTML

```
<script>alert('Hello World')</script>
```

However, it's better to add your JavaScript code to a separate file that can be linked to every file that needs your custom functionality.

The HTML script tag `<script>` lets us link to an external JavaScript file, which is how you configure your web app in this exercise.

1. In **Visual Studio Code**, open your `index.html` file.
2. Find the closing `</body>` element and place your cursor on a new line above it. Enter `script:src` and then select `Enter`. The opening and closing tags for a `<script>` element are added to your code.
3. Modify the `<script>` element to load your `app.js` file as shown in the following example. Ensure that it's located after the closing `</ul>` element for the list.

   HTML

   ```
   ...
   <ul>
     <li class="list">Add visual styles</li>
     <li class="list">Add light and dark themes</li>
     <li>Enable switching the theme</li>
   </ul>
   <script src="app.js"></script>
   ...
   ```

The `<script>` element could be placed in the `<head>` or elsewhere in the `<body>`. However, putting the `<script>` element at the end of the `<body>` section enables all the page content to display on the screen first, before the script is loaded.

Add fault tolerance

1. In your HTML file, add a `<noscript>` element after the closing `</script>` tag, which can be used to show a message if JavaScript is deactivated.

   HTML

   ```
   <script src="app.js"></script>
   <noscript>You need to enable JavaScript to view the full site.</noscript>
   ```

   Adding the `<noscript>` element is an example of *fault tolerance* or *graceful degradation*. When you use the `<noscript>` element, your code can detect and plan for when a feature isn't supported or available.

Set strict mode

JavaScript was designed to be easy to learn and allows certain mistakes to be made by the developer. For example, JavaScript doesn't throw an error when you use a misspelled variable, and instead creates a new global one. When you start learning JavaScript, having fewer errors is convenient. However, it can lead to writing code that is harder for browsers to optimize and harder for you to debug.

Switch to strict mode to get more useful errors when you make mistakes.

- In **Visual Studio Code**, open the `app.js` file, and enter the following.

```JavaScript
'use strict';
```

Add a button

You need a way to let your users switch between the light and dark themes in your web page. In this exercise, you implement that functionality with an HTML `<button>` element.

1. In your HTML file (`index.html`), add a `<button>` element. Put the button inside of a `<div>` element and add it just after the end of the list (`</ul>`).

```HTML
...
<ul>
  <li class="list">Add visual styles</li>
  <li class="list">Add light and dark themes</li>
  <li>Enable switching the theme</li>
</ul>
<div>
  <button class="btn">Dark</button>
</div>
<script src="app.js"></script>
...
```

   Notice that the `<button>` element in this example has a *class* attribute that you can use to apply CSS styles.

2. Save the changes to your HTML file with the keyboard shortcut `Control+S` on Windows or `Command+S` on macOS.

3. In your CSS file (`main.css`), add a new rule with a `.btn` class selector for your HTML button. To make the button colors different from the general light or dark theme colors, set the `color` and `background-color` properties in this rule. When your page displays, these `.btn` properties override the default properties set in the `body` rule of your CSS file.

```css
.btn {
  color: var(--btnFontColor);
  background-color: var(--btnBg);
}
```

4. Next, modify the `.btn` rule to add some styles for the size, shape, appearance, and placement of the button. The following CSS creates a round button to the right of the page heading.

```css
.btn {
  position: absolute;
  top: 20px;
  left: 250px;
  height: 50px;
  width: 50px;
  border-radius: 50%;
  border: none;
  color: var(--btnFontColor);
  background-color: var(--btnBg);
}
```

5. Next, update the CSS for the light and dark theme. Define some new variables, `--btnBg` and `--btnFontColor`, to specify the button-specific background color and font color.

```css
```

```
.light-theme {
  --bg: var(--green);
  --fontColor: var(--black);
  --btnBg: var(--black);
  --btnFontColor: var(--white);
}

.dark-theme {
  --bg: var(--black);
  --fontColor: var(--green);
  --btnBg: var(--white);
  --btnFontColor: var(--black);
}
```

6. Save the changes to your CSS file with the keyboard shortcut `Control+S` on Windows or `Command+S` on macOS.

Add an event handler

To make the button do something when you select it, you need an event handler in your JavaScript file. An event handler is a way to run a JavaScript function when an event happens on the page. For the button, let's add an event handler for the `click` event; the event handler function runs when the `click` event occurs.

Before you can add the event handler, you need a reference to the button element.

1. In your JavaScript file (`app.js`), use `document.querySelector` to get the button reference.

   JavaScript

   ```
   const switcher = document.querySelector('.btn');
   ```

   The `document.querySelector` function uses CSS selectors, just like the ones you used in your CSS file. `switcher` is now a reference to the button in the page.

2. Next, add the event handler for the `click` event. In the following code, you add a listener for the `click` event and define an event handler function that the browser executes when the `click` event occurs.

   JavaScript

   ```
   switcher.addEventListener('click', function() {
       document.body.classList.toggle('light-theme');
       document.body.classList.toggle('dark-theme');
   });
   ```

In the preceding code, you used the `toggle` method to modify the `<body>` element's class attribute. This method automatically adds or removes the `light-theme` and `dark-theme` classes. This code applies the dark styles instead of light styles on click, and then light styles instead of dark if you click again.

However, the label for the button also needs to be updated to show the correct theme, so you need to add an `if` statement to determine the current theme, and update the button label.

Here's what your JavaScript code should look like with the event handler added.

JavaScript

```
'use strict';

const switcher = document.querySelector('.btn');

switcher.addEventListener('click', function() {
    document.body.classList.toggle('light-theme');
    document.body.classList.toggle('dark-theme');

    const className = document.body.className;
    if(className == "light-theme") {
        this.textContent = "Dark";
    } else {
        this.textContent = "Light";
    }
});
```

It's a JavaScript convention to use *camel case* for variable names with more than one word; for example: the variable `className`.

Console message

As a web developer, you can create hidden messages that aren't visible on your webpage, but that you can read in the Developer Tools, in the **Console** tab. Using *console messages* is helpful for seeing the result of your code.

In your JavaScript file, add a call to `console.log` after the `if` statement, but inside the event listener.

After you make this change, your complete JavaScript code should look like this.

JavaScript

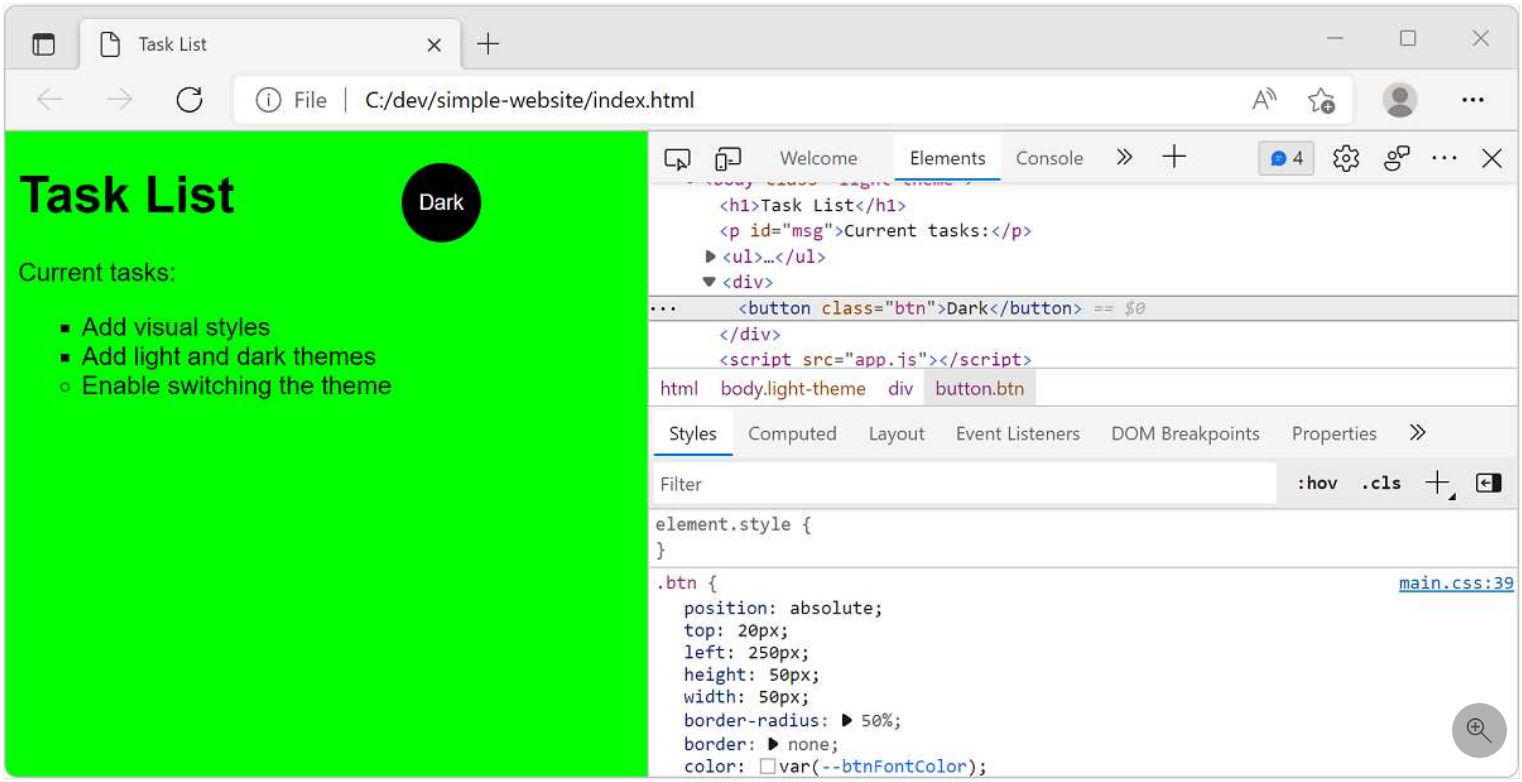
```
'use strict';

const switcher = document.querySelector('.btn');

switcher.addEventListener('click', function() {
    document.body.classList.toggle('light-theme');
    document.body.classList.toggle('dark-theme');

    const className = document.body.className;
    if(className == "light-theme") {
        this.textContent = "Dark";
    } else {
        this.textContent = "Light";
    }

    console.log('current class name: ' + className);
});
```

When you are in a JavaScript file in **Visual Studio Code**, you can use autocomplete for `console.log` by entering `log`, and then pressing `Enter`.

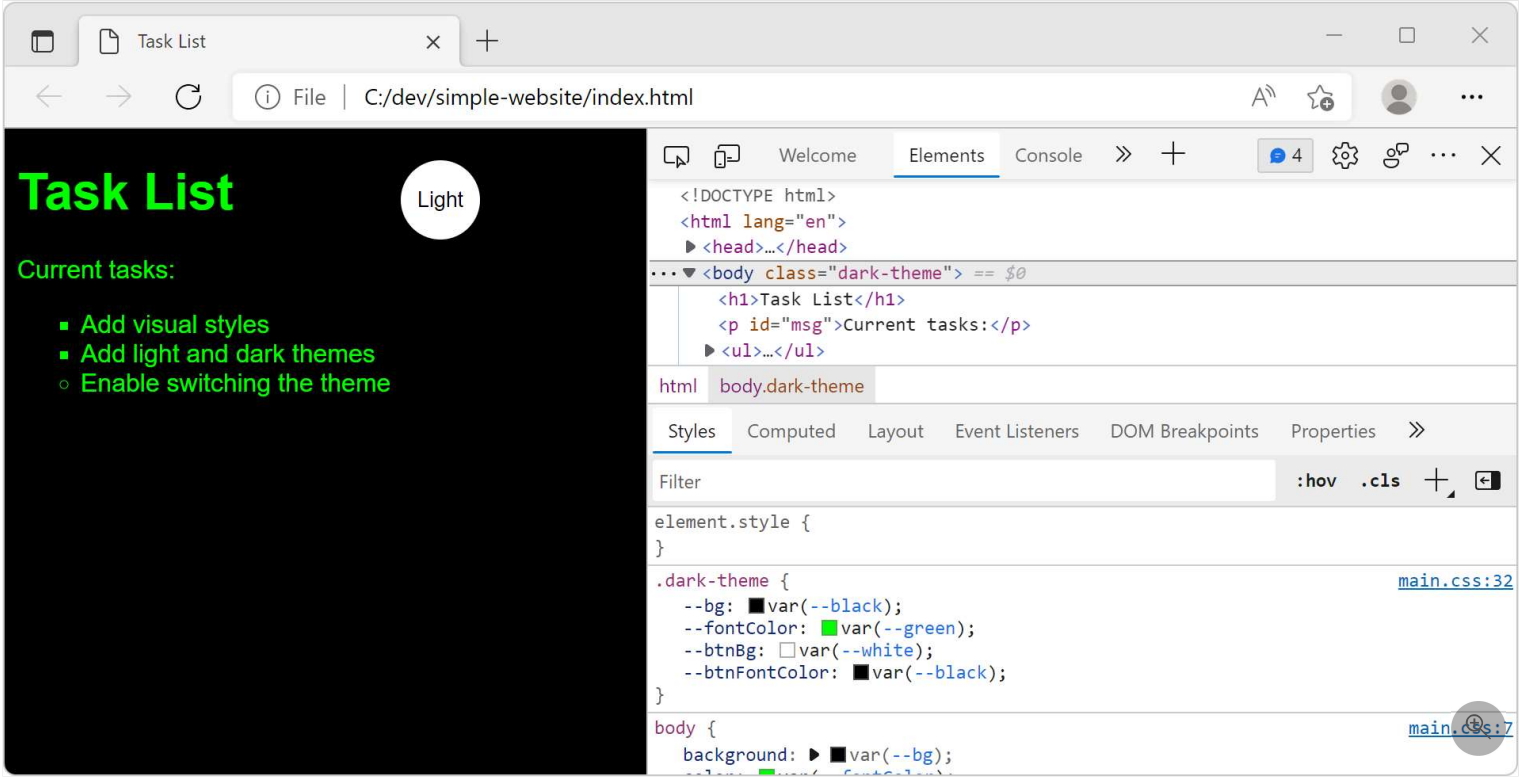
You can define a text *string* with single or double quotes around the text.

Open in the browser

1. To preview, select `index.html`, and select **Open In Default Browser**, or reload the same browser tab by pressing `F5`.



2. Select the new **Dark** button to switch to the dark theme.



3. Make sure that everything looks correct and behaves as expected. If not, you should review the preceding steps to see if you missed something

Check the page in the developer tools

1. Open Developer Tools.

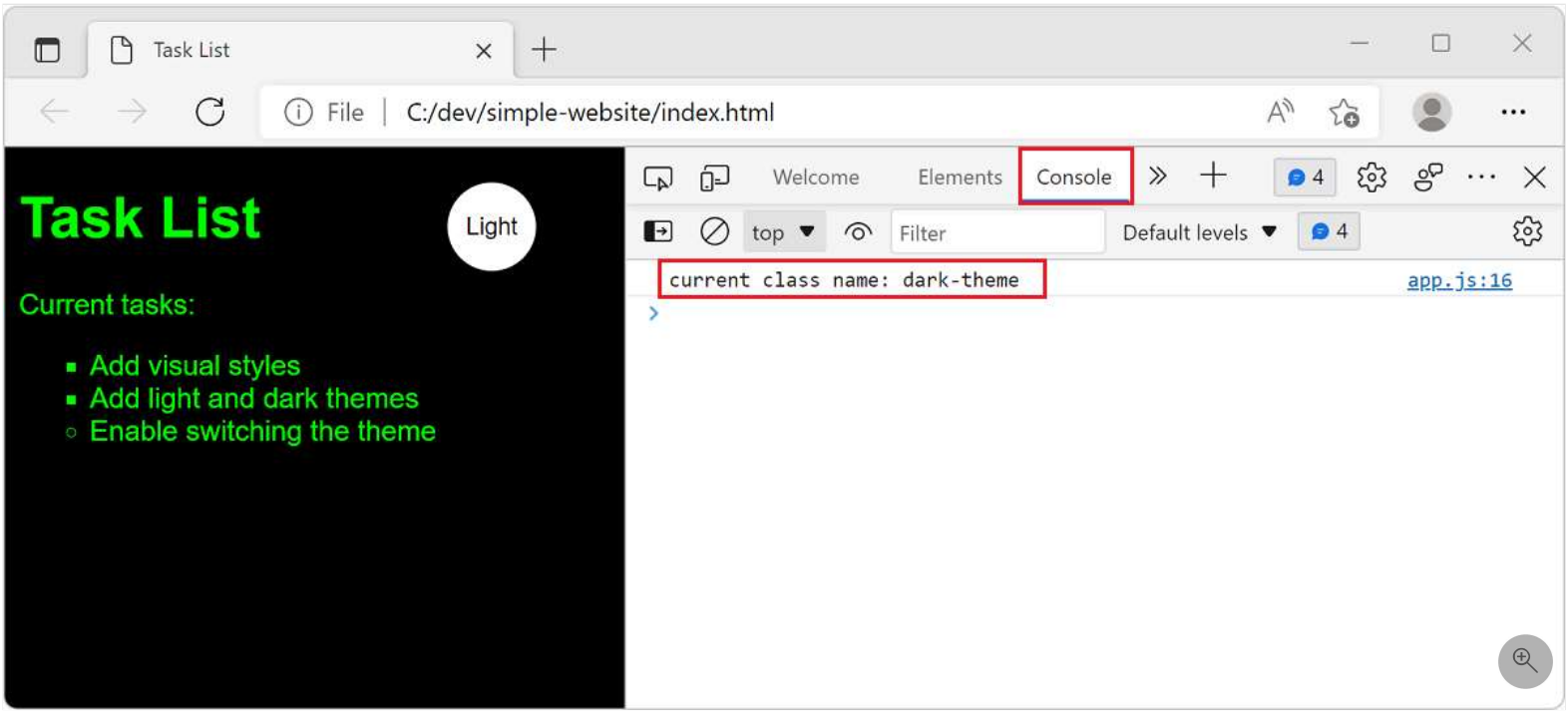
   - Right-click and select **Inspect**, or use the keyboard shortcut `F12`. Alternatively, use the `Ctrl+Shift+I` shortcut on Windows or Linux, and `Option+Command+I` on macOS.

2. Select the **Elements** tab and, inside the **Elements** tab, select the **Styles** tab.

3. Select the `<body>` element. In the **Styles** tab, look at the applied theme. If the current theme is dark, the `dark-theme` styles are applied.

   Make sure the dark theme is selected.

4. Select the **Console** tab to see the `console.log` message, `current class name: dark-theme`.



Using the console, you can get interesting insights from your JavaScript code. Add more console messages to understand which parts of your code are getting executed and to know the current values of other variables.

To learn more about the console, check out the Console overview article.

Next unit: Knowledge check

Continue >