Events

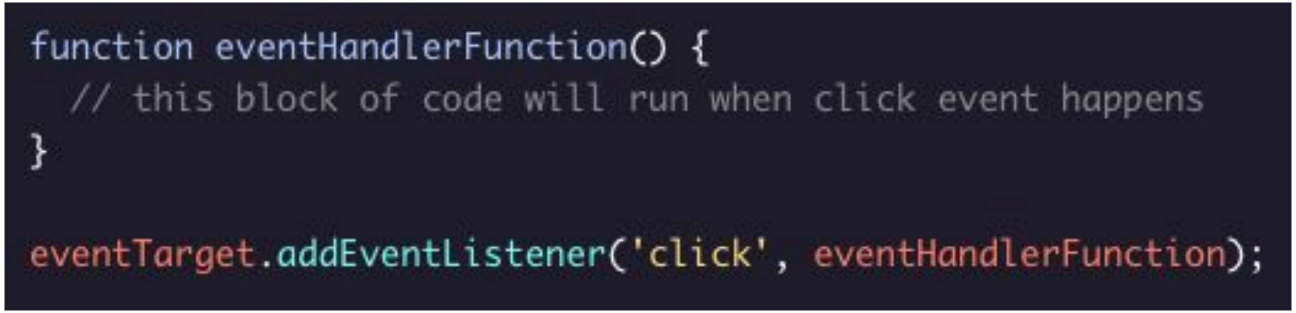
Events on the web are user interactions and browser manipulations that you can program to trigger functionality.

Being able to respond to these events makes your website interactive and therefore *dynamic*.

After a specific event fires on a specific element in the document object model (or DOM), an *event handler* function can be created to run as a response.

Event Handler Registration

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*.

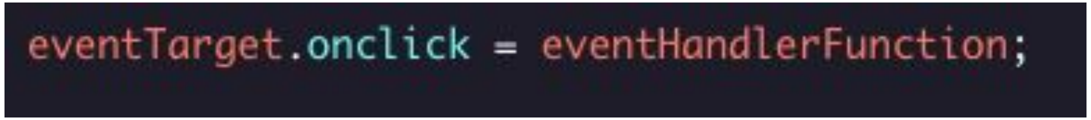


```
function eventHandlerFunction() {
  // this block of code will run when click event happens
}

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

Adding Event Handlers

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:

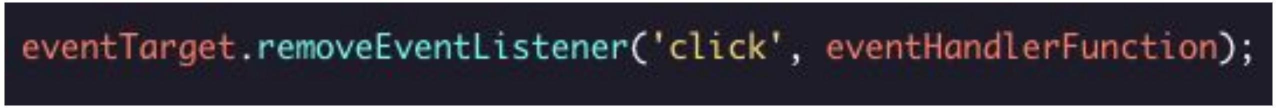


```
eventTarget.onclick = eventHandlerFunction;
```

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.

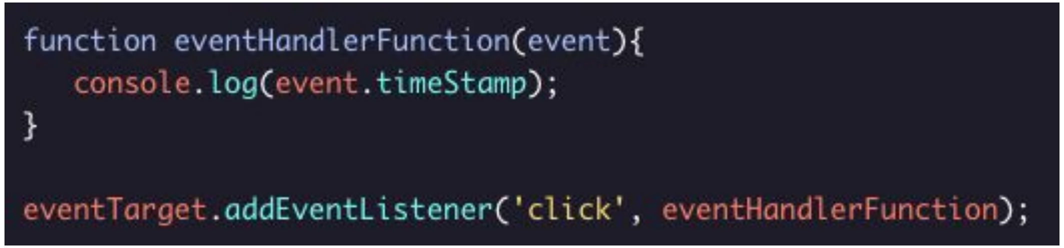
Because there can be multiple event handler functions associated with a particular event, `.removeEventListener()` needs both the exact event type name and the name of the event handler you want to remove. If `.addEventListener()` was provided an anonymous function, then that event listener cannot be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



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

Event Object Properties

JavaScript stores events as `Event` objects 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);
```

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 reference the element that the event is registered to.
- 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.

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.

Browsers can fire many other events without a user — you can check out a list of events on the MDN Events Reference page.

Mouse Events

The `wheel` event is fired when the user rotates a wheel button on a pointing device.

The `mousedown` event is fired when the user presses a mouse button down.

The `mouseup` event is fired when the user releases the mouse button.

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.

Keyboard Events

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.

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.

Activity