7.4 Event-driven programming

Events and event handlers

An ***event*** is an action, usually caused by a user, that the web browser responds to. Ex: A mouse movement, a key press, or a network response from a web server. Typically, the occurrence and timing of an event are unpredictable, since the user or web server can perform an action at any time.

Event-driven programming is a programming style where code runs only in response to various events. Code that runs in response to an event is called an ***event handler*** or ***event listener***.

The web browser supports event-driven programming to simplify handling the many events a webpage must process. When an event happens, the browser calls the event's specified handlers. The web browser internally implements the code for detecting events and executing event handlers.

The example below modifies an input's `style` property, which sets the element's inline CSS styles. The input's border color changes colors when the input receives the focus or when focus is removed.

PARTICIPATION ACTIVITY 7.4.1: Focus and blur event handling.

1 2 3 4 5 2x speed

```
<p>
   <label for="name">Name:</label>
   <input type="text" id="name">
</p>
<p>
   <label for="answer">Answer:</label>
   <input type="number" id="answer">
</p>
```

Name: Authur Dent

Answer: 42

```
let inputs = document.querySelectorAll("input");

for (let i = 0; i < inputs.length; i++) {
   let input = inputs[i];
   input.style.border = "1px solid red";
   input.addEventListener("focus", function() {
      this.style.border = "1px solid green";
   });
   input.addEventListener("blur", function() {
      this.style.border = "1px solid blue";
   });
}
```

User key presses are sent to Answer input box.

Captions

1. User clicks in the Name input box. Browser calls the input element's focus event handler, which changes the element's style. Browser then gives focus to input box.
2. User key presses are sent to Name input box.
3. User clicks the Answer input box. Browser calls the Name element's blur event handler, then calls the Answer element's focus handler, and then gives focus to the Answer input box.
4. User key presses are sent to Answer input box.
5. When the user clicks elsewhere, the browser calls the Answer blur event handler.

Feedback?

PARTICIPATION ACTIVITY 7.4.2: Event-driven programming.

1) The actions a web browser notices are called event handlers.
   - True
   - False (selected)

Correct

The actions are called events, and the code that executes in response to events are called event handlers.

2) A web developer must implement the code to detect events and call the appropriate handlers.
   - True
   - False (selected)

Correct

The web browser hides the code that calls the handlers so a web developer can focus on writing event handlers.

3) A mouse click causes an event.
   - True (selected)
   - False

Correct

A webpage may need to change functionality based on when and where the mouse is clicked, so mouse click events are detected by the browser. A web developer can write a handler that implements the desired behavior in the webpage when the event occurs.

Feedback?

Common events

Each event is given a name that represents the corresponding action. Ex: The event name for a mouse movement is `mousemove`, and the event name for a key down is `keydown`.

PARTICIPATION ACTIVITY 7.4.3: Mouse and keyboard events.

If unable to drag and drop, refresh the page.

click	Caused by a mouse click. The browser creates a click event when the user does a button press.	Correct
mouseover	Caused by mouse entering the area defined by an HTML element. A mouseover event occurs when the mouse moves over the HTML element area from outside the area.	Correct
mouseout	Caused by mouse exiting the area defined by an HTML element. A mouseout event occurs when the mouse moves from inside the HTML element area to an area outside that HTML element.	Correct
mousemove	Caused by mouse moving within the area defined by an HTML element. A mousemove event occurs when the mouse moves while remaining in the HTML element area.	Correct
keydown	Caused by the user pushing down a key on the keyboard. A keydown event occurs when the key is pressed. A separate event occurs when the key is released.	Correct
keyup	Caused by the user releasing a key on the keyboard.	Correct

A keyup event occurs when the key is released. A separate event occurs when the key is pressed.

Reset

Feedback?

The following are events for which web developers commonly write handlers:

- A ***change*** event is caused by an element value being modified. Ex: Selecting an item in a radio button group causes a change event.
- An ***input*** event is caused when the value of an input or textarea element is changed.
- A ***load*** event is caused when the browser completes loading a resource and dependent resources. Usually load is used with the body element to execute code once all the webpage's CSS, JavaScript, images, etc. have finished loading.
- A ***DOMContentLoaded*** event is caused when the HTML file has been loaded and parsed, although other related resources such as CSS, JavaScript, and image files may not yet be loaded.
- A ***focus*** event is caused when an element becomes the current receiver of keyboard input. Ex: Clicking in an input field causes a focus event.
- A ***blur*** event is caused when an element loses focus and the element will no longer receive future keyboard input.
- A ***submit*** event is caused when the user submits a form to the web server.

PARTICIPATION ACTIVITY 7.4.4: Common browser events.

1) A submit event occurs when any button is clicked.
   - ○ True
   - ◉ False

   Correct
   A submit event occurs when a submit button is clicked, or when a submit handler is registered for any button.

2) A blur event occurs when the mouse is moved over another input element.
   - ○ True
   - ◉ False

   Correct
   Moving the mouse causes mousemove and mouseout events. Another input element must be selected to cause the blur event. Input elements are usually selected by clicking on the element or pressing the tab key.

3) The DOMContentLoaded

   Correct

event is likely to occur before the load event.

- (•) True
- ( ) False

DOMContentLoaded occurs immediately after the HTML file is parsed, but load does not occur until all the webpage's resources have downloaded.

Feedback?

Registering event handlers

Handlers are written in three ways:

1. Embedding the handler as part of the HTML. Ex: `<button onclick="clickHandler()">Click Me</button>` sets the click event handler for the button element by using the `onclick` attribute. The attribute name used to register the handler adds the prefix "on" to the event name. Ex: The attribute for a mousemove event is `onmousemove`. Embedding a handler in HTML mixes content and functionality and thus should be avoided whenever possible.
2. Setting the DOM node event handler property directly using JavaScript. Ex: `document.querySelector("#myButton").onclick = clickHandler` sets the click event handler for the element with an id of "myButton" by overwriting the `onclick` JavaScript property. Using DOM node properties is better than embedding handlers within the HTML but has the disadvantage that setting the property only allows one handler for that element to be registered.
3. Using the JavaScript ***addEventListener()*** method to register an event handler for a DOM object. Ex: `document.querySelector("#myButton").addEventListener("click", cli` registers a click event handler for the element with the id "myButton". *Good practice is to use the `addEventListener()` method whenever possible, rather than using element attributes or overwriting JavaScript properties. The `addEventListener()` method allows for separation of content and functionality and allows multiple handlers to be registered with an element for the same event.*

Every handler has an optional ***event object*** parameter that provides details of the event. Ex: For a keyup event, the event object indicates which key was pressed and released, or for a click event, which element was clicked.

In the animation below, `keyupHandler()` uses `event.target` to access the text box object where the keyup event occurred. Inside an event handler, the `this` keyword refers to the element to which the handler is attached. So `event.target` and `this` both refer to the text box object in the event handler.

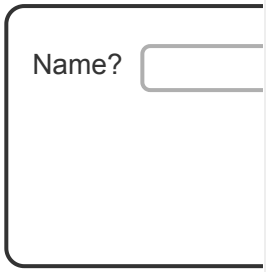
PARTICIPATION ACTIVITY 7.4.5: Registering event handlers with addEventListener().

1 2 3 4 5 6 7 ▶ ☑ 2x speed

```
<!DOCTYPE html>
<html>
  <title>Keyup Demo</title>
  <script>
 window.addEventListener("DOMContentLoaded", loadedHandler);

 function loadedHandler() {
    let textBox = document.querySelector("#name");
    textBox.addEventListener("keyup", keyupHandler);
 }

 function keyupHandler(event) {
    if (event.key == "Enter") {
       let textBox = event.target;
       alert("Hello, " + textBox.value + "!");
    }
 }
  </script>
  <body>
    <label for="id">Name?</label>
    <input type="text" id="name">
  </body>
</html>
```



The window's addEventListener() method registers the handler loadedHandler() for the DOMCor event.

Captions

1. The window's addEventListener() method registers the handler loadedHandler() for the DOMContentLoaded event.
2. After the rest of the HTML is loaded and parsed, the DOMContentLoaded event occurs, and loadedHandler() is called.
3. The text box's addEventListener() method registers the handler keyupHandler() for the keyup event.
4. When the user types the first letter, a keyup event occurs, which results in keyupHandler() being called.
5. The event.key is a string representing the pressed key ("P" for key P).
6. Each keyup causes keyupHandler() to execute. When the user presses Enter, event.key is "Enter", and the if statement is true.
7. event.target is the text box object that caused the keyup event. An alert dialog displays "Hello, Pam!"

Feedback?

PARTICIPATION ACTIVITY 7.4.6: Registering event handler using addEventListener().

The JavaScript code registers mouseover and mouseout event handlers for all elements that use the `highlight` class. Create and register a JavaScript event handler called `myClickHandler()` to handle click events for the same elements. The `myClickHandler()` function should reveal the hidden text by changing the `style.color` of the `event.target` to `"black"`.

```
<body>
   <p>
      Challenge your knowledge about event-driven programming
      each sentence below.  Click the missing text to reveal t
   </p>
   <ul>
      <li>
         Event handlers are also known as
         <span class="highlight hide">callback functions</spar
         because the handlers are "called back" when the appro
      </li>
      <li>
         Event-driven programming allows webpages to react to
         <span class="highlight hide">user actions</span> and
         <span class="highlight hide">web server actions</spar
```

Render webpage Reset code

Your webpage

Challenge your knowledge about event-driven programming by guessing the text that is missing from each sentence below. Click the missing text to reveal the answer.

- Event handlers are also known as _____, because the handlers are "called back" when the appropriate event happens.
- Event-driven programming allows webpages to react to _____ and _____.

Expected webpage

Challenge your knowledge about event-driven programming by guessing the text that is missing from each sentence below. Click the missing text to reveal the answer.

- Event handlers are also known as _____, because the handlers are "called back" when the appropriate event happens.
- Event-driven programming allows webpages to react to _____ and _____.

▼ View solution

Explain

```
--- START FILE: HTML ---

<body>
   <p>
      Challenge your knowledge about event-driven
programming by guessing the text that is missing from
      each sentence below.  Click the missing text to reveal
the answer.
   </p>
   <ul>
```

```
      <li>
         Event handlers are also known as
         <span class="highlight hide">callback
functions</span>,
         because the handlers are "called back" when the
appropriate event happens.
      </li>
      <li>
         Event-driven programming allows webpages to react
to
         <span class="highlight hide">user actions</span>
and
         <span class="highlight hide">web server
actions</span>.
      </li>
   </ul>
</body>


--- END FILE: HTML ---

--- START FILE: JavaScript ---

function myMouseoverHandler(event) {
   event.target.style.backgroundColor = "yellow";
}

function myMouseoutHandler(event) {
   event.target.style.backgroundColor = "white";
}

function myClickHandler(event) {
   event.target.style.color = "black";
}

// Register the event handlers here
let elements = document.getElementsByClassName("highlight");
for (let i = 0; i < elements.length; i++) {
   elements[i].addEventListener("mouseover",
myMouseoverHandler);
   elements[i].addEventListener("mouseout",
myMouseoutHandler);
   elements[i].addEventListener("click", myClickHandler);
}
```

```
--- END FILE: JavaScript ---

--- START FILE: CSS ---

.highlight {
   border: 1px dashed gray;
   cursor: pointer;
}
.hide {
   color: transparent;
}

--- END FILE: CSS ---
```

Feedback?

PARTICIPATION ACTIVITY 7.4.7: Registering event handlers.

Refer to the HTML below.

```
<body>
  <h1>Calculator</h1>
  <p>
    <input type="text" id="num1" size="5">
    <input type="text" id="num2" size="5">
    <span id="result"></span>
  </p>
  <input type="button" value="Add" id="addBtn">
</body>
```

1) What event registers `loadedHandler()` to be executed after the HTML has been loaded and parsed?

```
window.addEventListener("_____",
loadedHandler);
```

- ○ click
- ◉ DOMContentLoaded
- ○ ready

Correct

The DOMContentLoaded occurs when the DOM is complete from loading and parsing all HTML.

2) What is missing to register the `addNumbers()` function as a click event handler?

```
function loadedHandler() {
   let addBtn =
document.querySelector("#addBtn");
   addBtn.addEventListener("click",
_______);
}

function addNumbers() {
   let num1 = parseFloat(

document.querySelector("#num1").value);
   let num2 = parseFloat(

document.querySelector("#num2").value);

document.querySelector("#result").innerHTML
= num1 + num2;
}
```

- addNumbers()
- addNumbers(1, 2)
- addNumbers (selected)

Correct

Only the function name should be specified to register an event handler.

3) What code registers an anonymous function as a click event handler for the add button?

```
function loadedHandler() {
   let addBtn =
document.querySelector("#addBtn");

addBtn.addEventListener("click",
_______);
}
```

- function addNumbers() { ... }
- function() { ... } (selected)
- function { ... }

Correct

An anonymous function does not have a name. Some developers prefer anonymous functions for event handlers over named functions when anonymous functions require writing fewer lines of code.

4) The `highlightField()` function is an event handler for the mouseover and mouseout events. What parameter is `highlightField()` missing?

```
function
highlightField(_____) {
   if
(event.target.style.background
== "yellow") {

event.target.style.background
= "white";
   }
   else {

event.target.style.background
= "yellow";
   }
}
```

- (●) event
- ( ) field
- ( ) color

Correct

The event object parameter is used to toggle the element's background color.

5) What parameter is `highlightField()` missing to change the `textBox` background color to yellow?

```
textBox.addEventListener("mouseover",
highlightField);
function highlightField() {
   ____.style.background = "yellow";
}
```

- ( ) event.target
- ( ) event
- (●) this

Correct

`this` refers to the element that registered the mouseover event, which is the textBox object.

Feedback?

Capturing, at target, and bubbling phases

When an event occurs, the browser follows a simple DOM traversal process to determine which handlers are relevant and need to be called. This traversal process follows three phases: capturing, at target, and bubbling.

1. In the ***event capturing*** phase, the browser traverses the DOM tree from the root to the event target node, at each node calling any event-specific handlers that were explicitly registered for activation during the capturing phase.

2. In the ***at target*** phase, the browser calls all event-specific handlers registered on the target node.
3. In the ***event bubbling*** phase, the browser traverses the DOM tree from the event target node back to the root node, at each node calling all event-specific handlers registered for the bubbling phase on the current node.

The optional third parameter for the `addEventListener()` method indicates whether the handler is registered for the capturing phase or bubbling phase. If the third parameter is `false` or not specified, or if the event handler is registered using any other mechanism, the browser registers the handler for the event bubbling phase. If the parameter is `true`, the browser registers the handler for the capturing phase.

Some events do not bubble, such as blur, focus, and change. When a non-bubbling event occurs, the browser will follow the event capturing phase, the at target phase, and then stop.

PARTICIPATION ACTIVITY 7.4.8: Capturing and bubbling.

1 2 3 4 5 6 7 2x speed

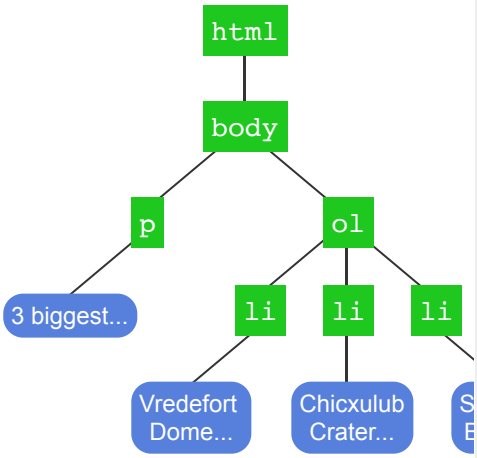
```
<!DOCTYPE html>
<html>
<title>Meteors</title>
<script src="meteors.js" defer></script>
<body>
   <p>3 biggest meteor strikes on Earth:</p>
   <ol id="strikeList">
      <li>Vredefort Dome, South Africa</li>
      <li>Chicxulub Crater, Mexico</li>
      <li>Sudbury Basin, Ontario, Canada</li>
   </ol>
</body>
</html>
```

3 biggest meteor strikes on Earth:
1. Vredefort Dome, South Africa
2. Chicxulub Crater, Mexico
3. Sudbury Basin, Ontario, Canada

```
let list = document.getElementById("strikeList");

// Register handler for event bubbling phase
list.addEventListener("mouseover", function(e) {
   e.target.style.color = "red";
});

// Register handler for event capturing phase
list.addEventListener("mouseout", function(e) {
   e.target.style.color = "black";
}, true);
```



The event bubbling phase looks for any relevant mouseout event handlers by moving up the DO
no elements have mouseout handlers registered for the bubbling phase.

Captions

1. The user moves the mouse cursor over the list's first item. A mouseover event occurs with the first li node as the target node.
2. Event capturing phase traverses the DOM tree from the root to the event target node. No capturing handlers are registered for the mouseover event.

3. At target phase looks for mouseover event handlers registered on the target node, but no mouseover handlers are registered for the first li node.
4. Event bubbling traverses DOM tree from the event node back to the root node. The ol node's bubbling event handler is called and changes the item's text to red.
5. A mouseout event occurs targeting the first li node. Event capturing phase traverses the DOM tree from the root to event target node. The mouseout event handler turns the list item black.
6. The at target phase looks for relevant mouseout event handlers registered on the target node, but no mouseout handlers are registered for the first li node.
7. The event bubbling phase looks for any relevant mouseout event handlers by moving up the DOM tree, but no elements have mouseout handlers registered for the bubbling phase.

Feedback?

PARTICIPATION ACTIVITY 7.4.9: Capturing and bubbling.

Given the HTML and JavaScript below, match the order of alerts to the action performed by the user.

```
<div id="div1">
   <div id="div2">
      <div id="div3">
      </div>
   </div>
</div>
```

```
let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");
let div3 = document.getElementById("div3");

div1.addEventListener("click", function(){ alert("Capture 1"); },
true);
div2.addEventListener("click", function(){ alert("Capture 2"); },
true);
div3.addEventListener("click", function(){ alert("Capture 3"); },
true);

div1.addEventListener("click", function(){ alert("Bubble 1"); });
div3.addEventListener("click", function(){ alert("Bubble 3"); });
```

If unable to drag and drop, refresh the page.

Capture 1, Bubble 1	User clicks on div with `div1` id. The browser calls the capturing and bubbling event handlers of	**Correct**

	div1 during the at target phase.	
Capture 1, Capture 2, Bubble 1	User clicks on div with `div2` id. The browser calls capturing event handlers of div1 and div2 and then calls the bubbling event handler of div1.	**Correct**
Capture 1, Capture 2, Capture 3, Bubble 3, Bubble 1	User clicks on div with `div3` id. The browser calls capturing event handlers of div1, div2, and div3 and then calls the bubbling event handlers of div3 and div1.	**Correct**

Reset

Feedback?

PARTICIPATION ACTIVITY 7.4.10: Bubbling and capturing.

1) The web browser performs the event capturing process before the bubbling process.
   - (●) True
   - ( ) False

Correct

The web browser traverses the DOM tree from the root to the node where the event occurs (capturing), and then traverses the DOM tree back to the root following the same path but in the reverse direction (bubbling).

2) If a web developer creates a "default" handler for a DOM node high in the tree and a more specific handler for a DOM node lower in the tree, the web browser will run both handlers for an event.
   - (●) True
   - ( ) False

Correct

By default, the web browser will call both handlers.

3) Bubbling is the preferred process for

Correct

the web browser to find appropriate handlers for an event.

- True
- False

The bubbling process matches other graphical event systems and is easier to use. A developer should only use the capturing process for very specific cases.

Feedback?

Preventing default behavior

The event capturing and bubbling process can be stopped by calling the ***stopPropagation()*** method on the event object provided to the handler. Once `stopPropagation()` is called, the browser stops the traversal process but still calls relevant registered handlers on the current node.

A web developer may want to prevent the browser from using a built-in handler for an event. Ex: Whenever a user clicks a form's submit button, the web browser sends the form data to the web server. The event object's ***preventDefault()*** method stops the web browser from performing the built-in handler. The built-in handlers that are often prevented are clicking elements, submitting forms, and moving the mouse into or out of an element.

The example below uses two event handlers for the password textbox:

1. `preventSpaces()` is a keydown event handler that listens for key presses. If the space key is pressed, `event.preventDefault()` stops the space from appearing in the textbox.
2. `checkPassword()` is an input event handler that is called when the password input changes. `checkPassword()` displays Weak, Stronger, Moderate, or Strong in the `<span>` tag depending on various criteria for the password.

Testing password strength.

Start typing a password. Verify the message to the right of the widget changes as the password is improved:

- abc - Weak
- abc1 - Stronger
- abc1D - Moderate
- abc1De - Strong

HTML JavaScript

```
1 <label for="password">Password:</label>
2 <input type="text" id="password">
3 <span id="strength"></span>
4
```

Render webpage Reset code

Your webpage

Password:

Feedback?

PARTICIPATION ACTIVITY 7.4.11: Preventing default behavior.

1) A web developer cannot prevent the web browser from performing built-in handlers.

- True
- False

Correct

The web developer can call `preventDefault()` on an event object to stop the built-in action from occurring.

2) If a web developer creates a "default" handler for a DOM node high in the tree and a more specific handler for a DOM node lower in the tree, `stopPropagation()` can be called in the more specific handler to keep the browser from calling the default handler.

- (●) True
- ( ) False

Correct

If `stopPropagation()` is called from the specific handler, the web browser will not call handlers higher in the DOM tree.

3) In the example above, a user may press a space in the password textbox, but the space does not appear.

- (●) True
- ( ) False

Correct

The `preventDefault()` function prevents the built-in keydown handler from executing, so spaces are ignored.

4) In the example above, `checkPassword()` prevents the built-in input handler from executing.

- ( ) True
- (●) False

Correct

The `checkPassword()` function never calls `preventDefault()`.

5) In the example above, the password "qwerty1" causes the webpage to display "Strong".

- ( ) True
- (●) False

Correct

The password is at least 6 characters long and contains a digit, but the password does not contain at least one uppercase character. The webpage would display "Moderate".

Feedback?

CHALLENGE ACTIVITY 7.4.1: Event-driven programming.

530096.4000608.qx3zqy7

Jump to level 1

1
2
3

Write and register an event handler that changes the background color of the h2 tag to blue on mouseover.

HTML JavaScript

```
1 let h2Element = document.getElementsByTagName("h2")[0];
2
3 /* Your solution goes here */
4 function handleMouseover(){
5    h2Element.style.backgroundColor = 'blue';
6 }
7 h2Element.addEventListener('mouseover',handleMouseover);
```

1 2 3

Check Next

Done. Click any level to practice more. Completion is preserved.

✔

✔ Checking background color on h2 element before any event
Yours *Yours has no value*

✔ Checking background color on h2 after event on h2
Yours `blue`

✔ Checking background color on h1 after event on h1
Yours *Yours has no value*

✔ Checking background color on h3 after event on h3
Yours *Yours has no value*

✔ Checking background color on p after event on p
Yours *Yours has no value*

✔ Checking background color on a after event on a
Yours *Yours has no value*

View your last submission

```
/* Your solution goes here */
function handleButtonClick() {
   pElement.style.backgroundColor = 'yellow';
}

pElement.addEventListener('click', handleButtonClick);
```

Feedback?

Exploring further:

- Event reference from MDN
- EventTarget.addEventListener() from MDN
- Event flow tutorial from Java2s

How was this section? Provide section feedback