Set 24: Event Handlers

Skill 24.01: Explain the purpose of an event handler

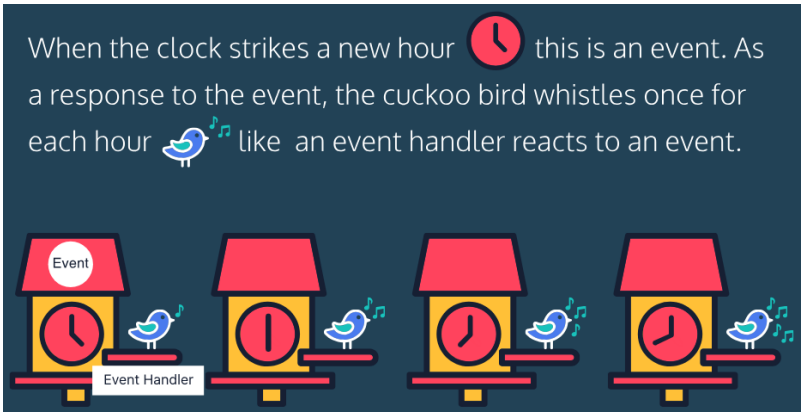
Skill 24.01 Concepts

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

As an example, consider a cuckoo clock depicted below. 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!



In this lesson, you'll learn to use JavaScript with events to create dynamic websites.

Skill 24.01 Exercise 1

Skill 24.02: Register an event handler

Skill 24.02 Concepts

HTML events are "things" that happen to HTML elements. Using JavaScript you can write functions to handle these interactions.

When an event handler function is created, it is attached to the DOM element being interacted with, or the *event target*.

Check out the syntax:

```
var myButton = document.createElement("button");
myButton.innerHTML = "Click Me!";
document.body.append(myButton);

myButton.onclick = clickFunction;

function clickFunction(){
    myButton.innerHTML = "You clicked me!";
}
```

Let's break the code above,

1. First we created a button element and assigned it the variable *myButton*.
2. Next, we appended the button to the body of the html document.
3. Then we created the event handler property which consists of the event target (*myButton*) 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 the mouse on a DOM element, *myButton.*
4. Lastly, we assigned an event handler function, *clickFunction* to the property.

Skill 24.02 Exercise 1

Skill 24.03: Register multiple handlers to a single event

Skill 24.03 Concepts

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 clickevent:

```
eventTarget.addEventListener('click', eventHandlerFunction);
```

The *.addEventListener()* method can be applied to allow multiple event handlers to be registered to a single event without changing its other event handlers.

In the below example, two click events have been registered to the body of the HTML page. When it is clicked, the text is changed and the background color is changed.

```
document.body.addEventListener("click", changeText);
document.body.addEventListener("click", changeBackground);

function changeText(){
    document.body.innerHTML = "The text has changed!";
}

function changeBackground(){
    document.body.style.backgroundColor = "pink";
}
```

Skill 24.03 Exercise 1

Skill 24.04 Remove an event handler

Skill 24.04 Concepts

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 clickevent:

```
eventTarget.removeEventListener('click', eventHandlerFunction);
```

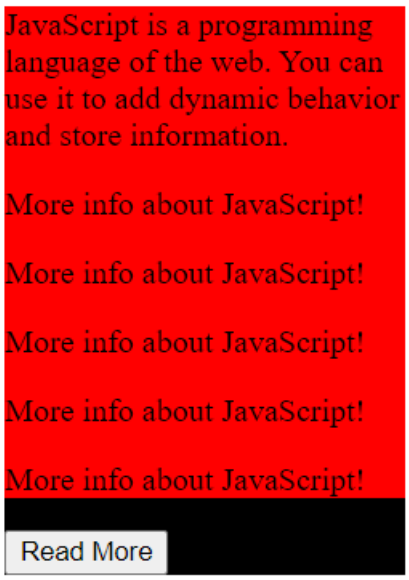
Because this method unregisters event handlers, it needs to identify which function to remove from the event. The event handler function passed to the *.removeEventListener()* method must be the same function of the corresponding *.addEventListener()*.

In a previous example you wrote code to add text to a div element when a button was clicked. The following code is similar,

```
var contentContainer = document.createElement("div");
contentContainer.style.backgroundColor = "black";
document.body.append(contentContainer);

var content = document.createElement("div");
content.innerHTML = "JavaScript is a programming language of the web.  You can use i
t to add dynamic behavior and store information."
content.style.backgroundColor = "red";
contentContainer.append(content);

var readMoreButton = document.createElement("button");
```

```
readMoreButton.innerHTML = "Read More";
contentContainer.append(readMoreButton);

readMoreButton.addEventListener(“click”, addContent);

function addContent(){
    var moreText = document.createElement("p");
    moreText.innerHTML = "More info about JavaScript!";
    content.append(moreText);
}
```

But, let’s consider what happens when the button is clicked multiple times. The click event continues to fire and the same content is added.



To prevent this from happening we can remove the event listener after the content is added.

```
function addContent(){
    var moreText = document.createElement("p");
    moreText.innerHTML = "More info about JavaScript!";
    content.append(moreText);
    readMoreButton.removeEventListener('click',addContent);
}
```

Skill 24.04 Exercises 1

Skill 24.05: Explore mouse events

Skill 24.05 Concepts

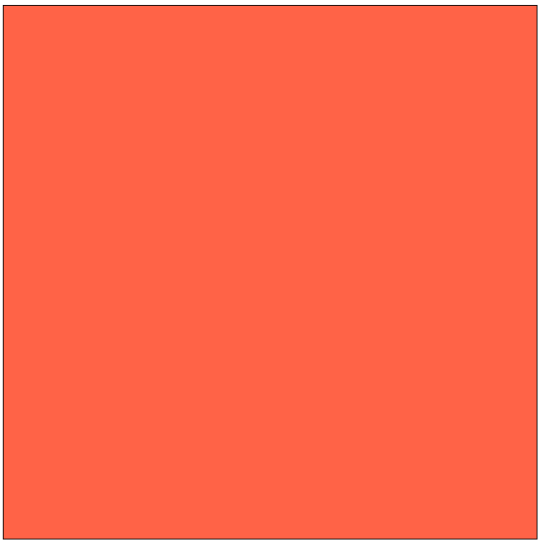
Beyond the click event, there are all types of DOM events that can fire in a browser! It's important to know *most* events in the DOM take place without being noticed because there are no event handlers connected to them.

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's 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 page - https://developer.mozilla.org/en-US/docs/Web/Events.

The following illustrates how to trigger an event using the mouse wheel,

```
//creates a simple div
var myDiv = document.createElement("div");
myDiv.style.width = "500px";
myDiv.style.height = "500px";
myDiv.style.border = "black solid thin";
document.body.append(myDiv);

//registers a wheel event
myDiv.addEventListener("wheel",changeColor);

//when the mouse wheel is used the
//color of the div is changed
function changeColor(){
    myDiv.style.backgroundColor = "tomato";
}
```



Skill 24.05 Exercises 1

Skill 24.06: Explore keyboard events

Skill 24.06 Concepts

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.

Below illustrates how to move an element using the keydown and keyup events,

```
var someElement = document.createElement("div");
someElement.style.width = "100px";
someElement.style.height = "100px";
someElement.style.backgroundColor = "purple";
someElement.style.position = "absolute";
someElement.style.top = "0px";
document.body.append(someElement);

document.body.addEventListener('keydown',moveDown);
document.body.addEventListener('keyup', moveUp);

function moveDown(){
    someElement.style.top = "300px";
    console.log("hello")
}

function moveUp(){
    someElement.style.top = "0px";
}
```

It is often time useful to know which key is pressed. The following code will print which key is pressed to the console. Notice in the below example, we passed a parameter the function. This parameter represents the event. .key is a property associated with the event.

```
document.body.addEventListener('keypress', getKey);

function getKey(e){

    console.log(e.key);
}
```