codecademy

JavaScript and the DOM

HTML script element src attribute

The `src` attribute of a `<script>` element is used to point to the location of a script file.
The file referenced can be local (using a relative path) or hosted elsewhere (using an absolute path).

```
<!-- Using a relative path -->
<script src="./script.js"></script>

<!-- Using an absolute path -->
<script
src="https://code.jquery.com/jquery-
3.3.1.min.js"></script>
```

HTML script element defer attribute

The `defer` attribute of a `<script>` tag is a boolean attribute used to indicate that the script can be loaded but not executed until after the HTML document is fully parsed. It will only work for externally linked scripts (with a `src` attribute), and will have no effect if it is applied to an inline script.
In the example code block, the `<h1>` tag will be loaded and parsed before the script is executed due to the `defer` attribute.

```
<body>
  <script src="main.js" defer></script>
  <h1>Hello</h1>
</body>
```

HTML script tag async attribute

Scripts are loaded synchronously as they appear in an HTML file, before the following HTML is loaded and parsed. The `async` attribute can be used to load the scripts asynchronously, such that they will load in the background without blocking the HTML parser from continuing.
In the example code block, the script will load asynchronously in the background, without blocking the HTML parser.

```
<body>
  <script src="main.js" async></script>
  <h1>Hello world!</h1>
</body>
```

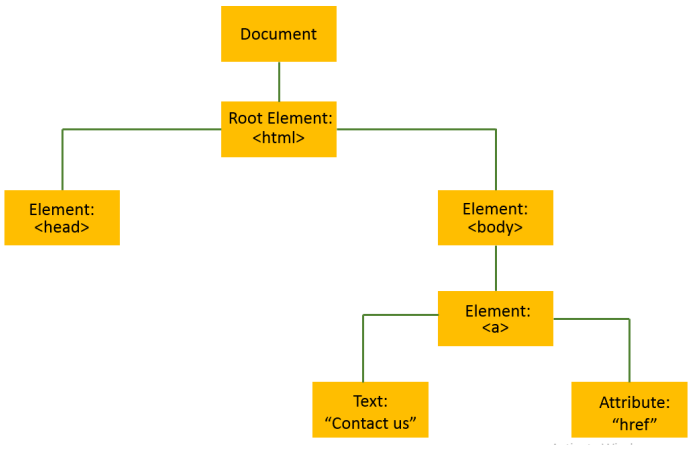
HTML script element

The HTML `<script>` element can contain or reference JavaScript code in an HTML file. The `<script>` element needs both an opening and a closing tag, and JavaScript code can be *embedded* between the tags.

```
<script>
  console.log("Hello world!");
</script>
```

Nodes in DOM tree

A *node* in the DOM tree is the intersection of two branches containing data. Nodes can represent HTML elements, text, attributes, etc. The *root node* is the top-most node of the tree. The illustration shows a representation of a DOM containing different types of nodes.



HTML DOM

The DOM is an interface between scripting languages and a web page's structure. The browser creates a Document Object Model or DOM for each webpage it renders. The DOM allows scripting languages to access and modify a web page. With the help of DOM, JavaScript has the ability to create dynamic HTML.

Accessing HTML attributes in DOM

The DOM nodes of type Element allow access to the same attributes available to HTML elements. For instance, for the given HTML element, the `id` attribute will be accessible through the DOM.

```
<h1 id="heading">Welcome!</h1>
```

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.
The DOM is tree-like and heirarchical, meaning that there is a single top-level object, and other objects descend from it in a branching structure.

The `removeChild()` Method

The `.removeChild()` method removes a specified child from a parent element. We can use this method by calling `.removeChild()` on the parent node whose child we want to remove, and passing in the child node as the argument.
In the example code block, we are removing `iceCream` from our `groceryList` element.

```
const groceryList =
document.getElementById('groceryList');
const iceCream =
document.getElementById('iceCream');

groceryList.removeChild(iceCream);
```

The `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.
In the code block above, we are calling on the `parentNode` of the `#first-child` element to get a reference to the `#parent` `div` element.

```
<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 div
</script>
```

The `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");
```

The `element.InnerHTML` Property

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 it.
In the code block above, we are reassigning the `box` element's inner HTML to a paragraph element with the text "Goodbye".

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

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

The `document` Object

The `document` object provides a Javascript interface to access the DOM. It can be used for a variety of purposes including referencing the `<body>` element, referencing a specific element with ID, creating new HTML elements, etc.
The given code block can be used to obtain the reference to the `<body>` element using the `document` object.

```
const body = document.body;
```

The `document.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.

```
// Save a reference to the element with id 'demo':
const demoElement = document.getElementById('demo');
```

The `.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.
In the above code block, we are using `.querySelector()` to select the first `div` element on the page, and to select the first element with a class of `button`, inside the `.main-navigation` element.

```
// 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');
```

The `document.body` Object

`document.body` returns a reference to the contents of the `<body>` HTML element of a document/HTML page.
The `<body>` element contains all the visible contents of the page.

The `element.onclick` Property

The `element.onclick` property can be used to set a function to run when an element is clicked. For instance, the given code block will add an `<li>` element each time the element with ID `addItem` is clicked by the user.

```
let element =
document.getElementById('addItem');
element.onclick = function() {
  let newElement =
document.createElement('li');

document.getElementById('list').appendChil
d(newElement);
};
```

The `element.appendChild()` Method

The `element.appendChild()` method appends an element as the last child of the parent.
In the given code block, a newly created `<li>` element will be appended as the last child of the HTML element with the ID `list`.

```
var node1 = document.createElement('li');
document.getElementById('list').appendChil
d(node1);
```

The `element.style` Property

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 element with the ID `colorful-element`. By setting the `backgroundColor` attribute of the `style` property to blue, the CSS property `background-color` becomes blue.
Also note that, if the CSS property contains a hyphen, such as `font-family` or `background-color`, Camel Case notation is used in Javascript for the attribute name, so `background-color` becomes `backgroundColor`.

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

The DOM Parent-Child Relationship

The parent-child relationship observed in the DOM is reflected in the HTML nesting syntax.
Elements that are nested inside the opening and closing tag of another element are the children of that element in the DOM.
In the code block, the two `<p>` tags are children of the `<body>`, and the `<body>` is the parent of both `<p>` tags.

```
<body>
  <p>first child</p>
  <p>second child</p>
</body>
```

Print Share

0

codecademy

DOM Events with JavaScript

.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);
```

Event handler

When an event fires in JavaScript (such as a keystroke or mouse movement), an *event handler* runs in response. Each event handler is registered to an element, connecting the handler to both an element and a type of event (keystroke, eg.). A method called an *event listener* "listens" for an event to occur, specifies what should happen as a response, and calls the event handler.

Event object

Event handler functions are passed an argument called an *event object*, which holds information about the event that was fired.
Event objects store information about the event target, the event type, and associated listeners in properties and methods. For example, if we wanted to know which key was pressed, the event object would store that information.

Keyboard events

Keyboard events describe a user interaction with the keyboard. Each event describes a separate interaction with a key on the keyboard by the user, which are then stored with the `.key` property.

- `keydown` events are fired when the key is first pressed.
- `keyup` events are fired when the key is released.
- `keypress` events are fired when the user presses a key that produces a character value (aka is not a modifier key such as CapsLock).

javascript event

On a webpage, a trigger such as a user interaction or browser manipulation can cause a client-side JavaScript event to be created. Events can be used to manipulate the DOM by executing a JavaScript function.
Events can include anything from a click to hovering a mouse over an element to a webpage loading or being refreshed. Events are defined as a part of the JavaScript API built into the web browser.

```
// An event is triggered when a user
clicks on the #button element,
// which then sets the #button element's
background-color to blue.
$('#button').on('click', event => {
  $(event.currentTarget).css('background-
color', 'blue');
});
```

JS Event Handlers

The goal of JavaScript is to make a page dynamic, which frequently means responding to certain events (for example, button clicks, user scrolls, etc). DOM elements can have functions hook onto events. The functions are called *event handlers* and the DOM element is known as an *event target*.
The example code block shows how to *register* a function as an *event handler*. The property name for event handlers starts with 'on' with the event appended afterwards. Examples: `onload`, `onclick`, `onfocus`, `onscroll`.

```
//Assuming there is an element with
ID='test' on the page

document.getElementById('test').onclick =
function(e) {
  alert('Element clicked!');
};
```

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.

Print Share

0

codecademy

HTML Forms

`<input>`: Checkbox Type

When using an HTML `input` element, the `type="checkbox"` attribute will render a single checkbox item. To create a group of checkboxes related to the same topic, they should all use the same `name` attribute. Since it's a checkbox, multiple checkboxes can be selected for the same topic.

```
<input type="checkbox" name="breakfast"
value="bacon">Bacon <br>
<input type="checkbox" name="breakfast"
value="eggs">Eggs <br>
<input type="checkbox" name="breakfast"
value="pancakes">Pancakes <br>
```

`<textarea>` Element

The `textarea` element is used when creating a text-box for multi-line input (e.g. a comment section). The element supports the `rows` and `cols` attributes which determine the height and width, respectively, of the element.

When rendered by the browser, `textarea` fields can be stretched/shrunk in size by the user, but the `rows` and `cols` attributes determine the initial size.

Unlike the `input` element, the `<textarea>` element has both opening and closing tags. The `value` of the element is the content in between these tags (much like a `<p>` element). The code block shows a `<textarea>` of size 10x30 and with a `name` of `"comment"`.

```
<textarea rows="10" cols="30"
name="comment"></textarea>
```

`<form>` Element

The HTML `<form>` element is used to collect and send information to an external source.
`<form>` can contain various input elements. When a user submits the form, information in these input elements is passed to the source which is named in the `action` attribute of the form.

```
<form method="post"
action="http://server1">
  Enter your name:
  <input type="text" name="fname">
  <br/>
  Enter your age:
  <input type="text" name="age">
  <br/>
  <input type="submit" value="Submit">
</form>
```

`<input>`: Number Type

HTML input elements can be of type `number`. These input fields allow the user to enter only numbers and a few special characters inside the field.
The example code block shows an input with a type of `number` and a name of `balance`. When the input field is a part of a form, the form will receive a key-value pair with the format: `name: value` after form submission.

```
<input type="number" name="balance" />
```

`<input>` Element

The HTML `<input>` element is used to render a variety of input fields on a webpage including text fields, checkboxes, buttons, etc. `<input>` element have a `type` attribute that determines how it gets rendered to a page.
The example code block will create a text input field and a checkbox input field on a webpage.

```
<label for="fname">First name:</label>
<input type="text" name="fname"
id="fname"><br>

<input type="checkbox" name="vehicle"
value="Bike"> I own a bike
```

`<input>`: Range Type

A slider can be created by using the `type="range"` attribute on an HTML `input` element. The range slider will act as a selector between a minimum and a maximum value. These values are set using the `min` and `max` attributes respectively. The slider can be adjusted to move in different steps or increments using the `step` attribute.
The range slider is meant to act more as a visual widget to adjust between 2 values, where the relative position is important, but the precise value is not as important. An example of this can be adjusting the volume level of an application.

```
<input type="range" name="movie-rating"
min="0" max="10" step="0.1">
```

`<select>` Element

The HTML `<select>` element can be used to create a dropdown list. A list of choices for the dropdown list can be created using one or more `<option>` elements. By default, only one `<option>` can be selected at a time. The value of the selected `<select>`'s `name` and the `<option>`'s `value` attribute are sent as a key-value pair when the form is submitted.

```
<select name="rental-option">
  <option value="small">Small</option>
  <option value="family">Family
Sedan</option>
  <option value="lux">Luxury</option>
</select>
```

Submitting a Form

Once we have collected information in a form we can send that information somewhere else by using the `action` and `method` attribute. The `action` attribute tells the form to send the information. A URL is assigned that determines the recipient of the information. The `method` attribute tells the form what to do with that information once it's sent. An HTTP verb is assigned to the `method` attribute that determines the action to be performed.

```
<form action="/index3.html" method="PUT">
</form>
```

`<input>`: Text Type

HTML `<input>` elements can support text input by setting the attribute `type="text"`. This renders a single row input field that users can type text inside.
The value of the `<input>`'s `name` and `value` attribute of the element are sent as a key-value pair when the form is submitted.

```
<input type="text" name="username">
```

`<datalist>` Element

When using an HTML input, a basic search/autocomplete functionality can be achieved by pairing an `<input>` with a `<datalist>`. To pair a `<input>` with a `<datalist>` the `<input>`'s `list` value must match the value of the `id` of the `<datalist>`. The `datalist` element is used to store a list of `<option>`s.
The list of data is shown as a dropdown on an `input` field when a user clicks on the input field. As the user starts typing, the list will be updated to show elements that best match what has been typed into the input field. The actual list items are specified as multiple `option` elements nested inside the `datalist`.
`datalist`s are ideal when providing users a list of pre-defined options, but to also allow them to write alternative inputs as well.

```
<input list="ide">

<datalist id="ide">
  <option value="Visual Studio Code" />
  <option value="Atom" />
  <option value="Sublime Text" />
</datalist>
```

`<input>`: Radio Button Type

HTML `<input>` elements can be given a `type="radio"` attribute that renders a single radio button. Multiple radio buttons of a related topic are given the same `name` attribute value. Only a single option can be chosen from a group of radio buttons.
The value of the selected/checked `<input>`'s `name` and `value` attribute of this element are sent as a key-value pair when the form is submitted.

```
<input name="delivery_option" type="radio"
value="pickup" />
<input name="delivery_option" type="radio"
value="delivery" />
```

Submittable Input

HTML `<input>` elements can have a type attribute set to submit, by adding `type="submit"`. With this attribute included, a submit button will be rendered and, by default, will submit the `<form>` and execute its action. The text of a submit button is set to `Submit` by default but can also be changed by modifying the `value` attribute.

`<input>` `name` Attribute

In order for a form to send data, it needs to be able to put it into key-value pairs. This is achieved by setting the `name` attribute of the `input` element. The `name` will become the `key` and the `value` of the input will become the `value` the form submits corresponding to the key.
It's important to remember that the name is not the same as the ID in terms of form submission. The ID and the name of the input may be the same, but the value will only be submitted if the `name` attribute is specified.
In the code example, the first input will be submitted by the form, but the second one will not.

```
<input name="username" id="username" />
<input id="address" />
```

`<label>` Element

The HTML `<label>` element provides identification for a specific `<input>` based on matching values of the `<input>`'s `id` attribute and the `<label>`'s `for` attribute. By default, clicking on the `<label>` will focus the field of the related `<input>`.
The example code will create a text input field with the label text "Password: " next to it. Clicking on "Password: " on the page will focus the field for the related `<input>`.

```
<label for="password ">Password:</label>
<input type="text" id="password"
name="password">
```

`<input>` Password Type

The HTML `<input>` element can have the attribute `type="password"` that renders a single row input field which allows the user to type censored text inside the field. It is used to type in sensitive information.
The value of this `<input>`'s `name` and `value` (actual value and not the censored version) attribute of this element are sent as a key-value pair when the form is submitted.
The code block shows an example of the fields for a basic login form - the username and password fields.

```
<input type="text" name="username" />
<input type="password" name="password" />
```

`required` Attribute

In HTML, input fields have an attribute called `required` which specifies that the field must include a value.
The example code block shows an input field that is required. The attribute can be written as `required="true"` or simply `required`.

```
<input type="password" name="password"
required >
```

`max` Attribute

HTML `<input>` s of type `number` have an attribute called `max` that specifies the maximum value for the input field.
The code block shows an `input` number field that is set to have a maximum value of `20`. Any value larger than `20` will mark the input field as having an error.

```
<input type="number" max="20">
```

`maxlength` Attribute

In HTML, input fields with type `text` have an attribute called `maxlength` that specifies the maximum number of characters that can be entered into the field. The code block shows an input text field that accepts text that has a maximum length of 140 characters.

```
<input type="text" name="tweet"
maxlength="140">
```

`pattern` Attribute

In a `text` input element, the `pattern` attribute uses a regular expression to match against (or validate) the value of the `<input>`, when the form is submitted.

```
<form action="/action_page.php">
  Country code:
  <input type="text" name="country_code"
         pattern="[A-Za-z]{3}"
         title="Three letter country
code">
  <input type="submit">
</form>
```

`minlength` Attribute

In HTML, an input field of type `text` has an attribute that supports minimum length validation. To check that the input text has a minimum length, add the `minlength` attribute with the character count.
The example code block shows an example of a text field that has a minimum length of `6`.

```
<input type="text" name="username"
minlength="6" />
```

HTML Form Validators

HTML forms allow you to specify different kinds of validation for your input fields to make sure that data is entered correctly before being submitted. HTML supports a number of different validators, including things like minimum value, minimum/maximum length, etc. The validators are specified as attributes on the `input` field.

`min` Attribute

In HTML, input fields with type `number` have an attribute called `min` that specifies the minimum value that can be entered into the field. The code block provided shows an input number field that accepts a number with minimum value 1.

```
<input type="number" name="rating" min="1"
max="10">
```

0