**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 examples of events are:

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

You can create interactivity on a website by assigning a function to respond to a specific event *firing*, or triggering. In this lesson, you’ll learn to use JavaScript with events to create dynamic websites.

**Instructions**

Check out the recipe webpage — you can play around with the events and their specific functions!

1. Increase the measurements of the ingredients by clicking the + button.
2. Hover over the instructions to see the estimated cook time.
3. Click on highlighted ingredients in the instructions for facts about the foods in the ingredient list.
4. Click the bar at the bottom to change its background color.

# "Firing" Events

After a specific event fires on a specific element in the [document object model](https://www.codecademy.com/paths/web-development/tracks/building-interactive-javascript-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 this to something more familiar: a cuckoo clock is an example of programming with events! When the clock hands strike a new hour, the cuckoo bird responds with a whistle for each hour. For example, the cuckoo bird will whistle twice when the clock strikes 2 o’clock.

As you can see in the diagram, the clock striking an hour is the specific event that causes a specific response from the cuckoo bird. Event handler functions wait for their specific events to fire like the cuckoo bird in the clock awaiting the next hour. These functions can be used to change a DOM 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.

JavaScript interprets registered events as *event objects* with properties and methods. When you create an event handler function on a specific event, you create a property of the event object.

An event handler function is registered as a property attached to the DOM element being interacted with, or the *event target*. Check out the syntax:

let eventTarget = document.getElementById('targetElement');

eventTarget.onclick = function() {

// this block of code will run

}

Let’s break this down!

1. First, we accessed the DOM element that serves as the event target by its ID using document.getElementById('targetElement').
2. Then we created the event handler property which consists of the event target followed by the event name (the prefix on- and the event type.) In this example, we’re using the click event which fires when the user presses and releases a mouse button on a DOM element.
3. Lastly, we assigned an event handler function to the property.

Event handlers can also be registered as an HTML element attribute, but you should always avoid inline JavaScript code in HTML files!

**Instructions**

**1.**

Check out the excerpt about JavaScript. There is more information, or moreInfo, to be read when the user clicks the readMore element, but it does not appear to be working. You are going to create an event handler to fix this!

First, create an event handler property for a click event that uses the readMore button as the event target.

Hint

This event handler will be an anonymous function, and the syntax for that is:

eventTarget.onevent = function() {

...

}

**2.**

Now, you need to make the moreInfo element appear when the click event fires. Create a statement that changes the .display style property of the moreInfo element to 'block'.

Once you’ve completed this run your code and fire your event!

Hint

1. Change the CSS property of the moreInfo element from a display of none to block. To change a CSS property in JavaScript, you must use the following syntax:

element.style.property = 'value';

1. Remember the property value **must** be in quotations!

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

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

// Write your code here:

readMore.onclick = function() {

  moreInfo.style.display = 'block';

}

**Adding Event Handlers**

It’s best practice to create named event handler functions, instead of anonymous functions. Your code will remain organized and reusable this way, even if your code gets complex. Check out the syntax:

let eventHandlerFunction = function() {

// this block of code will run

}

eventTarget.onclick = eventHandlerFunction;

The .addEventListener() method is another common syntax for registering event handlers. An *event listener* waits for a specific event to occur and calls a named event handler function to respond to it. This method requires two arguments:

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

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

eventTarget.addEventListener('click', eventHandlerFunction);

You’ll want to use the .addEventListener() method to allow multiple event handlers to be registered to a single event without changing its other event handlers.

**Instructions**

**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 she needs your help!

Right now, the open function makes the codey and the 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 that changes the text in the view element to 'Hello, World!'. Call this function textChange.

Hint

To change the content of an element in JavaScript use the .innerHTML property:

element.innerHTML = 'This is new text content.';

**2.**

Next, you must create a function named textReturn that changes the text of the view variable back to the default.

Hint

The original text for the view element was View — you’ll probably need to call the .innerHTML property to change this back.

**3.**

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

Hint

Because the view element already has an event handler property, you need to use a .addEventListener() method to add more event handlers.

**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!

Hint

Because the close element already has an event handler property, you need to use a .addEventListener() method to add more event handlers.

// Add the code you want to test below:

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.onclick = open;

close.onclick = hide;

// Write your code here

let textChange = function() {

  view.innerHTML = 'Hello, World!';

}

let textReturn = function() {

  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 code from “listening” for an event to fire when it no longer needs to. .removeEventListener also passes 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 this method unregisters event handlers, it needs to identify which function to remove from the event. The event handler function passed to the .removeEventListener() method **must** be the same function of the corresponding .addEventListener().

**Instructions**

**1.**

This digital restaurant needs your help because the lock on the door is broken! Click on the lock and unlock elements and you will see that people are able to get in even when the restaurant is closed. They need you to use .removeEventListener() to keep the door locked.

First, you must add an event listener to the lock element when a click event is fired with an anonymous function.

Inside the function, add a .removeEventListener() to turn off the openDoor function when a user tries to click the door element. Then run your code and fire the event to test out your event handlers.

Hint

You will need to use an anonymous function to create a nested function like so,

eventTarget.addEventListener('event', function() {

target.removeEventListener('event', eventHandlerFunction);

});

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

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

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

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

let cafeImage = document.getElementById('image');

cafeImage.hidden = true;

let openDoor = function() {

  door.hidden = true;

  cafeImage.hidden = false;

}

let closeDoor = function(){

  door.hidden = false;

  cafeImage.hidden = true;

}

unlock.onclick = function() {

  sign.innerHTML = 'OPEN';

  unlock.style.backgroundColor = '#6400e4';

  lock.style.backgroundColor = 'lightgray';

}

lock.onclick = function() {

  sign.innerHTML = 'CLOSED';

  lock.style.backgroundColor = '#6400e4';

  unlock.style.backgroundColor = 'lightgray';

}

unlock.addEventListener('click', function(){

  door.addEventListener('click', openDoor);

  cafeImage.addEventListener('click', closeDoor);

})

// Write your code here

lock.addEventListener('click', function(){

  door.removeEventListener('click', openDoor)

})

**Event Object Properties**

JavaScript stores events as *event objects* with their related data and functionality as properties and methods. 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 to access the element that triggered the event.
* the .type property to access the name of the event.
* the .timeStamp property to access the number of milliseconds that passed since the document loaded and the event was triggered.

**Instructions**

**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 to the sharePhoto function that will change the display property of the event .target to 'none'.

Hint

You’ll need to use the .target property on an event object to access the event target, like so event.target.

**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.

Hint

Use the .timeStamp property on an event object to access the number of milliseconds between the DOM loading and the event firing, like so event.timeStamp.

**3.**

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

Run your code!

Hint

Use the .onclick property on the share element to assign the sharePhoto event handler.

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 = 'You share the puppy in '

  + event.timeStamp + ' ms.';

}

share.onclick = sharePhoto;

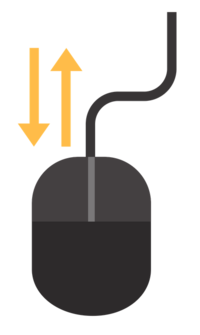
# Event Types

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**

**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 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(' + rgb(255) + ',' + rgb(255) + ',' + rgb(255) + ')';

Hint

The randomColor variable should be the first statement inside your function.

**2.**

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

Hint

To change the value of hyphenated style properties in JavaScript you must use camel case. For example, the font-size property will become fontSize.

**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

**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.

Hint

Stuck? Check out this [MDN documentation](https://developer.mozilla.org/en-US/docs/Web/Events/wheel) on wheel events. So you’ll want to use [the onwheel event handler](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onwheel).

// 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 rgb(num) {

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

}

// Write your code below

let colorChange = function (event) {

  let randomColor = 'rgb(' + rgb(255) + ',' + rgb(255) + ',' + rgb(255) + ')';

  event.target.style.backgroundColor = randomColor;

}

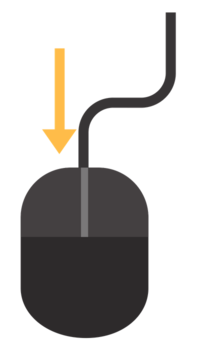
button.onclick = colorChange;

mysteryButton.onwheel = colorChange;

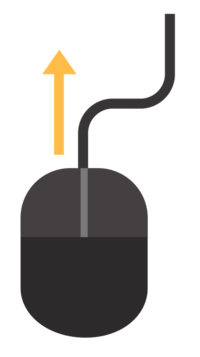
# Mouse Events

When you click down or move the mouse device mouse events fire. Outside of the click event, there are other events that can be fired when using a mouse.

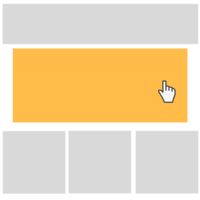
The mousedown event is fired when the user presses a mouse button down. This is different from a click event because mousedown doesn’t need the mouse button to be released to fire.



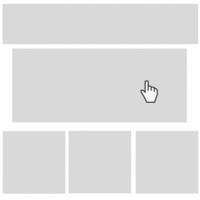
The mouseup event is fired when the user releases the mouse button. This is different from the click and mousedown events because mouseup doesn’t depend on the mouse button being pressed down to fire.



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**

**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 an event handler property on itemOne when the mouse hovers over it.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent

**2.**

Now, assign an anonymous event handler function that changes the width of itemOne to any size greater or less than 400px.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent = function() {

...

}

**3.**

Now, create an event handler property on itemTwo when the mouse is released over the element.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent

**4.**

Create an event handler function that changes the background color of itemTwo when the mouse is released over the element.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent = function() {

...

}

**5.**

Next, create an event handler property that fires when the mouse leaves the itemThree element.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent

**6.**

Create an anonymous event handler function that changes the text of itemThree to 'The mouse has left the element.'.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent = function() {

...

}

**7.**

Finally, create an event handler property that fires when the mouse is pressed down on itemFour.

Hint

This can be an anonymous function with syntax like this:

eventTarget.onevent

**8.**

Create an event handler function that makes the itemFive appear when the mouse is pressed down on itemFour.

Hint

Change the style property of the element from a display of none to block. To change a CSS property in JavaScript, you must use the following syntax:

element.style.property = 'value';

// 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 code for the first list item

itemOne.onmouseover = function(){

  itemOne.style.width = '430px';

};

// Write code for the second list item

itemTwo.onmouseup = function(){

  itemTwo.style.backgroundColor = 'green';

};

// Write code for the third list item

itemThree.onmouseout = function(){

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

};

// Write code for the fourth list item

itemFour.onmousedown = function(){

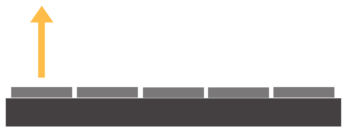
  itemFive.style.display = 'block';

}

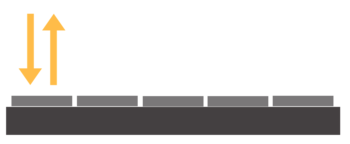
# Keyboard Events

Another popular type of event is the keyboard event! 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**

**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' above the platform element.

Hint

The bottom style property stores the vertical position of an element. It should be assigned to a value of '250px' as a string, not a number.

**2.**

Next, make a function named down that will run when the ball drops to the platform element. This function should change the bottom position of the ball to '50px'.

Hint

The bottom style property should be assigned to '50px' as a string, not a number.

**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.

Hint

The document represents the entire document object model — using the document as the event target allows you to fire the event anywhere on the DOM.

**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!

Hint

The document represents the entire document object model — using the document as the event target allows you to fire the event anywhere on the DOM.

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

// Write your code below

let up = function() {

  ball.style.bottom = '250px';

};

let down = function() {

  ball.style.bottom = '50px';

};

document.onkeydown = up;

document.onkeyup = 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:

* JavaScript engines register events as objects with properties and methods associated with them.
* Event handlers are registered as properties of their event object.
* 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.