**Chapter 4 JavaScript Interactivity**

*Last updated: 2021-08-24 14:54:58*

**4.1 Introduction**

In Chapters [1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html)–[2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#css), we mentioned that JavaScript is primarily used to control the interactive behavior of web pages. However, in Chapter [3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#javascript-basics) we introduced the JavaScript language “in isolation”: none of the JavaScript expressions we executed in the console had any effect on the contents of the web page where they were run. Indeed, how can we link JavaScript code with page contents, to interactively modify the web page? This is precisely the subject of the current chapter—employing JavaScript to query and modify web page content. As we will see, the mechanism which makes the link possible is the **Document Object Model (DOM)**.

**4.2 The Document Object Model (DOM)**

When a web browser loads an HTML document, it displays (renders) the contents of that document on the screen, possibly styled according to CSS styling rules. But that’s not all the web browser does with the tags, attributes, and text contents of the HTML document. The browser also creates and memorizes a “model” of that page’s HTML (Figure [4.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:dom)). In other words, the browser remembers the HTML tags, their attributes, and the order in which they appear in the file. This representation of the page is called the [**Document Object Model (DOM)**](https://en.wikipedia.org/wiki/Document_Object_Model).

The DOM is sometimes referred to as an [**Application Programming Interface (API)**](https://en.wikipedia.org/wiki/Application_programming_interface) for accessing HTML documents with JavaScript. An API is a general term for methods of communication between software components. The DOM can be considered an API, since it bridges between JavaScript code and page contents displayed in the browser. The DOM basically provides the information and tools necessary to navigate through, or make changes or additions, to the HTML on the page. The DOM itself isn’t actually JavaScript—it’s a standard from the [World Wide Web Consortium (W3C)](https://www.w3.org/DOM/) that most browser manufacturers have adopted and added to their browsers.

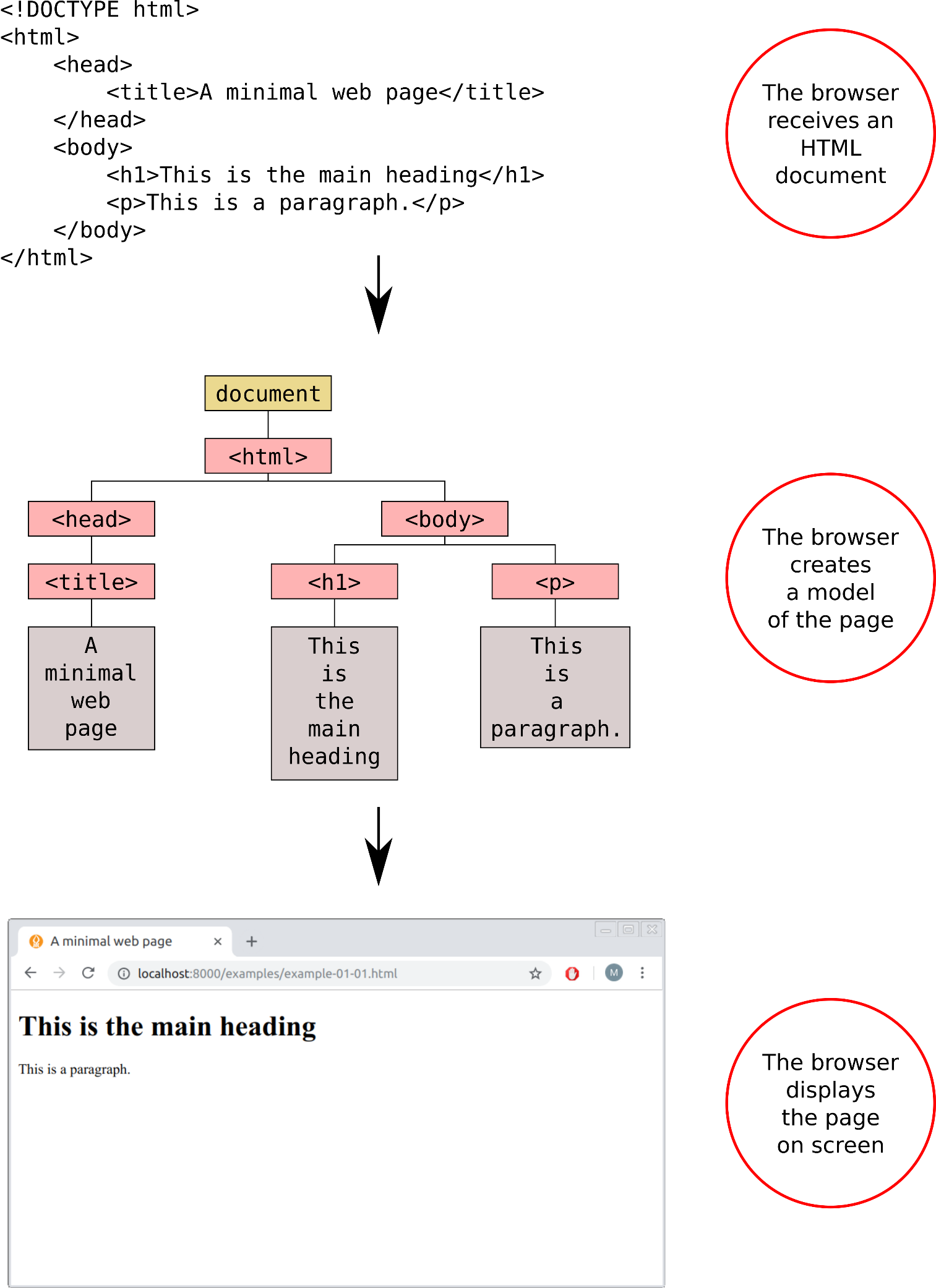


FIGURE 4.1: The Document Object Model (DOM) is a representation of an HTML document, constructed by the browser on page load

**4.3 Accessing and modifying elements**

**4.3.1 Overview.**All of the properties and methods of the DOM, available for manipulating web pages, are organized into objects which are accessible through JavaScript code. Two of the most important objects are:

* The window object, which represents the browser **window**, i.e., the global environment
* The document object, which represents the **document** itself[39](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn39)

Using the properties and methods of the window and document objects, we are able to access document element contents or their display settings, and also dynamically adjust them. For example, the document object has a property named title, referring to the text within the <title> element—the page title (Section [1.6.3.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html-title)). We can access document.title, to get a string with page title, or modify page title, by assigning a new string to document.title.

* Open the browser and browse to any web page you like.
* Open the JavaScript console (Section [3.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#the-javascript-console)) and try typing document.title and window.location.href.
* Check the type of returned values, using the typeof operator (Section [3.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#checking-type-of-variables)).
* Try assigning a new string into document.title. The new title should appear on top of the browser window!
* Go to any page (other than [https://www.google.com](https://www.google.com/)) and type window.location.href="https://www.google.com" into the console. What do you think will happen?

**4.3.2 Accessing elements.**The document.title property was just an example, referring to the specific <title> element. We need a more general method if we want to be able to locate, and modify, *any* element in our document. The document object indeed contains several methods for finding elements in the DOM. These methods are called DOM **selectors**, or DOM queries. The following expressions are examples of DOM selectors:

document.getElementById("firstParagraph");

document.getElementsByClassName("important");

document.getElementsByTagName("p");

These three expressions employ methods of the document object:

* The *first* selector uses the .getElementById method to select an individual element with a specific ID, id="firstParagraph".
* The *second* selector uses the .getElementsByClassName method to select *all* elements with a specific class, class="important".
* The *third* selector uses the .getElementsByTagName method to select *all* elements with a specific type, <p> (i.e., all paragraphs)[40](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn40).

Note that the first method returns an *individual* element reference. The second and third methods, however, returns an array-like *collection* of multiple references, even if there is just one element with the particular class or type (more on that in Section [4.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#making-multiple-selections)). We are going to see just one example of working with a collection of DOM references in Section [4.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#making-multiple-selections). Selecting one element at a time, using .getElementById, is going to be sufficient for all of the examples in this chapter and throughout most of the book.

* Open the console in example-02-01.html (Figure [2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#fig:example-02-01)), and type the following expressions:
  + document.getElementById("intro");
  + document.getElementsByTagName("p");
* Which element(s) does each of the resulting references represent?

**4.3.3 Getting element properties.**The result of a DOM query (Section [4.3.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#accessing-elements)) is a **reference** to an element, or to a collection of elements, in the DOM. That reference can be used to query or to modify (Section [4.3.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#modifying-elements)) the contents or behavior of the element(s). For example, the innerHTML property of a DOM element refers to the HTML code of that element. The following expression returns the HTML contents of the element that has id="firstParagraph":

document.getElementById("firstParagraph").innerHTML;

* Open the console in example-02-01.html (Figure [2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#fig:example-02-01)), and type the expression document.getElementById("intro").innerHTML;.
  + What does the returned data represent?
  + What is the type of the returned data? Check using the typeof operator (Section [3.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#checking-type-of-variables)).

**4.3.4 Modifying elements.**Element references can also be used to *modify* the contents and the behavior (Section [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners)) of the respecive elements. For example, assignment of a string into the .innerHTML property replaces the HTML code of the given element. The following expression uses the innerHTML property to modify the HTML content of the element which has id="firstParagraph", replacing any previous HTML code inside that element with Hello <b>world</b>!:

document.getElementById("firstParagraph").innerHTML = "Hello <b>world</b>!";

* Open the console in example-02-01.html (Figure [2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#fig:example-02-01)) once again, and then:
  + Type an expression which changes the contents of the paragraph that has id="intro" into the console.
  + Include an internal HTML element, such as <b> or <i>, in the new text contents of the paragraph.

It is important to understand that the HTML **source code** and the **DOM** are two separate entities. While the HTML source code sent to the browser is constant, the DOM, which is initially constructed from the HTML source code (Figure [4.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:dom)), can be dynamically altered using JavaScript code. As long as no JavaScript code that modifies the DOM is being run, the DOM and the HTML source code are identical. However, when JavaScript code *does* modify the DOM, the DOM changes and the displayed content in the browser window changes accordingly.

The current DOM state can be examined, for example, using the **Elements** tab of the **developer tools** (in Chrome). The HTML source code can be shown with **Ctrl+U** (in Chrome), as we have already discussed in Section [1.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#web-pages). Again, the source code remains exactly the same as initially sent from the server, no matter what happens on the page, while the DOM may be modified by JavaScript code and thus may change. This means that the HTML source code does not necessarily match the DOM once any JavaScript code was executed.

* While running the examples in this chapter, you can compare the HTML source code and the DOM to see how the DOM changes in response to executed JavaScript code, while the HTML source code remains the same as initially loaded.

**4.4 Event listeners**

**4.4.1 Event types.**Sometimes we want to change the DOM at once, for example on page load (Section [4.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#modifying-page-based-on-data)). In other cases, however, we want to change the DOM *dynamically*, in response to user actions on the page, such as clicking on a button. This is where the DOM **events** come into play. Each and every thing that happens to a web page is called an event. Web browsers are programmed to recognize various events, including user actions such as:

* Mouse movement and clicks
* Pressing on the keyboard
* Re-sizing the browser window

An event represents the precise moment when something happens inside the browser. This is sometimes referred to as the event being *fired*. There are different types of events, depending on the type of action taking place. For example, when you click a mouse, at the precise moment you release the mouse button the web browser signals that a "click" event has just occurred. In fact, web browsers fire several separate events whenever you click the mouse button. First, as soon as you press the mouse button, the "mousedown" event fires; then, when you let go of the button, the "mouseup" event fires; and finally, the "click" event fires.

To make your web page interactive, you need to write code that runs and does something useful in response to the appropriate type of event occurring on the appropriate element(s). This type of binded code is known as an **event listener**, or an event handler. For example, we may wish to set an event listener which responds to user click on an interactive map by adding a marker in the clicked location (Section [11.2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#adding-custom-marker)). In such case, we need to bind an event listener:

* to the interactive map object,
* with a function for adding a marker,
* which is executed in response to a mouse "click" event.

A mouse click is just one example, out of many predefined event types that the browser can detect. Table [4.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#tab:event-types) lists some commonly used event types.

| TABLE 4.1: Commonly used browser event types | | |
| --- | --- | --- |
| **Type** | **Event** | **Description** |
| Mouse | "click" | Click |
|  | "dblclick" | Double-click |
|  | "mousedown" | Mouse button pressed |
|  | "mouseup" | Mouse button released |
|  | "mouseover" | Mouse cursor moves into element |
|  | "mouseout" | Mouse cursor leaves element |
|  | "mousemove" | Mouse moved |
|  | "drag" | Element being dragged |
| Keyboard | "keydown" | Key pressed |
|  | "keypress" | Key pressed (character keys only) |
|  | "keyup" | Key released |
| Focus and Blur | "focus" | Element gains focus (e.g., typing inside <input>) |
|  | "blur" | Element loses focus |
| Forms | "submit" | Form submitted |
|  | "change" | Value changed |
|  | "input" | User input |
| Document/Window | "load" | Page finished loading |
|  | "unload" | Page unloading (new page requested) |
|  | "error" | JavaScript error encountered |
|  | "resize" | Browser window resized |
|  | "scroll" | User scrolls the page |

In this book, we will mostly use the "click", "mouseover", "mouseout", and "drag" events, all from the Mouse events category, as well as "change" from the Forms category (Table [4.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#tab:event-types)). However, it is important to be aware of the other possibilities, such as events related to window resize or scrolling, or keyboard key presses.

**4.4.2 Binding event listeners.**DOM element reference objects contain the .addEventListener method, which can be used to add event listeners to the respective element. The .addEventListener method accepts two arguments:

* A specification of the event **type** (Table [4.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#tab:event-types)) that are being listened to, such as "click"
* A **function**, which is executed each time the event fires

For example, to bind a "click" event listener to a paragraph that has id="important" on the page, you can use the following expression:

document

.getElementById("important")

.addEventListener("click", myFunction);

where myFunction is a function that defines *what* should happen each time the user clicks on that paragraph.

The function we pass to the event listener does not necessarily need to be a predefined named function, such as myFunction. You can also use an anonymous function (Section [3.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#methods)). For example, here is another version of the above event listener definition, this time using an anonymous function:

document

.getElementById("important")

.addEventListener("click", **function**() {

*// Code goes here*

});

**4.5 Hello example**

The following two examples (Sections [4.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#hello-example)–[4.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#poles-example)) demonstrate the ideas and techniques presented so far in this chapter: accessing and modifying elements (Section [4.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#accessing-and-modifying-elements)) and using event listeners (Section [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners)), in JavaScript.

Consider the following HTML code of example-04-01.html:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Hello JavaScript**</title>**

**</head>**

**<body>**

**<h2>**JavaScript**</h2>**

**<p** id="demo"**>**What can JavaScript do?**</p>**

**<input** type="button" value="Click Me!" id="change\_text"**>**

**<script>**

**function** hello() {

document

.getElementById("demo")

.innerHTML =

"JavaScript can change page contents!";

}

document

.getElementById("change\_text")

.addEventListener("click", hello);

**</script>**

**</body>**

**</html>**

In this example, we have a web page with an <h2> heading (*without* an ID), as well as two other elements *with* an ID:

* A <p> element with id="demo"
* An <input> button element with id="change\_text"

In the end of the <body>, we have a <script> element containing JavaScript code (Section [1.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#styling-and-scripts)). Since the <script> element is in the end of the <body>, it will be executed by the browser *after* the HTML code is loaded and processed. Let’s look at the last line in the JavaScript code:

document.getElementById("change\_text").addEventListener("click", hello);

The above expression does several things:

* **Selects** the element which has id="change\_text" (the *button*), using the document.getElementById method (Section [4.3.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#accessing-elements))
* **Binds** an event listener to it, using the .addEventListener method (Section [4.4.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#binding-event-listeners))
* The event listener specifies that when the element is clicked (i.e., the "click" event is fired), the hello function will be executed

What does the hello function do? According to its definition at the beginning of the <script>, we see that it has no parameters and just one expression:

**function** hello() {

document

.getElementById("demo")

.innerHTML = "JavaScript can change page contents!";

}

What the hello function does is:

* **Selects** the element with id="demo" (the *paragraph*), again using the document.getElementById method
* **Replaces** its HTML contents with "JavaScript can change page contents!", by assigning the new contents into the innerHTML property (Section [4.3.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#modifying-elements))

Note that in this example the expression inside the hello function is incidentally split into several (three) lines, to fit on the page. However, new-line symbols and repeated spaces are ignored by the JavaScript interpreter. You can imagine the characters being merged back into a long single-line expression, then executed. In other words, the computer still sees a single expression here, where we assign a string into the .innerHTML property of an element selected using document.getElementById.

The way that example-04-01.html appears in the browser is shown in Figure [4.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-01).

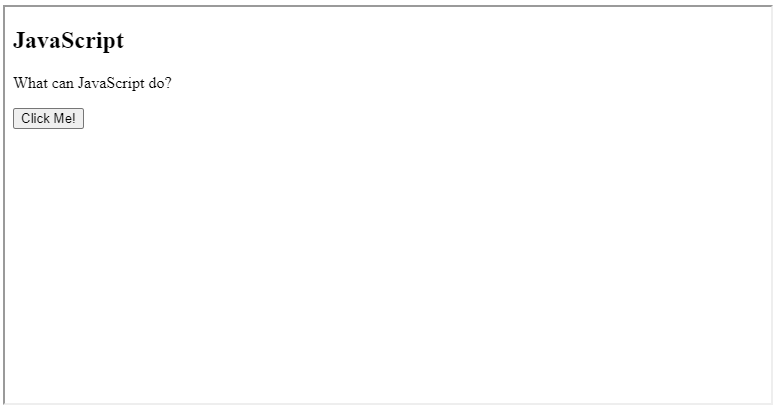


FIGURE 4.2: example-04-01.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-01.html)

* Open example-04-01.html in the browser, then open the **Elements** tab in the developer tools.
* Click the button that says “Click me!” (Figure [4.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-01)), and observe how the paragraph contents are being modified.
* Note how the affected part of the DOM is momentarily highlighted each time the button is pressed.

**4.6 Poles example**

The second example example-04-02.html is slightly more complex, but the principle is exactly the same as in example-04-01.html:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Earth poles**</title>**

**</head>**

**<body>**

**<h2>**Earth Poles Viewer**</h2>**

**<img** id="myImage" src="images/north.svg"**><br>**

**<input** type="button" value="North Pole" id="north"**>**

**<input** type="button" value="South Pole" id="south"**>**

**<script>**

**function** showNorth() {

document

.getElementById("myImage")

.src = "images/north.svg";

}

**function** showSouth() {

document

.getElementById("myImage")

.src = "images/south.svg";

}

document

.getElementById("north")

.addEventListener("click", showNorth);

document

.getElementById("south")

.addEventListener("click", showSouth);

**</script>**

**</body>**

**</html>**

In this example, we have two buttons, and we add two event listeners: one for the North pole button and one for the South pole button (Figure [4.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-02)). Both event listeners change the src attribute of the <img> element (Section [1.6.10.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html-img)) on the web page. The North pole button sets src="images/north.svg", while the South pole button sets src="images/south.svg". The resulting effect is that the displayed globe is switched from viewing the North pole or the South pole[41](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn41).

Note that the images loaded in this example—north.svg and south.svg—are in a format called [**Scalable Vector Graphics (SVG)**](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics), having the .svg file extension. SVG is an open standard *vector* image format. It is well supported by web browsers and commonly used to display vector graphics on web pages.

The result of example-04-02.html is shown in Figure [4.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-02).

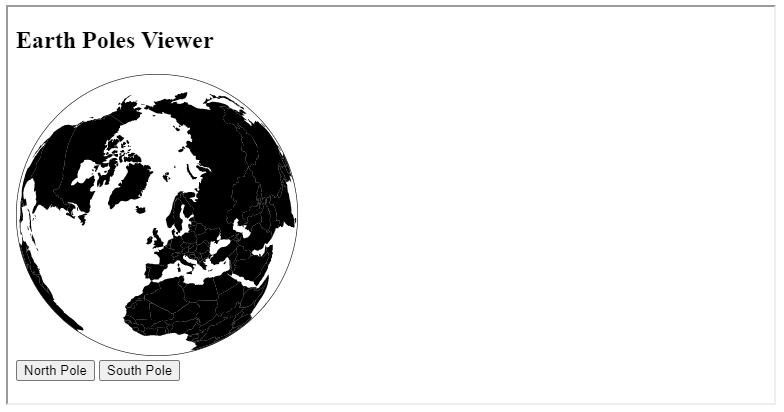


FIGURE 4.3: example-04-02.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-02.html)

**4.7 Operating on multiple selections**

**4.7.1 Example page.**The following small web page (example-04-03.html) will be used to demonstrate selecting and acting on multiple elements (Section [4.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#making-multiple-selections)), accessing input values (Section [4.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#getting-and-setting-input-values)) and working with the event target (Section [4.10.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-target)).

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Operating on multiple selections**</title>**

**</head>**

**<body>**

**<h1** id="header"**>**List**</h1>**

**<h2>**Buy groceries**</h2>**

**<ul>**

**<li** id="one" class="hot"**>**fresh figs**</li>**

**<li** id="two" class="hot"**>**pine nuts**</li>**

**<li** id="three" class="hot"**>**honey**</li>**

**<li** id="four"**>**balsamic vinegar**</li>**

**</ul>**

**<h2>**Comments**</h2>**

**<input** type="text" id="test3" value="Mickey Mouse"**>**

**</body>**

**</html>**

Figure [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-03) shows how example-04-03.html appears in the browser.

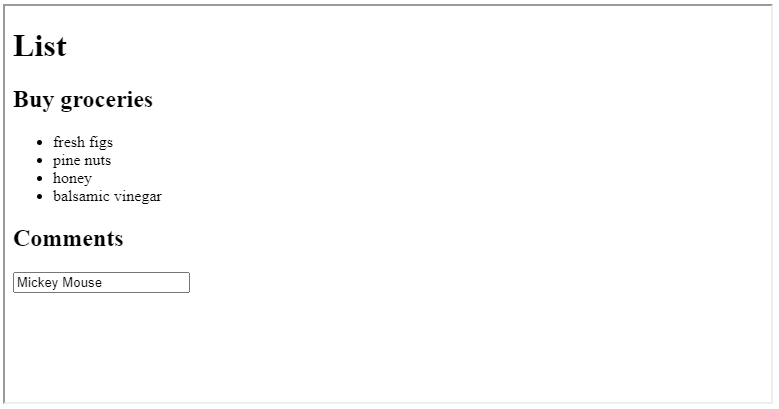


FIGURE 4.4: example-04-03.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-03.html)

You can open example-04-03.html in the browser and run the expressions shown in Sections [4.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#making-multiple-selections) and [4.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#getting-and-setting-input-values) (below) in the console, to experiment with their immediate effect.

**4.7.2 Making multiple selections.**So far, we used the document. getElementById method to make a selection of an individual element. Recall that the id attribute is unique (Section [1.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#id)), thus by definition the selection contains one element. For example, you can run the following command in the console where example-04-03.html is loaded:

document.getElementById("one").innerHTML;

to get the HTML contents of the element that has id="one". The following output should be printed in the console:

fresh figs

We can also modify the HTML contents of an element, as follows:

document.getElementById("one").innerHTML = "<i>Not very fresh</i> figs";

* Open example-04-03.html and run the above expression for changing the contents of the first <li> element in the console.
* Compare the HTML source code (**Ctrl+U**) and the DOM (**Elements** tab in the **developer tools**).
* Which one reflects the above change and why? (Hint: see Section [4.3.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#modifying-elements).)

The document object contains other methods for making DOM selections, potentially returning more than one element. For example:

* getElementsByClassName—Get all elements sharing the specified class
* getElementsByTagName—Get all elements of the specified HTML type[42](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn42)

For example, the following expression uses the getElementsByTagName method to get a reference to all <li> elements in the document:

**let** nodes = document.getElementsByTagName("li");

The returned object is a [collection](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection) of references to HTML elements in the DOM. The collection is an array-like object. For example, we can check its length (Section [3.8.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#array-methods)):

nodes.length; *// Returns 4*

or select an individual element from the collection using an index (Section [3.6.3.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#js-array)):

nodes[0].innerHTML; *// Returns "fresh figs"*

How can we do something with all of the elements in the collection? For example, how can we print the HTML contents of all of the <li> elements? Unfortunately, the collection does not have a .forEach method (Section [3.10.3.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#the-foreach-array-method)). Therefore, to iterate over the collection we need to use the standard for loop syntax (Section [3.10.3.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#standard-for-loop-syntax)):

**for**(**let** i = 0; i < nodes.length; i++) {

console.log(nodes[i].innerHTML);

}

Try to modify the code of the for loop to append an exclamation mark (“!”) at the end of each list item, so that “fresh figs” becomes “fresh figs!” etc.

Using loops, we can therefore work with multiple selections, one element at a time. In this book, we will rarely need to use multiple selection JavaScript techniques, since we will mostly be working with individual selections (using getElementById). However, working with multiple selections is often necessary so it is important to be familiar with this possibility.

**4.8 Getting and setting input values**

The .value property of a reference to an input element—such as <input> or <select>(Section [1.6.13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#input-elements))—can be used to get or set its current *value*. This is conceptually similar to the way that the .innerHTML property is used to get or set HTML contents (Sections [4.3.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#accessing-elements)–[4.3.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#modifying-elements)) and the .src property is used to get or set the .src attribute (Section [4.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#poles-example)).

For example, the web page we are experimenting with (example-04-03.html) contains the following <input> element (Figure [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-03)):

**<input** type="text" id="test3" value="Mickey Mouse"**>**

The following expression gets the current value of that element, which is equal to "Mickey Mouse" unless the user interacted with the text area and typed something else:

document.getElementById("test3").value; *// Returns "Mickey Mouse"*

If you manually modify the contents of the text input area, and run the above expression again, you will see the new, *currently* entered text. The DOM, on the other hand, will still show the original value="Mickey Mouse" attribute.

You may already have guessed that assigning a new value to the .value property modifies the current input value. The following expression sets the input value to "Donald Duck", replacing the previous value of "Mickey Mouse", or whatever else that was typed into the text area, with the new value "Donald Duck":

document.getElementById("test3").value = "Donald Duck";

Note that the .value property refers to the real-time value, including user changes such as currently entered text in a text box. These changes are not reflected in the value *attribute*, which retains the initial value set on page load. Compare the value attribute of the modified <input> (through the DOM inspector in the developer tools) with the .value property string to see this for yourself.

We will get back to another practical example using .value and <input> elements in Section [4.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#working-with-user-input).

* Create a small web page with:
  + A slider input, i.e., <input type="range"> (Section [1.6.13.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#range-input))
  + A paragraph
* The paragraph should display the up-to-date value of the slider, whether the initial value or a new value selected by the user.
* Hints:
  + Bind an event listener to the slider, with function that updates paragraph contents with the current value of the slider.
  + Use the .value property to obtain the current value of the slider.
  + Set the event listener to respond to the "input" event type (Section [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners)), so that the function is executed every time the slider is scrolled (i.e., gets any user *input*).

**4.9 Multiple event listeners**

It is often convenient to add multiple event listeners to the same element selection, in order to trigger different responses for different events. In our next example (example-04-04.html), two event listeners[43](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn43) are binded to the id="p1" paragraph. The first event listener responds to the "mouseover" event (mouse cursor entering the element), printing "You entered p1!" in the console. The second event listener responds to the "mouseout" event (mouse cursor leaving the element), printing "You left p1!" in the console.

**let** el = document.getElementById("p1");

el.addEventListener("mouseover", **function**() {

console.log("You entered p1!");

});

el.addEventListener("mouseout", **function**() {

console.log("You left p1!");

});

As a result, the phrases the "You entered p1!" or "You left p1!" are interchangeably printed in the console whenever the user moves the mouse into the paragraph or out of the paragraph. The small web page implementing the above pair of event listeners (example-04-04.html) is shown in Figure [4.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-04).

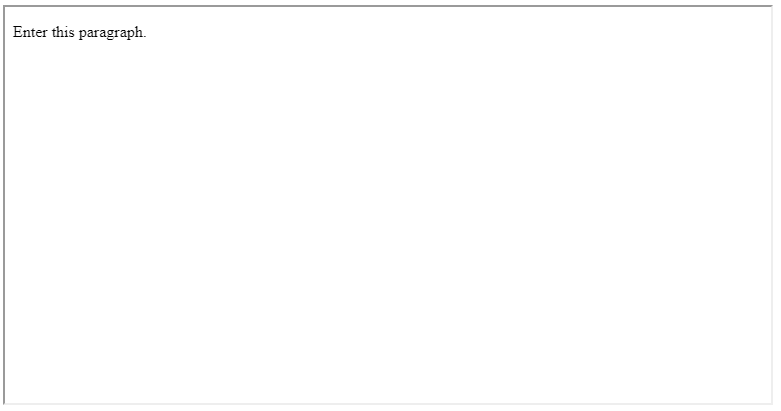


FIGURE 4.5: example-04-04.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-04.html)

* Open example-04-04.html in the browser, and open the JavaScript console.
* Move the mouse cursor over the paragraph and check out the messages being printed in the console (Figure [4.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-04)).
* Try modifying the source code of this example so that the messages are displayed on the web page itself, instead of the console. (Hint: add another paragraph for the messages in the HTML code, and use the .innerHTML property to update the paragraph contents in response to the events.)

At this stage you may note our code starts to have a lot of nested brackets of both types, ( and {. This is typical of JavaScript code, and a common source of errors while learning the language. Make sure you keep track of opening and closing brackets. Most plain text editors, such as Visual Studio Code or Notepad++, automatically highlight the matching opening/closing bracket when placing the cursor on the other one. This can help with code structuring and avoiding errors due to unmatched brackets.

**4.10 The event object**

**4.10.1 Event object properties.**So far (Sections [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners)–[4.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#multiple-event-listeners)), the functions which we passed to an event listener did not use any information regarding the event itself, other than the fact the event has *happened*. Sometimes, however, we may be interested in functions that have a variable effect, depending on event properties: such as *where* and *when* the event happened.

In fact, every time an event happens, an **event object** is passed to any event listener function responding to the event. We can use the event object to construct functions with variable effects depending on event properties. The event object has methods and properties related to the event that occurred. For example[44](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn44):

* .type—Type of event ("click", "mouseover", etc.)
* .key—Button or key that was pressed
* .pageX, .pageY—Mouse position
* .timeStamp—Time in milliseconds, from when the document was opened
* .target—The element that triggered the event (see Section [8.7.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#styling-in-response-to-events))

Every function that responds to an event can take the event object as its parameter. That way, we can use the event object properties in the function body, to trigger a specific action according to the properties of the event.

The following small page example-04-05.html uses the .pageX and .pageY properties of the "mousemove" event to display up-to-date mouse coordinates every time the mouse moves on screen. Note how the event object—conventionally named e, but you can choose another name—is now a parameter of the event listener function. Also note that we are binding the event listener to the entire document, rather than to an individual element, since we want to “listen to” mouse movement over the entire document:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Event object**</title>**

**</head>**

**<body>**

**<p** id="position"**></p>**

**<script>**

**let** el = document.getElementById("position");

document.addEventListener("mousemove", **function**(e) {

el.innerHTML = e.pageX + " " + e.pageY;

});

**</script>**

**</body>**

**</html>**

The result is shown in Figure [4.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-05). The current mouse coordinates are displayed in the top-left corner of the screen. Every time the mouse is moved, the coordinates are updated.

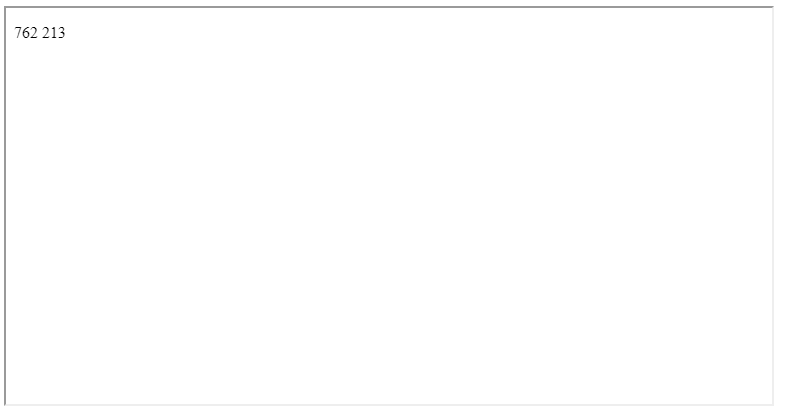


FIGURE 4.6: example-04-05.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-05.html)

What do you think will happen if the string "mouseover" will be replaced with "click", in the code of example-04-05.html? Edit the code file and open the modified copy to check your answer!

Different types of events are associated with different event object properties. Moreover, *custom* event types and event properties can be defined in JavaScript libraries. For example, later on in the book we will use the **map click** event to detect the clicked coordinates on an interactive map, then to trigger specific actions regarding that location. This is made possible by the fact that the "click" event on a map contains a custom property named .latlng, which can be accessed by the event listener function. We will see how this technique can be used for several purposes, such as:

* Displaying the clicked coordinates (Section [6.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#introducing-map-events))
* Adding a marker in the clicked location on the map (Section [11.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#adding-marker-on-click))
* Making a database query based on proximity to the clicked location (Section [11.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#adding-nearest-points-to-map))

In Chapter [13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\collaborative-mapping.html#collaborative-mapping), we will see another group of custom events referring to *drawing* and *editing* shapes on the map, such as "draw:created", which is fired when a new shape is drawn on the map.

**4.10.2 The .target property.**One of the most useful properties of the event object is the event target. The target property provides a reference to the element that *triggered* the event. Naturally, this is most useful in cases when the same event listener is being binded to numerous elements and our function need to be responsive to the particular properties of the element that triggered the event.

For example, the following code section uses the getElementsByTagName method (Section [4.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#making-multiple-selections)) to get a collection of references to all <li> elements in the document. Then, a for loop goes over all elements and binds a "click" event listener to each one. The event listener function uses e.target to obtain the HTML contents of the particular <li> element that was clicked:

**let** nodes = document.getElementsByTagName("li");

**for**(**let** i = 0; i < nodes.length; i++) {

nodes[i].addEventListener("click", **function**(e) {

console.log(e.target.innerHTML);

});

}

We are going to use a similar technique later on, for example to influence the style or display the properties of a particular polygon that is being hovered with the mouse in a web map (Figure [8.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#fig:example-08-07)).

**4.11 Modifying page based on data**

One of the most important use cases of JavaScript is dynamically generating page contents based on *data*. The data we wish to display on the page can come from various sources, such as an object in the JavaScript environment, a file, or a database. JavaScript code can be used to process the data into HTML elements, which can then be added to the DOM (Section [4.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#operating-on-selection)) and consequently displayed in the browser. Modifying page contents based on data is also a use case where iteration (Sections [3.10.3.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#iterative-for-loop-syntax)–[3.10.3.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#the-foreach-array-method)) turns out to be very useful.

Our next example is a simple and small web page, containing just two paragraphs and one unordered list (Figure [4.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-06)). We already know how to write the necessary HTML code for such a web page (Chapter [1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html)). The novel part in example-04-06.html is that we are going to generate some of the HTML code based on data and using JavaScript, rather than have the entire HTML contents predefined.

Let’s assume we need to dynamically create a list of items on our page, and we have an array with the contents that should go into each list item. We will use the .forEach array method (Section [3.10.3.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#the-foreach-array-method)) for iterating over the array and preparing the HTML contents that goes into the <ul> element on the page. For example, suppose we have the following array named data, including the Latin names of eight Oncocyclus *Iris* [species](https://en.wikipedia.org/wiki/Iris_subg._Iris#Oncocyclus) found in Israel:

**let** data = [

"Iris atrofusca",

"Iris atropurpurea",

"Iris bismarckiana",

"Iris haynei",

"Iris hermona",

"Iris lortetii",

"Iris mariae",

"Iris petrana"

];

In the HTML code, we may initially have an empty <ul> *placeholder*:

**<ul** id="species"**></ul>**

which we would like to fill with the <li> HTML elements (Section [1.6.8.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html-ul-ol-li)) based on the above array of species names[45](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn45), as follows:

**<ul** id="species"**>**

**<li><i>**Iris atrofusca**</i></li>**

**<li><i>**Iris atropurpurea**</i></li>**

**<li><i>**Iris bismarckiana**</i></li>**

**<li><i>**Iris haynei**</i></li>**

**<li><i>**Iris hermona**</i></li>**

**<li><i>**Iris lortetii**</i></li>**

**<li><i>**Iris mariae**</i></li>**

**<li><i>**Iris petrana**</i></li>**

**</ul>**

Why not embed the above HTML code directly into the HTML document, instead of constructing it with JavaScript? Two reasons why the former may not always be a good idea:

* The contents can be much longer, e.g., tens or hundreds of elements, which means it is less convenient to type the HTML by hand. We could use tools such as [**Pandoc**](https://pandoc.org/) to programmatically convert text plus markup to HTML, thus reducing the need to manually type all of the HTML tags, but then it is probably more flexible to do the same with JavaScript anyway.
* We may want to build page content based on *real-time* data, loaded each time the user accesses the website, and/or make it responsive to user input. For example, a list of headlines in a news website can be based on a real-time news stories database and/or customized according to user preferences.

To build the above HTML code *programmatically*, based on the data array, we can use *iteration*, such as a for loop (Section [3.10.3.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#standard-for-loop-syntax)–[3.10.3.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#iterative-for-loop-syntax)) or the .forEach array method (Section [3.10.3.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-basics.html#the-foreach-array-method)). In the following example, we will use .forEach.

First, recall that .forEach accepts a function that can do something with the current element the array, subsequently applied on *all* elements. For example, here is how we can sequentially print all array elements in the console:

data.forEach(**function**(element) {

console.log(element);

});

As a result, the following output is printed in the console:

Iris atrofusca

Iris atropurpurea

Iris bismarckiana

Iris haynei

Iris hermona

Iris lortetii

Iris mariae

Iris petrana

Now that we know how to iterate over our array data, we just need to modify the function that is being applied. Instead of just printing the array contents in the console, we want the function to create eight new <li> elements in the DOM. Additionally, we need to create the empty <ul> placeholder element in our document (using HTML) that the <li> elements will be appended to, as shown above.

To create an empty <ul> element, we can add the following HTML code inside the <body>:

**<ul** id="species"**></ul>**

Note that we are using an ID to identify the particular <ul> element, in this case id="species". This is very important since we need to identify the particular element which our JavaScript code will operate on!

Next, to add the a new <li> element inside the <ul> element, we can:

* Start with an empty string
* Append new HTML code with an <li> element per each element in data, using the .forEach iteration, to the string
* Assign the string, now containing eigth <li> elements, to the .innerHTML property of the <ul> element

Here is how we accomplish those steps. The first part, creating the empty string named html is as follows:

**let** html = "";

Second, the .forEach iteration “pastes” the element contents (species name) with the start and end HTML tags, appending it to the html string:

data.forEach(**function**(element) {

html += "<li><i>" + element + "</i></li>";

});

Finally, the complete string is assigned to the .innerHTML property of the <ul> element:

document.getElementById("species").innerHTML = html;

The complete code for dynamic construction of the list looks like this:

**let** html = "";

data.forEach(**function**(element) {

html += "<li><i>" + element + "</i></li>";

});

document.getElementById("species").innerHTML = html;

And here is the complete code of the small web page example-04-06.html, implementing the dynamic creation of an unordered list using JavaScript:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Populating list**</title>**

**</head>**

**<body>**

**<p>**List of rare **<i>**Iris**</i>** species in Israel:**</p>**

**<ul** id="species"**></ul>**

**<p>**It was created dynamically using JavaScript.**</p>**

**<script>**

**let** data = [

"Iris atrofusca",

"Iris atropurpurea",

"Iris bismarckiana",

"Iris haynei",

"Iris hermona",

"Iris lortetii",

"Iris mariae",

"Iris petrana"

];

**let** html = "";

data.forEach(**function**(element) {

html += "<li><i>" + element + "</i></li>";

});

document.getElementById("species").innerHTML = html;

**</script>**

**</body>**

**</html>**

The result is shown in Figure [4.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-06).

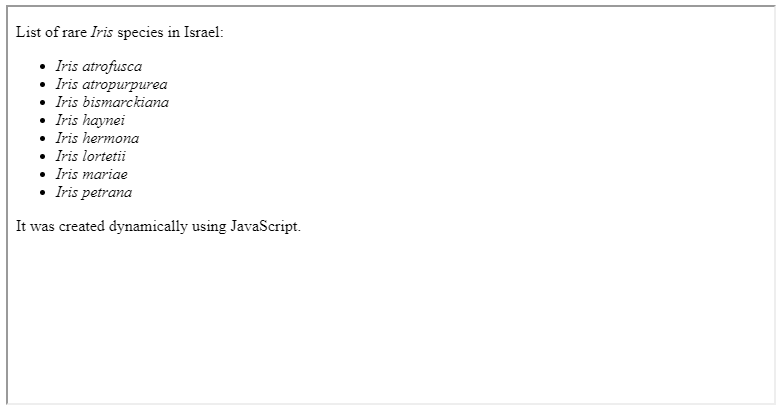


FIGURE 4.7: example-04-06.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-06.html)

Again, in this small example the advantage of generating the list with JavaScript, rather than writing the HTML code by hand, may not be evident. However, this technique is very powerful in more realistic settings, such as when our data is much larger and/or needs to be constantly updated. For example, later on we will use this technique to dynamically generate a dropdown list with dozens of plant species names, according to real-time data coming from a database (Section [10.4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\non-spatial-queries-1.html#adding-options)).

**4.12 Calculator example**

Our last example in this chapter concerns dynamic responses to user input (Section [1.6.13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#input-elements)) on the page. In example-04-07.html, we will build a simple calculator app where clicking on a button multiplies two numbers the user entered, and prints the result on screen (Figure [4.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-07)). In fact, we already covered everything we need to know to write the code for such an application, so example-04-07.html is just to summarize and practice the material one more time.

As usual, we start with HTML code. Importantly, our code contains three <input> elements of type="number" (Section [1.6.13.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#numeric-input)) and type="button" (Section [1.6.13.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#buttons)):

* The first number (type="number")
* The second number (type="number")
* The “Multiply!” button (type="button")

Here is the HTML code for just those three <input> elements:

**<input** type="number" id="num1" min="0" max="100" value="5"**>**

**<input** type="number" id="num2" min="0" max="100" value="5"**>**

**<input** type="button" id="multiply" value="Multiply!"**>**

Note that all input elements are associated with IDs: num1, num2, and multiply. We need the IDs for referring to the particular elements in our JavaScript code. Below the <input> elements, we have an empty <p> element with id="result". This paragraph will hold the multiplication result. Initially, the paragraph is empty, but it will be filled with content using JavaScript code:

**<p** id="result"**></p>**

The only scenario where the contents of the page change is when the user clicks on the “Multiply!” button. Therefore, our <script> is mainly composed of an event listener, responding to "click" events on the id="multiply" button:

document.getElementById("multiply").addEventListener("click", **function**() {...});

The function being passed to the event listener modifies the text contents of the <p> element (result). The inserted text is the multiplication result of the .value properties of both input numbers (num1 and num2). For convenience, we may want to keep the three element references ("num1", "num2", and "result") in predefined variables num1, num2, and result:

**let** num1 = document.getElementById("num1");

**let** num2 = document.getElementById("num2");

**let** result = document.getElementById("result");

document.getElementById("multiply").addEventListener("click", **function**() {

result.innerHTML = "The result is: " + num1.value \* num2.value;

});

It is important to note that all input values are returned as text *strings*, even when we are talking about numeric <input> elements. You can see this by opening example-04-07.html and typing document.getElementById("num1").value in the console. However, when using the multiplication operator \* strings are automatically converted to numbers, which is why multiplication still works[46](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fn46). Here is the complete code of our arithmetic web application, given in example-04-07.html:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Working with user input**</title>**

**</head>**

**<body>**

**<p>**First number

**<input** type="number" id="num1" min="0" max="100" value="5"**>**

**</p>**

**<p>**Second number

**<input** type="number" id="num2" min="0" max="100" value="5"**>**

**</p>**

**<input** type="button" id="multiply" value="Multiply!"**>**

**<p** id="result"**></p>**

**<script>**

**let** num1 = document.getElementById("num1");

**let** num2 = document.getElementById("num2");

**let** result = document.getElementById("result");

document.getElementById("multiply").addEventListener("click", **function**() {

result.innerHTML = "The result is: " + num1.value \* num2.value;

});

**</script>**

**</body>**

**</html>**

The result is shown in Figure [4.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-07).

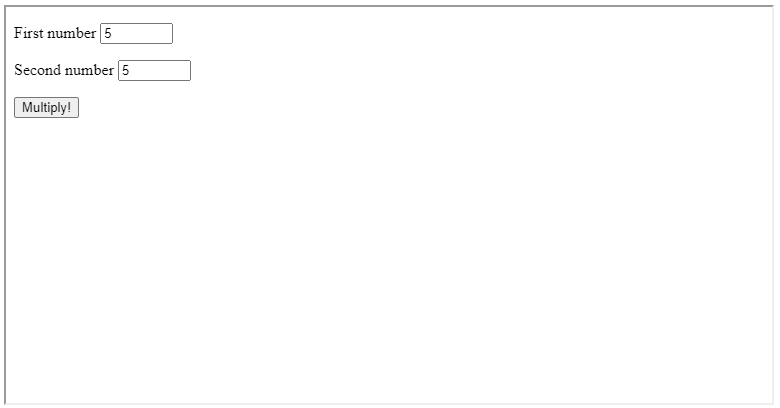


FIGURE 4.8: example-04-07.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-04-07.html)

**4.13 Exercise**

* Modify example-04-07.html by adding the following functionality:
  + Make calculator history append at the bottom, to keep track of the previous calculations. For example, if the user multiplied 5 by 5, then 5 \* 5 = 25 will be added at the bottom of all other history items. If the user made another calculation, it will be added below the previous one, and so on (Figure [4.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:solution-04)).
  + Add a second button for making *division* of the two entered numbers.
  + Add a third button for *clearing* all previous history from screen. (Hint: you can use .innerHTML="" to clear the contents of an HTML element.)
  + 

FIGURE 4.9: solution-04.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\solution-04.html)

1. The document object, like all other global objects, is in fact a property of the window object. That is, document is a shortcut for window.document.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref39)
2. Note the letter s in .getElementsByClassName and in .getElementsByTagName (element**s** = plural), as opposed to .getElementById (element = singular).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref40)
3. We will come back to the subject of specifying file paths in Section [5.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\web-servers-1.html#urls-and-file-structure).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref41)
4. Other commonly used selectors include querySelectorAll, which gets all elements matching the given CSS selector, and querySelector, which gets the just the *first* element matching the given CSS selector.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref42)
5. It is possible to bind two (or more) event listeners to the same selection in the *same expression*, thanks to the fact that the .on method returns the original selection.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref43)
6. A list of all standard event object properties can be found in the *HTML DOM Events* (<https://www.w3schools.com/jsref/dom_obj_event.asp>) reference by W3Schools.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref44)
7. Note that we are also using the <i> element (Section [1.6.6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html-i)) to display species in *italics*.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref45)
8. Try typing "5" \* "5" in the console to see this behavior in action.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fnref46)