**The DOM is a structural model of a web page that allows for scripting languages to access that page.**

**The system of organization in the DOM mimics the nesting structure of an HTML document.**

**Elements nested within another are referred to as the children of that element. The element they are nested within is called the parent element of those elements.**

**The DOM also allows access to the attributes of an HTML element such as style, id, etc.**

**With this understanding, you can begin to leverage the power of scripting languages to create, update, and maintain web pages!**

**SEE PICTURE**

**-------------------**

**The Document Keyword**

The *Document Object Model*, abbreviated DOM, is a powerful tree-like structure that organizes the elements on a web page and allows scripting languages to access them. This lesson will focus on some of the most useful methods and properties of the [DOM interface](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) in JavaScript. This interface is implemented by every modern browser.

First things first! The document object in JavaScript is the door to the DOM structure. The document object allows you to access the root node of the DOM tree. Before you can access a specific element in the page, first you must access the document structure itself. The document object allows scripts to access children of the DOM as properties.

For example, if you want to access the <body> element from your script, you can access it as a property of the document object by using document.body. This property will return the body element of that DOM.

Similarly, you could access the <title> element with the .title property. Here is a [comprehensive list](https://developer.mozilla.org/en-US/docs/Web/API/Document) of all document properties.

**Tweak an Element**

When using the DOM in your script to access an HTML element, whether it’s an <li> element or the entire <body> element, you also have access to all of that element’s properties.

This includes the ability to modify the contents of the element as well as its attributes and properties, which can range from modifying the text inside a <p> element to assigning a new background color to a <div>. For example, the .innerHTML property allows you to access and set the contents of an element.

Let’s take a look at how we can reassign the contents of the <body> element to the text 'The cat loves the dog':

document.body.innerHTML = 'The cat loves the dog.';

The .innerHTML property can also add any valid HTML elements. The following example replaces the contents of the <body> element by assigning an <h2> element as a child inside the <body> element:

document.body.innerHTML = '<h2>This is a heading</h2>';

**Instructions**

Checkpoint 1 Passed

**1.**

Use the .innerHTML property to modify the content of the <body> element to display an <h1> heading with the text 'This is now the heading of the body element'.

Notice how the previous content inside of the <body> element has been replaced!

document.body.innerHTML = "<h1>This is now the heading of the body element</h1>";

**Select and Modify Elements**

In the previous exercise, we accessed the <body> element with the document keyword!

What if we wanted to select a specific element besides the entire <body> element? The DOM interface allows us to access a specific element with CSS selectors.

*CSS selectors* define the elements to which a set of CSS rules apply, but we can also use these same selectors to access DOM elements with JavaScript! Selectors can include a tag name, a class, or an ID.

The .querySelector() method allows us to specify a CSS selector as a string and returns the first element that matches that selector. The following code would return the first paragraph in the document.

document.querySelector('p');

Along with .querySelector(), JavaScript has more targeted methods that select elements based on their class, id, or tag name.

For example, if you want to access an element directly by its id, you can use the aptly named .getElementById() method:

document.getElementById('bio').innerHTML = 'The description';

In this example, we’ve selected the element with an ID of 'bio' and set its .innerHTML to the text 'The description'. Notice that the ID is passed as a string, wrapped in quotation marks (' ').

There are also the .getElementsByClassName() and .getElementsByTagName() methods which return an array of elements, instead of just one element. You can use bracket notation to access individual elements of an array:

// Set first element of .student class as 'Not yet registered'  
document.getElementsByClassName('student')[0].innerHTML = 'Not yet registered';  
  
// Set second <li> tag as 'Cedric Diggory'  
document.getElementsByTagName('li')[1].innerHTML = 'Cedric Diggory`;

In the above example code, the first element with the 'student' class and the second <li> element have had their inner HTML changed.

**Instructions**

Checkpoint 1 Passed

**1.**

Use the .querySelector() method to select the first 'h1' element. Access that element’s .innerHTML property to change the h1 title to 'Most popular Harry Potter characters'.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Use the .getElementById() method to access the element with an ID of 'fourth'. Set that element’s inner HTML to read 'Professor Snape'.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

Use the .getElementsByClassName() method to access the first element with the class name of 'slytherin'. Replace that element’s inner HTML with the text 'Salazar Slytherin'.

Stuck? Get a hint

Checkpoint 4 Passed

**4.**

Use the .getElementsByTagName() method to access the first element with the <li> tag. Access that element’s .innerHTML to replace the content to read 'Dobby'.

document.querySelector("h1").innerHTML = "Most popular Harry Potter Characters";

document.getElementById("fourth").innerHTML = "Professor Snape";

document.getElementsByClassName("slytherin")[0].innerHTML = "Salazar Slytherin";

document.getElementsByTagName("li")[0].innerHTML = "Dobby";

**Style an Element**

Another way to modify an element is by changing its CSS style. The .style property of a DOM element provides access to the inline style of that HTML tag.

The syntax follows an element.style.property format, with the property representing a CSS property. For example, the following code selects the first element with a class of blue and assigns blue as the background-color:

let blueElement = document.querySelector('.blue');  
blueElement.style.backgroundColor = 'blue';

Unlike CSS, the DOM .style property does not implement a hyphen such as background-color, but rather camel case notation, backgroundColor. Check out this [W3 Reference on the HTML DOM style object](https://www.w3schools.com/jsref/dom_obj_style.asp) to see a list of how CSS properties are converted into JavaScript.

The following *chaining* syntax would also work:

document.querySelector('.blue').style.fontFamily = 'Roboto';

**Instructions**

Checkpoint 1 Passed

**1.**

Style the backgroundColor of the <body> element in the blog post to '#201F2E' to match the Codecademy text editor.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Next, change the font family of the element with the heading class to 'Roboto'.

document.body.style.backgroundColor = "#201F2E";

document.querySelector("h1").style.fontFamily = "Roboto";

**Traversing the DOM**

Let’s recap the parent and children relationship in the DOM hierarchy. A *parent node* is any node that is a direct ancestor of another node. A *child node* is a direct descendant of another node, called the parent node.

These relationships follow the nested structure of the HTML code. Elements nested within one HTML element are children of that parent element.

Each element has a .parentNode and .children property. The [.parentNode property](https://developer.mozilla.org/en-US/docs/Web/API/Node/parentNode) returns the parent of the specified element in the DOM hierarchy. Note that the document element is the *root node* so its .parentNode property will return null. The .children property returns an array of the specified element’s children. If the element does not have any children, it will return null.

<ul id='groceries'>  
  <li id='must-have'>Toilet Paper</li>  
  <li>Apples</li>  
  <li>Chocolate</li>  
  <li>Dumplings</li>  
</ul>

In the HTML code above, we have an <ul> element with the ID of groceries with four <li> elements inside.

let parentElement = document.getElementById('must-have').parentNode; // returns <ul> element  
let childElements = document.getElementById('groceries').children; // returns an array of <li> elements

Here, the parentElement variable stores the .parentNode of the <li> element with the ID of must-have, which will be the <ul> element with the ID of groceries. The childElements variable is set to the children of the <ul> element with the ID of groceries, which will be an array of four <li> elements.

**Instructions**

Checkpoint 1 Passed

**1.**

First, create a variable called first and set it to the first child of the document body.

Then, set the inner HTML of the first element to 'BROWN BEARS ARE AWESOME!'. Take a moment to note which element has been modified.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Use the .parentNode property to access the parent element of the variable first and modify its .style.backgroundColor to 'beige'.

Take a moment to notice the change in the web page.

let first = document.body.children[0];

first.innerHTML = "BROWN BEARS ARE AWESOME!";

first.parentNode.style.backgroundColor = "beige";

**Create and Insert Elements**

Just as the DOM allows scripts to modify existing elements, it also allows for the creation of new ones.

The [.createElement() method](https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement) creates a new element based on the specified tag name passed into it as an argument. However, it does not append it to the document. It creates an empty element with no inner HTML.

let paragraph = document.createElement('p');

In the example code above, the .createElement() method takes 'p' as its argument which creates an empty <p> element and stores it as the paragraph variable.

We can assign values to the properties of the newly created element like how we’ve done previously with existing elements.

paragraph.id = 'info';   
paragraph.innerHTML = 'The text inside the paragraph';

Above, we use the .id property to assign 'info' as ID and the .innerHTML property to set 'The text inside the paragraph' as the content of the <p> element.

In order to create an element and add it to the web page, you must assign it to be the child of an element that already exists on the DOM, referred to as the parent element. We call this process *appending*. The .appendChild() method will add a child element as the parent element’s last child node. The following code appends the <p> element stored in the paragraph variable to the document body.

document.body.appendChild(paragraph);

The .appendChild() method does not replace the content inside of the parent, in this case, body. Rather, it appends the new element as the last child of that parent.

**Instructions**

Checkpoint 1 Passed

**1.**

Create a <li> element using the .createElement() method and save it in a variable called newAttraction.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

On the following line, assign the newAttraction element an id of 'vespa'.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

Next, assign the newAttraction element the text 'Rent a Vespa' as its inner HTML.

Stuck? Get a hint

Checkpoint 4 Passed

**4.**

Lastly, append the newAttraction element to the list of top attractions with the ID of italy-attractions.

let newAttraction = document.createElement("li");

newAttraction.id = "vespa";

newAttraction.innerHTML = "Rent a Vespa";

document.getElementById("italy-attractions").appendChild(newAttraction);

**Remove an Element**

In addition to modifying or creating an element from scratch, the DOM also allows for the removal of an element. The .removeChild() method removes a specified child from a parent.

let paragraph = document.querySelector('p');  
document.body.removeChild(paragraph);

In the above example code, the .querySelector() method returns the first paragraph in the document. Then, the paragraph element is passed as an argument of the .removeChild() method chained to the parent of the paragraph—document.body. This removes the first paragraph from the document body.

If you want to hide an element rather than completely deleting it, the [.hidden property](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/hidden) allows you to hide it by setting the property as true or false:

document.getElementById('sign').hidden = true;

The code above did not remove the element with ID of 'sign' from the DOM but rather, hid it.

**Instructions**

Checkpoint 1 Passed

**1.**

First, save the element with the ID of vespa as a variable named elementToRemove.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

The elementToRemove element is a child of the list of top attractions with the ID of italy-attractions.

Remove the elementToRemove element from its parent.

let elementToRemove = document.getElementById("vespa");

elementToRemove.remove();

**Add Click Interactivity**

You can add interactivity to DOM elements by assigning a function to run based on an [event](https://developer.mozilla.org/en-US/docs/Web/Events). Events can include anything from a click to a user mousing over an element. We will learn more about events in the upcoming [DOM Events with JavaScript](https://www.codecademy.com/courses/build-interactive-websites/lessons/dom-events) lesson. For now, let’s take a look at how to modify an element when a click event happens.

The .onclick property allows you to assign a function to run on when a click event happens on an element:

let element = document.querySelector('button');  
  
element.onclick = function() {   
  element.style.backgroundColor = 'blue'   
};

You can also assign the .onclick property to a function by name:

let element = document.querySelector('button');  
  
function turnBlue() {  
   element.style.backgroundColor = 'blue';  
}  
  
element.onclick = turnBlue;

In the above example code, when the <button> element detects a click event, the backgroundColor will change to 'blue'.

**Instructions**

Checkpoint 1 Passed

**1.**

Inside the turnButtonRed() function, add the code to modify the button stored in the element variable as follows:

1. Assign the .style.backgroundColor to 'red'
2. Assign the .style.color to 'white'
3. Modify the .innerHTML to 'Red Button'

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Below our turnButtonRed() function definition, trigger the turnButtonRed() function when the element detects a click event.

let element = document.querySelector('button');

function turnButtonRed(){

  // Add code to turn button red

element.style.backgroundColor = "red";

element.style.color = 'white';

element.innerHTML = "Red Button";

}

element.onclick = turnButtonRed;

### Review

In this lesson, you manipulated a webpage structure by leveraging the Document Object Model (DOM) interface in JavaScript.

Let’s review what we learned:

* The document keyword grants access to the root of the DOM in JavaScript.
* The DOM Interface allows you to select a specific element with CSS selectors by using the .querySelector() method.
* You can access an element directly by its ID with the .getElementById() method which returns a single element.
* You can access an array of elements with the .getElementsByClassName() and .getElementsByTagName() methods, then call a single element by referencing its placement in the array.
* The .innerHTML and .style properties allow you to modify an element by changing its contents or style respectively.
* You can create, append, and remove elements by using the .createElement(),.appendChild() and .removeChild() methods respectively.
* The .onclick property can add interactivity to a DOM element based on a click event.
* The .children property returns a list of an element’s children and the .parentNode property returns the element’s closest connected node in the direction towards the root.

### What is an Event?

When you refresh your email inbox, double tap on a post, or scroll through a newsfeed — something cool happens in your browser. These actions are known as events!

Events on the web are user interactions and browser manipulations that you can program to trigger functionality. Some other examples of events are:

* A mouse clicking on a button
* Webpage files loading in the browser
* A user swiping right on an image

When a user does any of the above actions, they’re causing the event to be fired or be triggered. As in, “a click event fired when the button was clicked”. Being able to respond to these events makes your website interactive and therefore dynamic.

### "Firing" Events

After a specific event fires on a specific element in the [document object model](https://www.codecademy.com/paths/web-development/tracks/build-interactive-websites/modules/web-dev-interactive-websites/lessons/intro-dom/exercises/what-is-the-dom) (or DOM), an event handler function can be created to run as a response. In this lesson, we will learn about event handler functions that modify and update DOM elements after an event fires.

Let’s compare the firing of events to something more familiar: a dog trained to eat when they hear the sound of a bell! (This is known as the [Pavlovian conditioning](https://en.wikipedia.org/wiki/Classical_conditioning).)

As you can see in the diagram, the ringing of the bell is analogous to a JavaScript event firing. The dog is trained to anticipate the ringing of the bell and this action is analogous to creating an event listener. After the dog hears the bell, it’ll come over and eat its food! This response is like an event handler function that executes code which, in a website, could change an element’s color, text, and much more!”

Most events in the browser take place without being noticed — the events are undetected because there is no event handler associated with the event to execute an action. Event handlers are crucial for creating interactive websites with JavaScript.

**Event Handler Registration**

You’re doing great! Now it’s time to dive into using event handler functions to create interactivity.

Using the .addEventListener() method, we can have a DOM element listen for a specific event and execute a block of code when the event is detected. The DOM element that listens for an event is called the *event target* and the block of code that runs when the event happens is called the *event handler*.

Let’s take a look at the code below:

let eventTarget = document.getElementById('targetElement');  
  
eventTarget.addEventListener('click', function() {  
  // this block of code will run when click event happens on eventTarget element  
});

Let’s break this down!

* We selected our event target from the DOM using document.getElementById('targetElement').
* We used the .addEventListener() method on the eventTarget DOM element.
* The .addEventListener() method takes two arguments: an event name in *string* format and an event handler function. We will learn about different values we can use as event names in a later lesson.
* In this example, we used the 'click' event, which fires when the user clicks on eventTarget.
* The code block in the event handler function will execute when the 'click' event is detected.

Instead of using an anonymous function as the event handler, it’s best practice to create a named event handler function. Your code will remain organized and reusable this way, even if your code gets complex. Check out the syntax:

function eventHandlerFunction() {  
  // this block of code will run when click event happens  
}  
  
eventTarget.addEventListener('click', eventHandlerFunction);

The named function eventHandlerFunction is passed as the second argument of the .addEventListener() method instead of defining an anonymous function within the method!

**Instructions**

Checkpoint 1 Passed

**1.**

Look at the browser and notice that there are two elements, one containing informational text about JavaScript and a button. When the button is clicked, there should be more text that appears. Currently, clicking the button doesn’t seem to do anything. You are going to create an event handler to fix this!

First, create a function called showInfo() which we will use as the event handler function to make the hidden element containing the additional informational text (moreInfo) appear when the 'click' event fires.

Inside the function, create a statement that changes the .display style property of the moreInfo element to 'block'.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Now, create an event handler for a click event using .addEventListener(). Use readMore as the event target and showInfo as the event handler function.

Run your code and fire your event once you’re done.

let readMore = document.getElementById('read-more');

let moreInfo = document.getElementById('more-info');

// Write your code here:

const showInfo = () =>{

  moreInfo.style.display = "block";

};

readMore.addEventListener("click", showInfo);

**Adding Event Handlers**

We looked at registering event handlers using the .addEventListener() method, but there is also another way!

Event Handlers can also be registered by setting an .onevent property on a DOM element (event target). The pattern for registering a specific event is to append an element with .on followed by the lowercased event type name. For instance, if we want to register a click event with this pattern, we would write:

eventTarget.onclick = eventHandlerFunction;

Here, we give the DOM element eventTarget the [.onclick property](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onclick) and set its value as the event handler function eventHandlerFunction.

It’s important to know that this .onevent property and .addEventListener() will both register event listeners. With .onevent, it allows for one event handler function to be attached to the event target. However, with the .addEventListener() method , we can add multiple event handler functions. In the later exercises, we’ll be using the .addEventListener() syntax, but we wanted to also introduce the .onevent syntax because both are widely used.

**Instructions**

Checkpoint 1 Passed

**1.**

Play around with the view and close elements to meet the Codecademy mascot, Codey. Codey is super happy you made it this far and they need your help!

Right now, the open() function makes the codey and close elements visible by changing their .display properties to 'block'. The hide() function hides the same elements by assigning a 'none' value to the .display properties.

Codey wants you to create a function named textChange() that changes the text in the view element to 'Hello, World!'.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Next, you must create a function named textReturn() that changes the text of the view element variable back to 'View'.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

Assign textChange as an event handler function to a click event fired on the view variable and run your code.

Stuck? Get a hint

Checkpoint 4 Passed

**4.**

Assign textReturn as an event handler function to a click event fired on the close variable. Then run your code and fire the events!

let view = document.getElementById('view-button');

let close = document.getElementById('close-button');

let codey = document.getElementById('codey');

let open = function() {

  codey.style.display = 'block';

  close.style.display = 'block';

};

let hide = function() {

  codey.style.display = 'none';

  close.style.display = 'none';

};

view.addEventListener('click', open);

close.addEventListener('click', hide);

// Write your code here

const textChange = () => {

  view.innerHTML = "Hello, World";

};

const textReturn = () => {

  view.innerHTML = "View";

};

view.addEventListener("click", textChange);

close.addEventListener("click", textReturn);

**Removing Event Handlers**

The .removeEventListener() method is used to reverse the .addEventListener() method. This method stops the event target from “listening” for an event to fire when it no longer needs to. .removeEventListener() also takes two arguments:

1. The event type as a string
2. The event handler function

Check out the syntax of a .removeEventListener() method with a click event:

eventTarget.removeEventListener('click', eventHandlerFunction);

Because there can be multiple event handler functions associated with a particular event, .removeEventListener() needs both the exact event type name and the name of the event handler you want to remove. If .addEventListener() was provided an anonymous function, then that event listener cannot be removed.

**Instructions**

Checkpoint 1 Passed

**1.**

Check out this website that shows your daily fortune. But the website is broken! You should only be able to see your daily fortune once, but you will see that you can keep pressing the button to get a new fortune. You need to use the .removeEventListener() method after a fortune is displayed.

Inside the showFortune() function, add a .removeEventListener() to prevent a new fortune from being displayed when a user tries to click the button element.

let fortunes = ["A beautiful, smart, and loving person will be coming into your life.",

  "A fresh start will put you on your way.",

  "A golden egg of opportunity falls into your lap this month.",

  "A smile is your personal welcome mat.",

  "All your hard work will soon pay off."

]

let button = document.getElementById('fortuneButton');

let fortune = document.getElementById('fortune');

function fortuneSelector(){

  let randomFortune = Math.floor(Math.random() \* fortunes.length);

  return fortunes[randomFortune];

}

function showFortune(){

  fortune.innerHTML = fortuneSelector();

  button.innerHTML = "Come back tomorrow!";

  button.style.cursor = "default";

  button.removeEventListener("click", showFortune);

  //add your code here

}

button.addEventListener('click', showFortune);

**Event Object Properties**

JavaScript stores events as [Event objects](https://developer.mozilla.org/en-US/docs/Web/API/Event) with their related data and functionalities as properties and methods. When an event is triggered, the event object can be passed as an argument to the event handler function.

function eventHandlerFunction(event){  
   console.log(event.timeStamp);  
}  
  
eventTarget.addEventListener('click', eventHandlerFunction);

In the example above, when the 'click' event is triggered on the eventTarget, the eventHandlerFunction receives event, the Event object, which has information related to the 'click' event. Then, we log the time it took for the event to be triggered since the document was loaded by accessing the .timeStamp property of the event object.

There are pre-determined properties associated with event objects. You can call these properties to see information about the event, for example:

* the [.target property](https://developer.mozilla.org/en-US/docs/Web/API/Event/target) to reference the element that the event is registered to.
* the [.type property](https://developer.mozilla.org/en-US/docs/Web/API/Event/type) to access the name of the event.
* the [.timeStamp property](https://developer.mozilla.org/en-US/docs/Web/API/Event/timeStamp) to access the number of milliseconds that passed since the document loaded and the event was triggered.

**Instructions**

Checkpoint 1 Passed

**1.**

Everyone loves a good puppy picture! Use what you’ve learned about event object properties to share this puppy photo with your friends.

First, add a statement inside the sharePhoto() function that will change the .display property of the event .target to 'none'.

Remember that the event .target is a DOM element and you can access the .display property by referencing the .style property first.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Next, add a statement to the sharePhoto function that will modify the text element to state the number of milliseconds from the DOM loading to the event firing.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

Now, to create functionality for the event object, assign the sharePhoto function to a click event on the share element.

Run your code!

let social = document.getElementById('social-media');

let share = document.getElementById('share-button');

let text = document.getElementById('text');

// Write your code below

let sharePhoto = function(event) {

  event.target.style.display = "none";

  text.innerHTML = event.timeStamp;

};

share.addEventListener("click", sharePhoto);

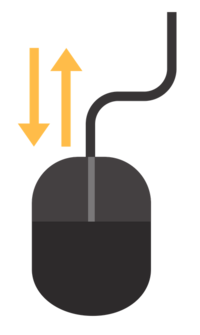
**Event Types**

Beyond the click event, there are all types of DOM events that can fire in a browser! It’s important to know *most* events in the DOM take place without being noticed because there are no event handlers connected to them.

It’s also important to know some registered events don’t depend on user interactions to fire. For instance, the load event fires after website files completely load in the browser.

Browsers can fire many other events without a user — you can check out a list of events on the [MDN Events Reference](https://developer.mozilla.org/en-US/docs/Web/Events) page.

Many events need user interaction with the DOM to fire. One user interaction event you’ve become familiar with is the click event. A click event fires when the user presses and releases a mouse button on an element in the DOM.



In the rest of this lesson, you’ll explore more user interaction event types like the mouse and keyboard events. To explore more event types, check out the [MDN Events Reference](https://developer.mozilla.org/en-US/docs/Web/Events).

**Instructions**

Checkpoint 1 Passed

**1.**

We made a really cool color generator to help people find different colors — try it out! Uh oh, it seems to be broken. We need you to use your new knowledge to fix the website.

Complete the colorChange() function, which will be used as an event handler function, to randomly change the colors of the buttons when specific events are fired on them.

First, add the following variable to the event handler function:

let randomColor = 'rgb(' + colorValue() + ',' + colorValue() + ',' + colorValue() + ')';

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Add a statement that changes the background color of the event target and set it equal to randomColor.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

There are two elements that should change color on this web page. First, create an event handler property on the button element that fires when it’s clicked. Use colorChange as the event handler function.

Then run your code and fire the click event.

Stuck? Get a hint

Checkpoint 4 Passed

**4.**

Next, create an event handler property on the mysteryButton element. This property needs an event that fires when you rotate the mouse wheel or slide down on the mousepad. Use the [MDN Events Reference](https://developer.mozilla.org/en-US/docs/Web/Events#Mouse_Events) page to find the correct event type. Assign the same colorChange event handler function to the event handler property.

Then run your code and fire the event.

// This variable stores the "Pick a Color" button

let button = document.getElementById('color-button');

// This variable stores the "Mystery Color" button

let mysteryButton = document.getElementById('next-button');

// This random number function will create color codes for the randomColor variable

function colorValue() {

  return Math.floor(Math.random() \* 256);

}

function colorChange(event){

  let randomColor = 'rgb(' + colorValue() + ',' + colorValue() + ',' + colorValue() + ')';

  event.target.style.backgroundColor = randomColor;

}

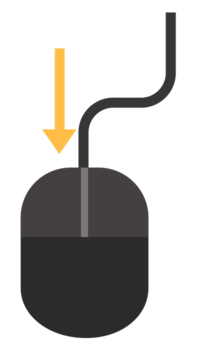
button.addEventListener("click", colorChange);

mysteryButton.addEventListener("wheel", colorChange, {once: true});

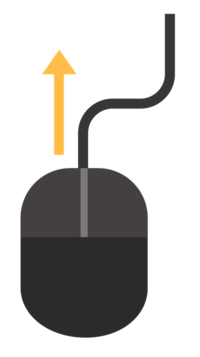
**Mouse Events**

When you use a mouse device on a website, *mouse events* fire. You’ve seen the click and wheel events, but, there are even more mouse events to explore!

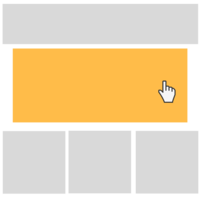
The mousedown event is fired when the user presses a mouse button down. This is different from a click event because mousedown doesn’t need the mouse button to be released to fire.



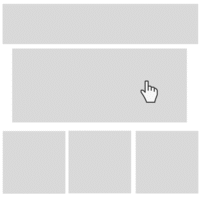
The mouseup event is fired when the user releases the mouse button. This is different from the click and mousedown events because mouseup doesn’t depend on the mouse button being pressed down to fire.



The mouseover event is fired when the mouse enters the content of an element.



The mouseout event is fired when the mouse leaves an element.



**Instructions**

Checkpoint 1 Passed

**1.**

In this exercise, you’ll modify the list elements using mouse events. You can use the reset element that is already programmed to set the other list element back to their default styles.

First, create a function called increaseWidth() that changes the .width of itemOne to a size of '401px' or greater.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Now, create an event handler for itemOne that will trigger the increaseWidth() function when the mouse hovers on it.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

Next, create a function called changeBackground() that changes the .backgroundColor of itemTwo.

Stuck? Get a hint

Checkpoint 4 Passed

**4.**

Let’s use the changeBackground() function we just created as an event handler for itemTwo that will be triggered when the mouse is released over the element!

Stuck? Get a hint

Checkpoint 5 Passed

**5.**

Now, create a function called changeText() that changes the text of itemThree to 'The mouse has left the element'.

Stuck? Get a hint

Checkpoint 6 Passed

**6.**

Create an event handler for itemThree that will fire changeText() function when the mouse leaves the element.

Stuck? Get a hint

Checkpoint 7 Passed

**7.**

Finally, let’s create a function called showItem() that makes the itemFive element appear by changing the .display style property to 'block'.

Stuck? Get a hint

Checkpoint 8 Passed

**8.**

Now, create an event handler for itemFour that triggers the showItem() function when the mouse is pressed down on the element.

// These variables store the boxes on the side

let itemOne = document.getElementById('list-item-one');

let itemTwo = document.getElementById('list-item-two');

let itemThree = document.getElementById('list-item-three');

let itemFour = document.getElementById('list-item-four');

let itemFive = document.getElementById('list-item-five');

let resetButton = document.getElementById('reset-button');

// This function programs the "Reset" button to return the boxes to their default styles

let reset = function() {

  itemOne.style.width = ''

  itemTwo.style.backgroundColor = ''

  itemThree.innerHTML = 'The mouse must leave the box to change the text'

  itemFive.style.display = "none"

};

resetButton.onclick = reset;

// Write your code here

const increaseWidth = () => {

  itemOne.style.width = "500px";

};

itemOne.addEventListener("mouseover", increaseWidth);

const changeBackground = () =>{

  itemTwo.style.backgroundColor = "white";

};

itemTwo.addEventListener("mouseup", changeBackground);

const changeText = () =>{

  itemThree.innerHTML = "The mouse has left the element";

};

itemThree.addEventListener("mouseout", changeText);

const showItem = () =>{

  itemFive.style.display = "block";

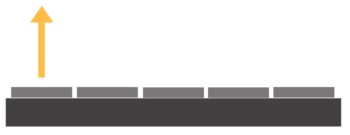
};

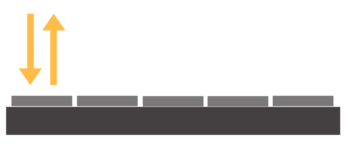
itemFour.addEventListener("mousedown", showItem);

**Keyboard Events**

Other popular types of events are keyboard events! *keyboard events* are triggered by user interaction with keyboard keys in the browser.

The keydown event is fired while a user presses a key down.

The keyup event is fired while a user releases a key.

The keypress event is fired when a user presses a key down and releases it. This is different from using keydown and keyup events together, because those are two complete events and keypress is one complete event.

Keyboard events have unique properties assigned to their event objects like the .key property that stores the values of the key pressed by the user. You can program the event handler function to react to a specific key, or react to any interaction with the keyboard.

**Instructions**

Checkpoint 1 Passed

**1.**

Now it’s time to create a game! Program this code to dribble the ball on the platform using any key on a keyboard. When a user presses a key down, it should lift the ball up. When the user releases the key, the ball should drop.

First, make a function named up() that will raise the .bottom position of the ball to '250px'.

Stuck? Get a hint

Checkpoint 2 Passed

**2.**

Next, make a function named down() that will lower the .bottom position of the ball to '50px'.

Stuck? Get a hint

Checkpoint 3 Passed

**3.**

Create an event handler property that runs the up() function when a keydown event fires on the document object, or anywhere on the DOM, as the event target.

Stuck? Get a hint

Checkpoint 4 Passed

**4.**

Create an event handler property that runs the down() function when a keyup event fires on the document.

Run your code and play around with the keyboard events to make the ball bounce on the platform. You can try keys like the space bar, letter keys, or number keys!

let ball = document.getElementById('float-circle');

// Write your code below

const up = () =>{

  ball.style.bottom = "250px";

};

const down = () =>{

  ball.style.bottom = "50px";

};

document.addEventListener("keydown", up);

document.addEventListener("keyup", down);

### Review

Congrats, you completed the lesson! Now you’ve learned about JavaScript events and you can leverage these events on the DOM to create interactivity with event handlers.

Let’s review what you’ve learned:

* You can register events to DOM elements using the addEventListener() method.
* The addEventListener() method takes two arguments: an event type and an event handler function.
* When an event is triggered on the event target, the registered event handler function executes.
* Event handler functions can also be registered as values of onevent properties of their event target.
* Event object properties like .target, .type, and .timeStamp are used to provide information about the event.
* The addEventListener() method can be used to add multiple event handler functions to a single event.
* The removeEventListener() method stops specific event handlers from “listening” for specific events firing.