**Chapter 6 Leaflet**

**6.1 Introduction**

Now that we have covered the basics of web technologies, we are moving on to the main topic of this book: web mapping. This chapter and the next two (Chapters [6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#leaflet)–[8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#symbology-and-interactivity)) introduce **Leaflet**, a JavaScript library used to create interactive web maps. Using Leaflet, you can create a simple map using just two or three JavaScript expressions, or you can build a complex map using hundreds of lines of code.

In this chapter, we will learn how to initialize a Leaflet web map on our web page, and how to add several types of layers on the map: tile layers (Section [6.5.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-a-tile-layer)) and simple shapes such as point markers (Section [6.6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-markers)), lines (Section [6.6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-lines)), and polygons (Section [6.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-polygons)). We will also learn to add interactive popups for our layers (Section [6.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-popups)) and a panel with a textual description of our map (Section [6.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-a-description)). Finally, we will introduce map events—browser events associated with the web map (Section [6.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#introducing-map-events)).

In the next two Chapters [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1)–[8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#symbology-and-interactivity), we will learn some more advanced Leaflet functionality. In Chapter [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1), we will learn to add complex shapes coming from external GeoJSON files. Then, in Chapter [8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#symbology-and-interactivity), we will learn how to define symbology and interactive behavior in our web map.

**6.2 What is a web map?**

We already introduced the concept of web mapping in Section [0.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\index.html#what-is-web-mapping). We mentioned that the term “web map” usually implies a map that is not simply *on* the web, but rather one that is *powered* by the web. It is usually interactive, and not always self-contained, meaning that it “pulls” content from other locations, such as tile layer servers (Section [6.5.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tile-layer-providers)) or database APIs (Section [9.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\databases.html#the-carto-sql-api))[52](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn52).

Similarly to spatial information displayed in [GIS software](https://en.wikipedia.org/wiki/Geographic_information_system), web maps are usually composed of one or more **layers**. Web map layers can be generally classified into two categories:

* **Background** layers, or basemaps, comprising collections of gridded images or vector tiles, which are usually general-purpose and not prepared specifically for our map
* **Foreground** layers, or overlays, which are usually vector layers (points, lines, and polygons), commonly prepared and/or fine-tuned for the specific map web where they are shown

Background layers are usually static and non-interactive. Conversely, foreground layers are usually dynamic and associated with user interaction, such as the ability to query layer attributes by clicking on a feature (Section [6.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-popups)).

**6.3 What is Leaflet?**



[**Leaflet**](https://leafletjs.com/) ([Crickard III 2014](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-crickard2014leaflet)) is an open-source JavaScript library for building interactive web maps. Leaflet was initially released in 2011 (Table [6.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tab:web-mapping-libraries)). It is lightweight, relatively simple, and flexible. For these reasons, Leaflet is probably the most popular *open-source* web-mapping library at the moment. As the Leaflet home page puts it, the guiding principle behind this library is simplicity:

“*Leaflet doesn’t try to do everything for everyone. Instead it focuses on making the basic things work perfectly.*”

Advanced functionality is still available through Leaflet [plugins](https://leafletjs.com/plugins.html). Towards the end of the book, we will learn about two Leaflet plugins: **Leaflet.heat** (Section [12.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\client-side-geoprocessing.html#heatmaps-with-leafletheat)) and **Leaflet.draw** (Section [13.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\collaborative-mapping.html#the-drawing-control)).

**6.4 Alternatives to Leaflet**

In this book, we will exclusively use Leaflet for building web maps. However, it is important to be aware of the landscape of alternative web-mapping libraries, their advantages and disadvantages. Table [6.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tab:web-mapping-libraries) lists Leaflet along with other popular JavaScript libraries for web mapping[53](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn53).

| TABLE 6.1: Popular web-mapping libraries | | | |
| --- | --- | --- | --- |
| **Library** | **Released** | **Type** | **URL** |
| Google Maps | 2005 | Commercial | <https://developers.google.com/maps/> |
| OpenLayers | 2006 | Open-source | <https://openlayers.org/> |
| ArcGIS API for JS | 2008 | Commercial | <https://developers.arcgis.com/javascript/> |
| Leaflet | 2011 | Open-source | <https://leafletjs.com/> |
| D3 | 2011 | Open-source | <https://d3js.org/> |
| Mapbox GL JS | 2015 | Commercial | <https://www.mapbox.com/mapbox-gl-js/api/> |



[**Google Maps JavaScript API**](https://developers.google.com/maps/documentation/javascript/) ([Dincer and Uraz 2013](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-dincer2013google)) is a proprietary web-mapping library by Google. The biggest advantage of the Google Maps API is that it brings the finely crafted look and feel of the Google Maps background layer to your own web map. On the other hand, background layers other than Google’s are not supported. The Google Maps API also has advanced functionality not available elsewhere, such as integration with [Street View](https://developers.google.com/maps/documentation/javascript/examples/streetview-service) scenes. However the library is closed-source, which means the web maps cannot be fully customized. Also, it requires a [paid subscription](https://developers.google.com/maps/documentation/javascript/usage-and-billing).



[**OpenLayers**](https://openlayers.org/) ([Gratier, Spencer, and Hazzard 2015](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-gratier2015openlayers); [Langley and Perez 2016](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-langley2016openlayers); [Farkas 2016](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-farkas2016mastering)) is an older, more mature, and more richly featured open-source JavaScript library for building web maps, otherwise very similar to Leaflet in its scope. However, OpenLayers is also more complex, heavier (in terms of JavaScript file size), and more difficult to learn. Leaflet can be viewed as a lighter and more focused alternative to OpenLayers.



[**ArcGIS API for JavaScript**](https://developers.arcgis.com/javascript/) ([Rubalcava 2015](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-rubalcava2015arcgis)) is another commercial web-mapping solution. The ArcGIS API is primarily designed to be used with services published using [ArcGIS Online](https://www.esri.com/software/arcgis/arcgisonline) or [ArcGIS Server](http://server.arcgis.com/en/server/latest/get-started/windows/what-is-arcgis-for-server-.htm), though general data sources can also be used. The ability of the ArcGIS API to tap into web services originating from ArcToolbox that perform geoprocessing on the server is a feature which has no good equivalent in open-source software. However, using an ArcGIS Server requires a paid (expensive) license. Also, the APIs are free to use for development or educational use, but require a fee for commercial use.



[**D3**](https://d3js.org/) ([Murray 2017](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-murray2017interactive); [Janert 2019](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-janert2019d3)) is an open-source JavaScript library for [visualization](https://bl.ocks.org/mbostock) in general, though it is commonly used for [mapping](https://d3indepth.com/geographic/) too ([Newton and Villarreal 2014](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#ref-newton2014learning)). D3 is primarily used for displaying vector layers, as raster tile layers are not well supported. D3 is probably the most complex and difficult to learn among the libraries listed in Table [6.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tab:web-mapping-libraries). However, it is very flexible and can be used to create truly innovative map designs. Go back to the examples from Section [0.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\index.html#what-is-web-mapping)—some of the most impressive ones, such as [*Earth Weather*](https://earth.nullschool.net/), were created with the help of D3.



[**Mapbox GL JS**](https://docs.mapbox.com/mapbox-gl-js/api/) is a web-mapping library provided by a commercial company named Mapbox. Notably, Mapbox GL JS uses customizable vector tile layers (Section [6.5.10.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#vector-tiles)) as background. You can use existing basemaps, or build your own using an interactive [“studio”](https://www.mapbox.com/mapbox-studio/) web application. Like Google Maps, Mapbox GL JS also requires a [paid subscription](https://www.mapbox.com/pricing/), though the first 50,000 monthly map views are free.

**6.5 Creating a basic web map**

**6.5.1 Overview.**In this section, we will learn to create a basic web map using Leaflet. The map is going to contain a single background (tile) layer, initially zoomed-in on the Ben-Gurion University. The final result is shown in Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02).

**6.5.2 Web page setup.**We start with the following minimal HTML document, which we are familiar with from Chapter [1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html):

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Basic map**</title>**

*<!-- More content will go here -->*

**</head>**

**<body>**

*<!-- More content will go here -->*

**</body>**

**</html>**

* Create an empty text file named index.html and copy the above code into that file.
* Follow the steps described in the Sections [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#creating-a-basic-web-map)–[6.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-a-description), to build your own web map from the ground up.

At this stage, the web page is empty. From here, we will do the following four things to add a map to our page:

* Include the Leaflet CSS file using a <link> element and the Leaflet JavaScript file using a <script> element (Section [6.5.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#javascript-libraries)–[6.5.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#including-leaflet-css-js))
* Add a <div> element that will hold the interactive Leaflet map (Section [6.5.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-map-div))
* Add another, custom <script>, to create a map object and initiate the map inside the <div> element (Section [6.5.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#creating-map-object))
* Add the tile OpenStreetMap basemap to our map using L.tileLayer (Sections [6.5.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#what-are-tile-layers)–[6.5.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tile-layer-providers))

**6.5.3 JavaScript libraries.**A JavaScript [**library**](https://en.wikipedia.org/wiki/JavaScript_library) is a collection of JavaScript code, which allows for easier development of JavaScript-based applications. There are a lot of JavaScript libraries that simplify common tasks (e.g., DOM manipulation) or specialized tasks (e.g., web mapping), to make life easier for web developers. Often, you will be working with a library that is already written, instead of “reinventing the wheel” and writing your own JavaScript code for every single task.

**6.5.4 Including a library.**Before using any object, function, or method from Leaflet, the library needs to be **included** in our web page. Practically, this means that the Leaflet script is being run on page load, defining Leaflet objects, functions, and methods, which you can then use in the subsequent scripts that are executed on that page.

To include the Leaflet library—or any other script for that matter—we need to place a <script> element referring to that script in our HTML document. Scripts for loading libraries are commonly placed inside the <head> of our document. Placing a <script> in the <head> means that the script is loaded before anything visible (i.e., the <body>) is loaded. This can be safer and easier for maintenance, yet with some performance implications. Namely, page load is being halted until the <scripts> elements have been processed. Since the Leaflet script—like most JavaScript library scripts—is small (~150 kB), there is no noticeable delay for downloading and processing it, so we can safely place it in the <head>.

As mentioned in Section [1.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#styling-and-scripts), when using the <script> element to load an external script file, we use the src attribute to specify file location. The location specified by src can be either a path to a local file (Section [6.5.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#loading-a-local-script)), or a URL of a remote file hosted elsewhere on the web (Section [6.5.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#loading-a-remote-script)).

**6.5.5 Loading a local script.**When loading a script from a **local file**, we need to have an actual copy of the file on our server (Section [5.5.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\web-servers-1.html#css-and-javasccript)). Basically, we need to download the Leaflet code file, e.g., from the [download section](https://leafletjs.com/download.html) on the official Leaflet website, and save it along with our HTML file. In case the Leaflet script we downloaded is named leaflet.js, and assuming the script is placed in the same folder where the HTML document is, the first few lines of the document may look as follows:

<!DOCTYPE html>

**<html>**

**<head>**

**<script** src="leaflet.js"**></script>**

**</head>**

...

**6.5.6 Loading a remote script.**When loading a script from a **remote file**, hosted on the web in some location other than our own computer, we need to provide the file URL. A reliable option is to use a [**Content Delivery Network (CDN)**](https://en.wikipedia.org/wiki/Content_delivery_network), such as [UNPKG](https://unpkg.com/). A CDN is a series of servers designed to serve static files very quickly. In case we are loading the Leaflet library from UNPKG, the first few lines of the document may look as follows:

<!DOCTYPE html>

**<html>**

**<head>**

**<script** src="https://unpkg.com/[...]/leaflet.js"**></script>**

**</head>**

...

The src attribute value is truncated with [...] to save space. Here is the complete URL that needs to go into the src attribute value:

[https://unpkg.com/leaflet@1.7.1/dist/leaflet.js](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\unpkg.com\leaflet%401.7.1\dist\leaflet.js)

* Browse to the above URL to view the Leaflet library contents. You can also download the file by clicking **Save as…** to view it in a text editor instead of the browser.
* You will note that the code is formatted in a strange way, as if the entire content is in a single line, with very few space and new line characters. This makes it hard for us to read the code. Can you guess what is the reason for this type of formatting?

**6.5.7 Including Leaflet CSS and JavaScript.**We need to include the Leaflet library on our web page before we can start using it. As mentioned above (Sections [6.5.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#loading-a-local-script)–[6.5.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#loading-a-remote-script)), there are two options for doing this. We can either download the library files from the Leaflet website and load those local files, or we can use a hosted version of the files from a CDN. In the examples, we use the *local copy* option.

Many JavaScript libraries, such as the Turf.js library we will learn about later on (Section [12.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\client-side-geoprocessing.html#geoprocessing-with-turfjs)), consist of just one .js file. The Leaflet library is slightly more complex: it consists of two files, a JavaScript file and a CSS file, plus several image files. The image files are used by the Leaflet script, for example to display markers on the map (Section [11.2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#adding-custom-marker)).

The Leaflet JavaScript and CSS files are included just like any other JavaScript and CSS files. To include the Leaflet CSS file, we add a <link> element (Section [2.7.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#external-css)) referring to the file within the <head> section (after the <title>). We will use a local copy named leaflet.css placed inside a sub-directory called css:

**<link** rel="stylesheet" href="css/leaflet.css"**>**

Remember that the path to the local file needs to correspond to the website directory structure (Section [5.5.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\web-servers-1.html#file-structure)). For example, in the above <link> element, we are using a relative path: css/leaflet.css. This means that the leaflet.css file is located in a sub-directory named css, inside the directory where the HTML document is. For loading the file from a CDN (Section [6.5.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#loading-a-remote-script)), we could replace the css/leaflet.css part with the following URL:

[https://unpkg.com/leaflet@1.7.1/dist/leaflet.css](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\unpkg.com\leaflet%401.7.1\dist\leaflet.css)

After[54](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn54) including the Leaflet CSS file, we need to add a <script> element referring to the Leaflet JavaScript file. We are going to load a local copy named leaflet.js placed inside a sub-directory named js. This means we need to put the following <script> element inside the <head>:

**<script** src="js/leaflet.js"**></script>**

The path js/leaflet.js refers to a file named leaflet.js inside a sub-directory named js. For loading the Leaflet JavaScript file from a CDN, replace js/leaflet.js with the following URL which was already shown above (Section [6.5.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#loading-a-remote-script)):

[https://unpkg.com/leaflet@1.7.1/dist/leaflet.js](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\unpkg.com\leaflet%401.7.1\dist\leaflet.js)

Either way, after adding the above <link> and <script> elements, the Leaflet CSS and JavaScript files will be loaded and processed by the browser, which means we can use its functions and methods in other scripts on that page. Note that the local files provided in the online supplement, as well as the above remote URLs, refer to a specific version of Leaflet—namely Leaflet version 1.7.1—which is the newest version at the time of writing (Table [0.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\index.html#tab:javascript-libraries)). In case we need to load a newer or older version, the local copies or URLs can be modified accordingly.

When working with a local copy of the Leaflet library, in addition to the JavaScript and CSS files we also need to create an images directory within the directory where our CSS file is (e.g., css), and place several PNG image files[55](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn55) there, such as marker-icon.png and marker-shadow.png. These files are necessary for displaying markers and other images on top of our map (Section [6.6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-markers)). More information on how Leaflet actually uses those PNG images will be given in Section [11.2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#adding-custom-marker).

**6.5.8 Adding map <div>.**Our next step is to add a <div> element, inside the <body>. The <div> will be used to hold the interactive map. As we learned in Section [1.6.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#grouping), a <div> is a generic grouping element. It is used to collect elements into the same block group so that it can be referred to in CSS or JavaScript code. The <div> intended for our map is initially empty, but needs to have an ID. We will use JavaScript to “fill” this element with the interactive web map, later on:

**<div** id="map"**></div>**

In case we want the web map to cover the entire screen (e.g., Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02)), which is what we will usually do in this book, we also need the following CSS rules:

body {

**padding**: 0;

**margin**: 0;

}

html, body, #map {

**height**: 100%;

**width**: 100%;

}

This CSS code can be added in the <style> element in the <head>. Recall that this method of adding CSS is known as *embedded* CSS (Section [2.7.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#embedded-css)).

**6.5.9 Creating a map object.**Now that the Leaflet library is loaded, and the <div> element which will be used to contain it is defined and styled, we can move on to actually adding the map, using JavaScript. The Leaflet library defines a global object named L, which holds all of the Leaflet library functions and methods. Using the L.map method, our first step is to create a map object. When creating a map object there are two important arguments supplied to L.map:

* The **ID** of the <div> element where the map goes in
* Additional map **options**, passed as an object

In our case, since the <div> intended for our map has id="map" (Section [6.5.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-map-div)), we can initiate the map with the following expression, which goes inside a <script> element at the end of the <body>:

L.map("map");

As for additional map options[56](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn56), there are several ones that we can set. The options are passed together, as a single object, where property name refers to the option and the property value refers to the value which we want to set. The most essential options of L.map are those specifying the initially viewed map extent. One way to specify the initial extent is using the center and zoom option, passing the coordinates where the map is initially panned to and its initial zoom level, respectively. For example, to focus on the Ben-Gurion University we can indicate the [31.262218, 34.801472] location and set the zoom level to 17. The L.map expression now takes the following form:

L.map("map", {center: [31.262218, 34.801472], zoom: 17});

Note that the center option is specified in **geographic coordinates** (longitude and latitude) using an array of the form [lat, lon] rather than [lon, lat], that is, in the [Y, X] rather than the [X, Y] order. This may seem unintuitive to GIS users, but the [lat, lon] ordering is actually very common in many applications that are not specifically targeted to geographers, including Leaflet and other web-mapping libraries such as the [Google Maps API](https://developers.google.com/maps/documentation/javascript/adding-a-google-map). When working with Leaflet, you need to be constantly aware of the convention to use [lat, lon] coordinates, to avoid errors such as displaying the wrong area (also see Section [7.3.2.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#general-structure)).

One more thing to keep in mind regarding coordinates is that most web mapping libraries, including Leaflet, usually work with geographic coordinates (longitude, latitude) only as far as the *user* is concerned. This means that all placement settings (such as map center), as well as layer coordinates (Section [6.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-vector-layers)), are passed to Leaflet functions as geographic coordinates (latitude, longitude), i.e., the [**WGS84**](https://en.wikipedia.org/wiki/World_Geodetic_System) (EPSG:4326) [coordinate reference system](https://en.wikipedia.org/wiki/Spatial_reference_system). The map itself, however, is displayed using coordinates in a different system, called the [**Web mercator**](https://en.wikipedia.org/wiki/Web_Mercator_projection) projection (EPSG:3857) (Figure [6.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:EPSG-4326-3857)). For example, the following [lon, lat] coordinates (EPSG:4326) for central London:

[-0.09, 51.51]

will be internally converted to the following [X, Y] coordinates in Web mercator (EPSG:3857) before being displayed on screen:

[-10018.75, 6712008]

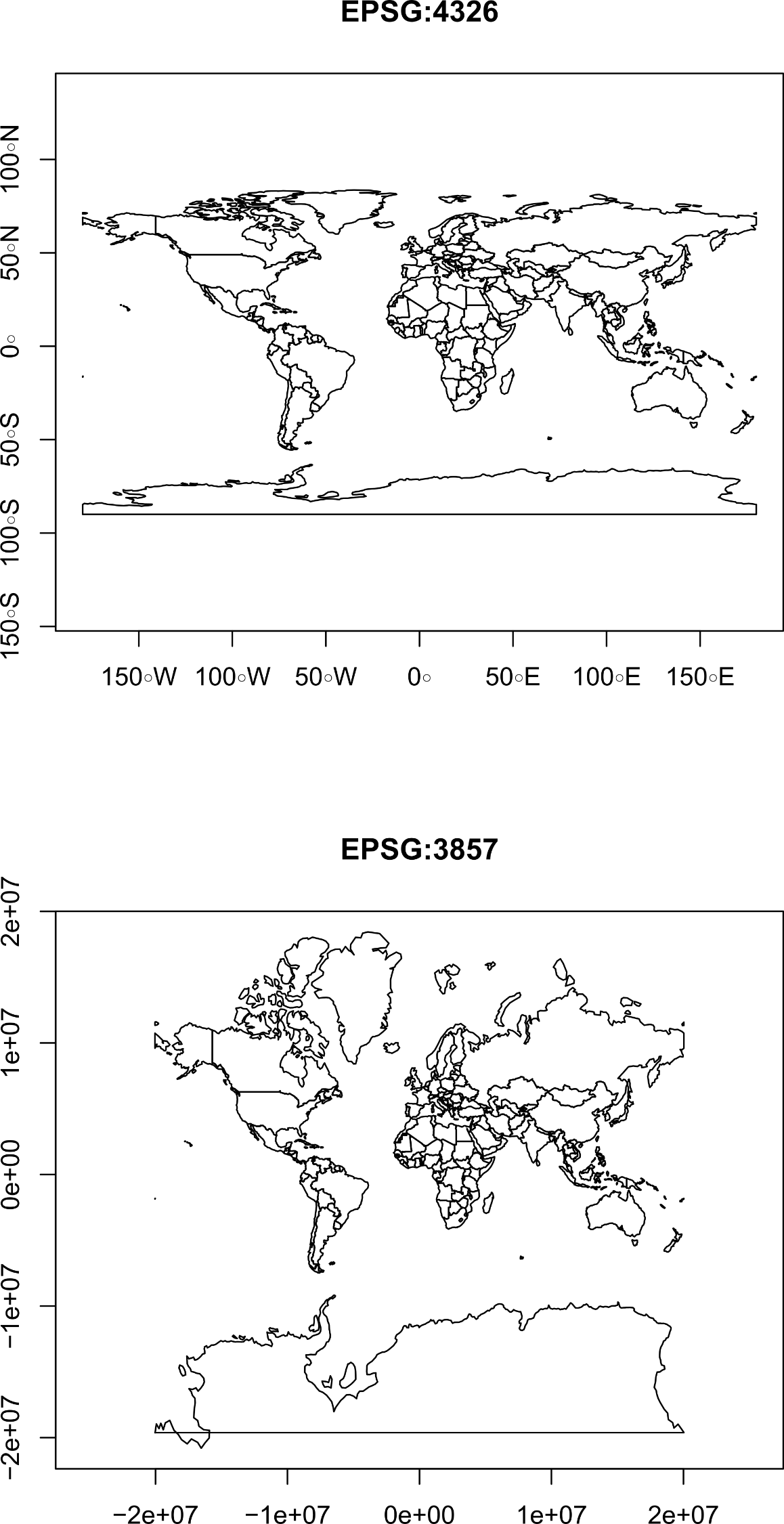


FIGURE 6.1: World map in the WGS84 (EPSG:4326) and Web mercator (EPSG:3857) projections

The user never has to deal with the Web mercator system, since it is only used *internally*, before drawing the final display on screen. However, it is important to be aware that it exists. As for the zoom, you may wonder what does the 17 zoom level mean. This will be explained when discussing tile layers (Section [6.5.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#what-are-tile-layers)).

The L.map function returns a **map object**, which has useful methods and can be passed to methods of other objects (such as .addTo, see below). Therefore, we usually want to save that object in a variable, such as the one named map in the following expression, so that we can refer to it when running those methods later on in our script. Combining all of the above considerations, we should now have the following JavaScript code in the <script> element in our web page:

**let** map = L.map("map", {center: [31.262218, 34.801472], zoom: 17});

If you open the map at this stage, you should see that it has no content, just grey background with the “+” and “-” (zoom-in and zoom-out) buttons, which are part of the standard map interface. Our next step is to add a background layer (Section [6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#what-is-web-map)) on the map.

**6.5.10 What are tile layers?**

**6.5.10.1 Overview.**Tile layers are a fundamental technology behind web maps. They comprise the *background* layer in most web maps, thus helping the viewer to locate the *foreground* layers in geographical space. The word *tile* in tile layers comes from the fact that the layer is split into individual rectangular tiles. Tile layers come in two forms, which we are going to cover next: raster tiles (Section [6.5.10.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#raster-tiles)) and vector tiles (Section [6.5.10.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#vector-tiles)).

**6.5.10.2 Raster tiles.**The oldest and simplest tile layer type is where tiles are raster images, also known as [**raster tiles**](https://en.wikipedia.org/wiki/Tiled_web_map). With raster tiles, tile layers are usually composed of PNG images. Traditionally, each PNG image is \(256\times 256\) pixels in size. A separate collection of tiles is required for each zoom level the map can be viewed on, with increasing numbers of tiles needed to cover a (global) extent in higher zoom levels. Conventionally, at each sequential zoom level, all of the tiles are divided into four “new” ones (Figure [6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:tile-zoom-levels)). For example, for covering the world at zoom level 0, we need just one tile. When we go to zoom level 1, that individual tile is split to \(2\times 2=4\) separate tiles. When we go further to zoom level 2, each of the four tiles is also split to four, so that we already have \(4\times 4=16\) separate tiles, and so on. In general, a global tile layer at zoom level \(z\) contains \(2^z\times 2^z=4^z\) tiles. At the default maximal zoom level in a Leaflet map (18), we need 68,719,476,736 tiles to cover the earth[57](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn57)!

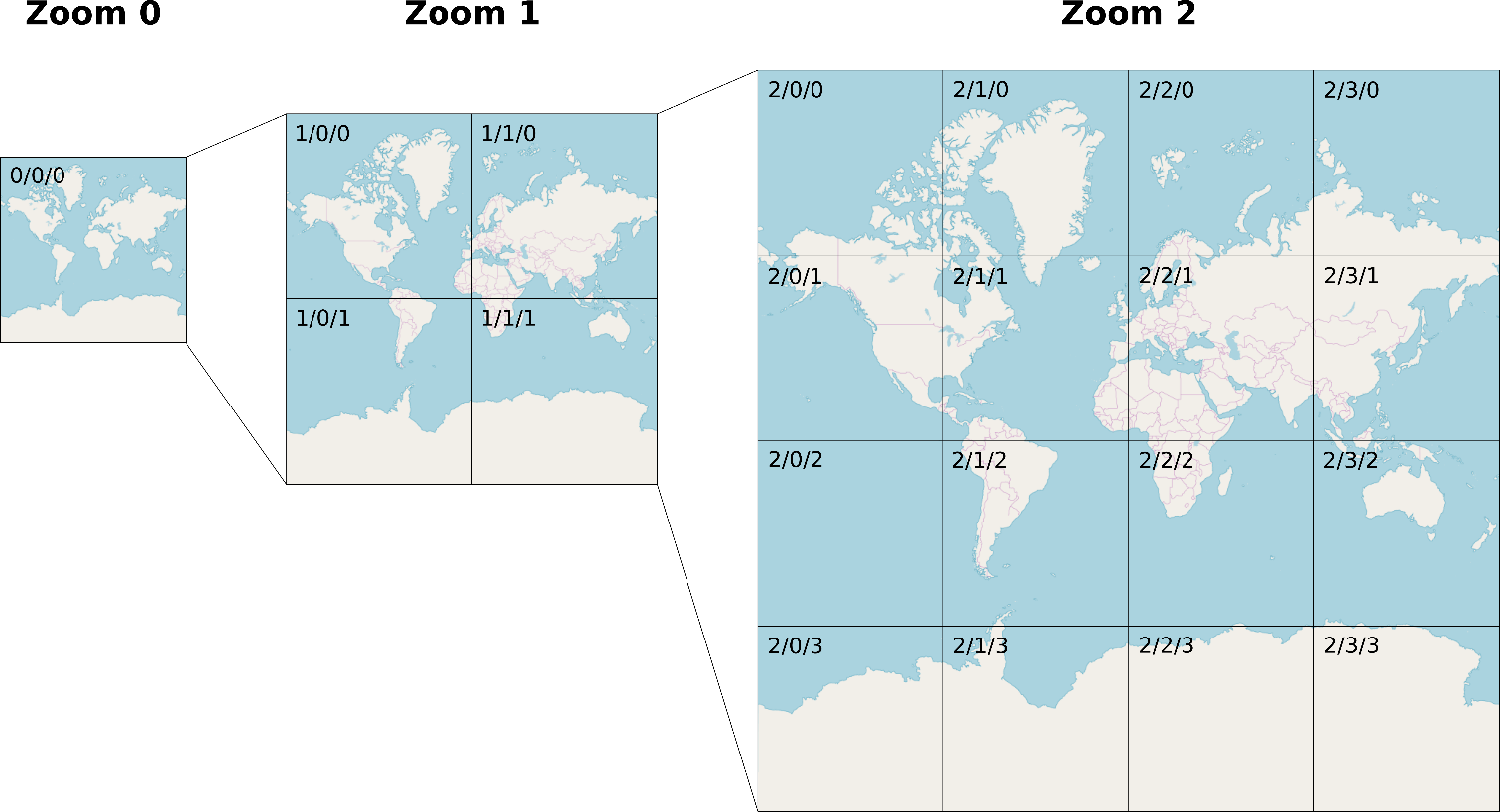


FIGURE 6.2: OpenStreetMap tiles for global coverage at zoom levels 0, 1, and 2. The Z/X/Y values (zoom/column/row) of each tile are shown in the top left corner.

The main rationale for tiled maps is that only the relevant content is loaded in each particular viewed extent. Namely, only those specific tiles that cover the particular extent we are looking at, in a particular zoom level, are transferred from the server. For example, just one or two dozen individual tiles (Figure [6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:individual-tile)) are typically needed to cover any given map extent, such as the one shown in Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02). This results in minimal size of transferred data, which makes tiled maps appear smooth and responsive.

The PNG images of all required tiles are sent from a **tile server**, which is simply a static server (Section [5.4.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\web-servers-1.html#static-servers)) where all tile images are stored in a particular directory structure[58](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn58). To load a particular tile, we enter a URL such as http://.../Z/X/Y.png, where http://.../ is a constant prefix, Z is the zoom level, X is the column (i.e., the x-axis) and Y is the row (i.e., the y-axis) (Figure [6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:tile-zoom-levels)). For example, the following URL refers to an individual tile at zoom level 17, column 78206 and row 53542, focused on building #72 at the Ben-Gurion University, from the [standard OpenStreetMap tile server](https://wiki.openstreetmap.org/wiki/Tile_servers) (Figure [6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:individual-tile)).

[https://a.tile.openstreetmap.org/17/78206/53542.png](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\a.tile.openstreetmap.org\17\78206\53542.png)

Note how the URL for this tile is structured. The constant prefix of the URL is https://a.tile.openstreetmap.org/, referring to the OpenStreetMap tile server. The variable parts are specific values for Z (e.g., 17), X (e.g., 78206), and Y (e.g., 53542). You may recognize where this tile fits in the map view in Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02).

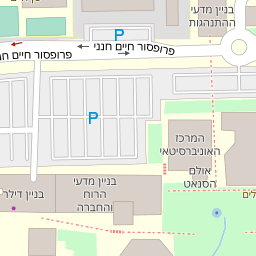


FIGURE 6.3: Individual OpenStreetMap raster tile, from zoom level 17, column 78206 and row 53542 (downloaded on 2019-03-29)

* Open the above URL in the browser to view the PNG image of the specific tile.
* Modify the Z, X, and Y parts of the URL to view other tiles.

**6.5.10.3 Vector tiles.**A more recent tile layer technology is where tiles are *vector* layers, rather than PNG images, referred to as [**vector tiles**](https://en.wikipedia.org/wiki/Vector_tiles). Vector tiles are distinguished by the ability to rotate the map while the labels keep their horizontal orientation, and by the ability to zoom in or out smoothly—without the strict division to discrete zoom levels that raster tile layers have (Figure [6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:tile-zoom-levels)). Major advantages of vector tiles are their smaller size and flexible styling. For example, [Google Maps](https://maps.google.com/) made the transition from raster to vector tiles [in 2013](https://en.wikipedia.org/wiki/Vector_tiles#Services_and_applications).

The Leaflet library does not natively support vector tiles, though there is a plugin called [**Leaflet.VectorGrid**](https://github.com/Leaflet/Leaflet.VectorGrid) for that. Therefore, in this book we will restrict ourselves to using raster tiles as background layers. There are other libraries specifically built around vector tile layers, such as the **Google Maps API** and **Mapbox GL JS**, which we mentioned previously (Section [6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#alternatives-to-leaflet)). The example-06-01.html shows a web map with a vector tile layer built with Mapbox GL JS (Figure [6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-01)). This is the only non-Leaflet web-map example that we are going to see in the book; it is provided for demonstration and comparison of raster and vector tiles.

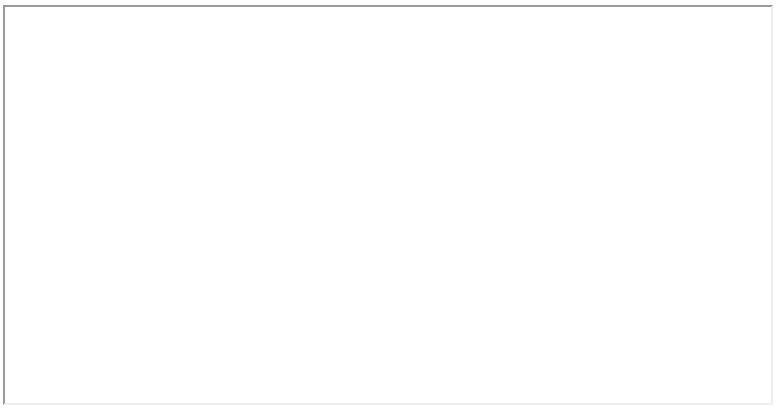


FIGURE 6.4: example-06-01.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-01.html)

* Open example-06-01.html in the browser.
* Try changing map perspective by dragging the mouse while pressing the **Ctrl** key.
* Compare this type of interaction with that of example-06-02.html.
* What is the reason for the differences?

**6.5.11 Adding a tile layer.**We now go back to discussing Leaflet and raster tile layers. Where can we get a tile layer from? There are many tile layers prepared and provided by several organizations, available on dedicated servers (Section [6.5.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tile-layer-providers)) that you can add to your maps. Most of them are based on OpenStreetMap data (Section [13.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\collaborative-mapping.html#crowdsourcing)), because it is the most extensive free database of map data with global coverage. The tile layer we use in the following examples, and the one that the tile shown in Figure [6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:individual-tile) comes from, is developed and maintained by OpenStreetMap itself. It is the default tile layer displayed on the <https://www.openstreetmap.org/> website.

To add a tile layer to a Leaflet web map, we use the L.tileLayer function. This function accepts:

* The **URL** of the tile server
* An object with additional **options**

Any raster tile server URL includes the {z}, {x}, and {y} placeholders, which are internally replaced by zoom level, column, and row each time the Leaflet library loads a given tile, as discussed previously (Section [6.5.10.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#raster-tiles)). The additional {s} placeholder refers to one of the available sub-domains (such as a, b, c), for making parallel requests to different servers hosting the same tile layer. Here is the URL for loading the default OpenStreetMap tile layer:

https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png

The options object is used to specify other parameters related to the tile layer, such as the required attribution (shown in the bottom-right corner on the map), or the minZoom and maxZoom levels[59](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn59).

For example, the following L.tileLayer call defines a tile layer using the default OpenStreetMap server and sets the appropriate attribution. The .addTo method is then applied to *add* the tile layer to the map object, referring to our web map defined earlier (Section [6.5.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#creating-map-object)). The .addTo method is applicable for adding any type of layer on our map, not just a tile layer, as we will see later on (Sections [6.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-vector-layers)–[6.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-a-description)). In the attribution option, note that &copy; is a special HTML character entity to display the copyright symbol (©):

L.tileLayer(

"https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",

{attribution: "&copy; OpenStreetMap"}

).addTo(map);

The attribution we used:

"&copy; OpenStreetMap"

can also be replaced with the following alternative version, where the word “OpenStreetMap” becomes a link (Section [1.6.9.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html-a)) to the OpenStreetMap [Copyright and License](https://www.openstreetmap.org/copyright) web page (see bottom-right corner in Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02)):

'&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>'

Now that the tile layer is in place, one last thing we are going to add to our first Leaflet map is the following <meta> element (Section [1.6.3.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#html-meta)) in the document <head>:

**<meta** name="viewport" content="width=device-width,

initial-scale=1.0, maximum-scale=1.0, user-scalable=no"**>**

This <meta> element disables unwanted scaling of the page when on mobile devices[60](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn60). Without it, map symbols and controls will appear too small when viewed using a browser on a mobile device. Finally, our complete code for a basic Leaflet web map (example-06-02.html) is:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Basic map**</title>**

**<meta** name="viewport" content="width=device-width,

initial-scale=1.0, maximum-scale=1.0, user-scalable=no"**>**

**<link** rel="stylesheet" href="css/leaflet.css"**>**

**<script** src="js/leaflet.js"**></script>**

**<style>**

body {

**padding**: 0;

**margin**: 0;

}

html, body, #map {

**height**: 100%;

**width**: 100%;

}

**</style>**

**</head>**

**<body>**

**<div** id="map"**></div>**

**<script>**

**let** map = L.map("map", {center: [31.262218, 34.801472], zoom: 17});

L.tileLayer(

"https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",

{attribution: '&copy; <a href="http://' +

'www.openstreetmap.org/copyright">OpenStreetMap</a>'}

).addTo(map);

**</script>**

**</body>**

**</html>**

The resulting map is shown in Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02).

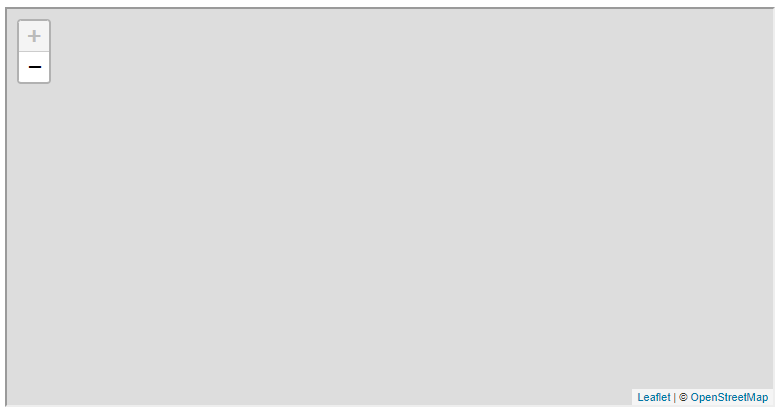


FIGURE 6.5: example-06-02.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-02.html)

* Using the developer tools, we can actually observe the different PNG images of each tile being loaded while navigating on the map.
* Open example-06-02.html in the browser.
* Open the developer tools (in Chrome: by pressing **F12**).
* Go to the **Network** tab of the developer tools.
* Pan and/or zoom-in and/or zoom-out to change to a different viewed extent.
* In the Network tab, you should see a list of the new PNG images being transferred from the server, as new tiles are being shown on the map (Figure [6.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:tiles-network-traffic)).
* Double-click on any of the PNG image file names to show the image itself.

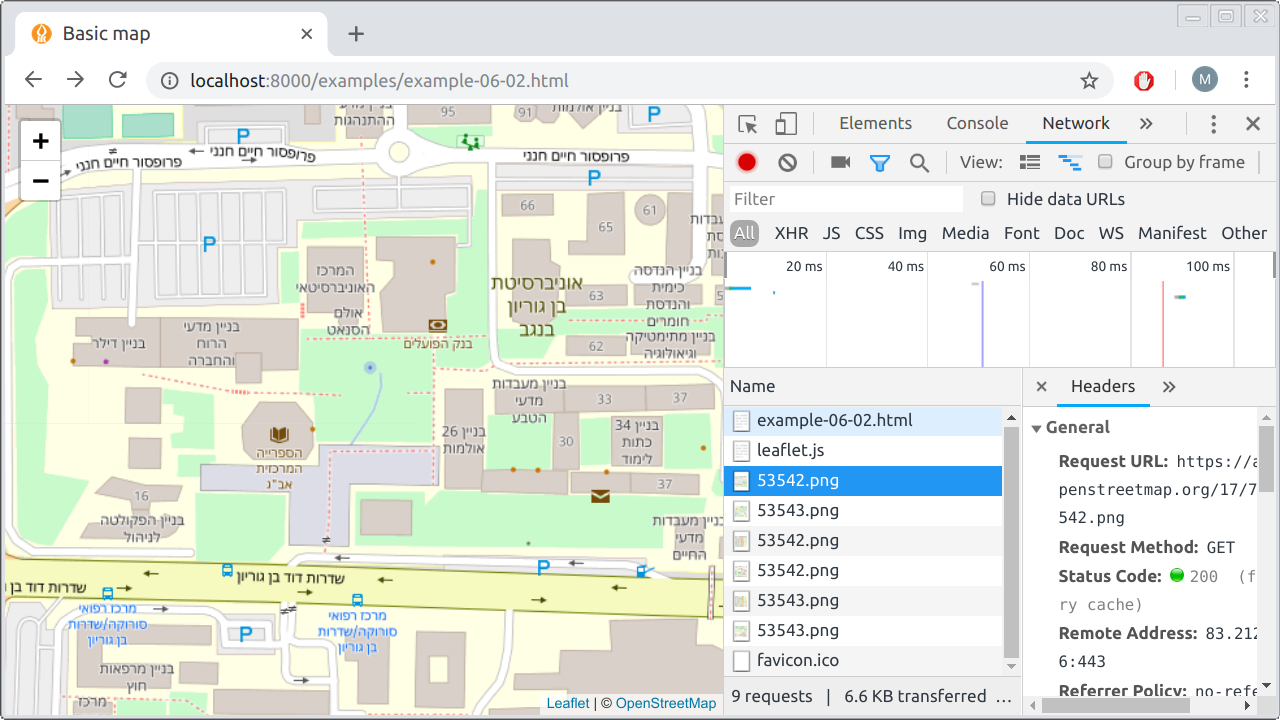


FIGURE 6.6: Observing network traffic as new tiles are being loaded into a Leaflet map

**6.5.12 Tile layer providers.**As mentioned previously (Section [6.5.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-a-tile-layer)), the default OpenStreetMap tile layer we just added on our map (Figure [6.5](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-02)) is one of many available tile layers provided by different organizations. An interactive demonstration of various popular tile layer providers can be found in the [*Leaflet Provider Demo*](https://leaflet-extras.github.io/leaflet-providers/preview/) web page. Once on the leaflet-providers web page, you can select a tile layer from the list in the right panel, then explore its preview in the central panel (Figure [6.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:leaflet-providers)). Conveniently, the text box on the top of the page gives the exact L.tileLayer JavaScript expression that you can use in a <script> for loading the tile layer in a Leaflet map.

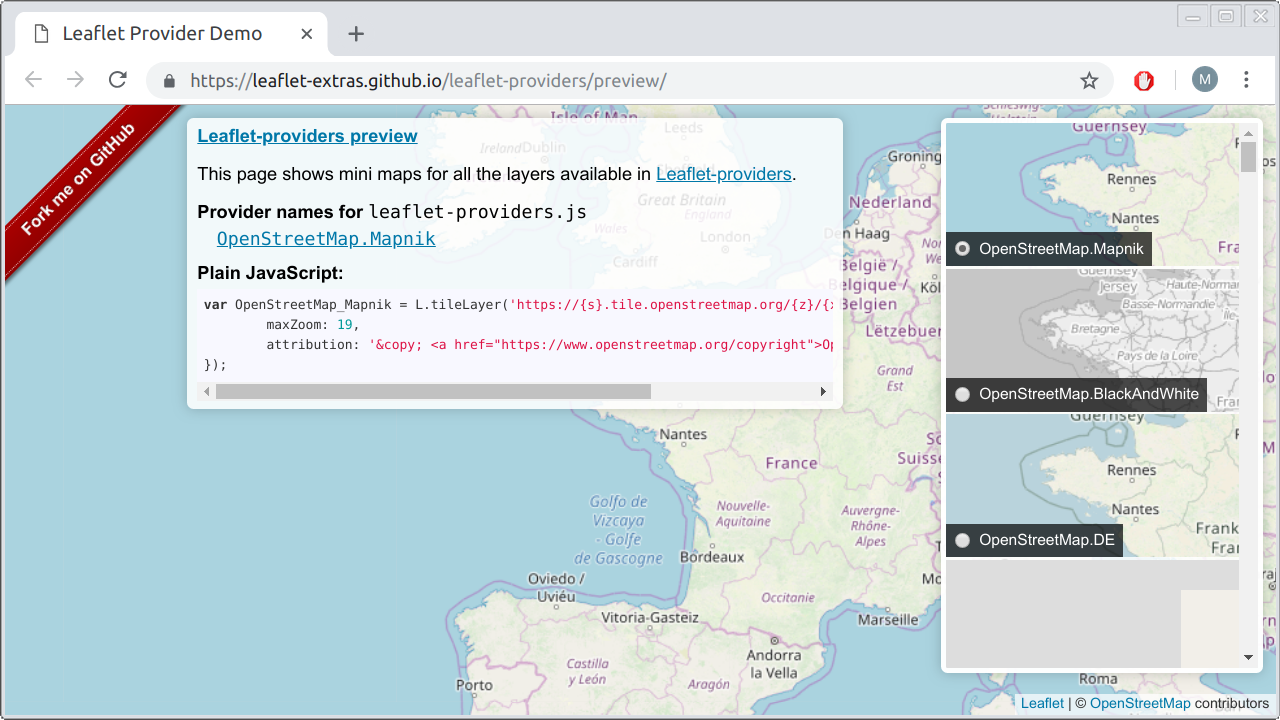


FIGURE 6.7: Interactive preview of [tile layer providers](https://leaflet-extras.github.io/leaflet-providers/preview/)

* Browse to the leaflet-providers page on <https://leaflet-extras.github.io/leaflet-providers/preview/> (Figure [6.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:leaflet-providers)) and choose a tile layer you like.
* Replace the L.tileLayer(...) expression in the source code of example-06-02.html (after making a local copy) with the appropriate expression for your chosen tile layer.
* Save the file and refresh the page in the browser.
* You should now see the new tile layer you chose on the map!

**6.6 Adding vector layers**

**6.6.1 Overview.**So far we learned how to create a Leaflet map (Sections [6.5.1](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#overview-creating-a-basic-web-map)–[6.5.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#creating-map-object)) and add a tile layer on top of it (Sections [6.5.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#what-are-tile-layers)–[6.5.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tile-layer-providers)). Tile layers are generic and thus typically serve as background, to help the user in positioning the foreground layers in geographical space. The foreground, on the other hand, is usually made of **vector layers**, such as points, lines, and polygons, though it is possible to add manually created [image layers](https://leafletjs.com/reference-1.7.1.html#imageoverlay) too. There are several methods for adding vector layers on a Leaflet map.

In this section, we are going to add vector layers using those methods where the layer coordinates (i.e., its geometry) are manually specified by passing numeric arrays to the respective function. That way, we are going to add markers (Section [6.6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-markers)), lines (Section [6.6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-lines)), and polygons (Section [6.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-polygons)) as foreground on top of the tile layer on our map. The latter coordinate-array methods are mostly useful for drawing simple, manually defined shapes. In Chapter [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1), we will learn how to add vector layers based on GeoJSON strings rather than coordinate arrays, which is more useful to add pre-existing, complex layers.

**6.6.2 Adding markers.**There are three ways of *marking* a specific point on our Leaflet map:

* **Marker**—A PNG image, such as Leaflet’s default blue marker (Figure [6.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-03))
* **Circle Marker**—A circle with a fixed radius in *pixels* (Figures [8.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#fig:solution-08), [12.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\client-side-geoprocessing.html#fig:example-12-05))
* **Circle**—A circle with a fixed radius in *meters*

To add a marker on our map, we use the L.marker function. The L.marker function creates a marker object, which we can add to our map with its .addTo method, just like we added the tile layer (Section [6.5.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-a-tile-layer)). When creating the marker object with L.marker we specify the longitude and latitude where it should be placed, using an array of length 2 in the [lat, lon] format[61](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn61) (Section [6.5.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#creating-map-object)). For example, the following expression—when appended at the end of the <script> in example-06-02.html—adds a marker in the yard near building #72 at Ben-Gurion University:

**let** pnt = L.marker([31.262218, 34.801472]).addTo(map);

You may wonder why we are assigning the marker layer to a variable, named pnt in this case, rather than just adding it to the map the way we did with the tile layer:

L.marker([31.262218, 34.801472]).addTo(map);

The answer is that the marker layer object contains useful methods, in addition to .addTo. We can apply those methods, later on, to make changes in the marker layer, such as adding popups on top of it (see Section [6.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-popups) below). The resulting map example-06-03.html—with the additional marker as a result of including the above expression—is shown in Figure [6.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-03).

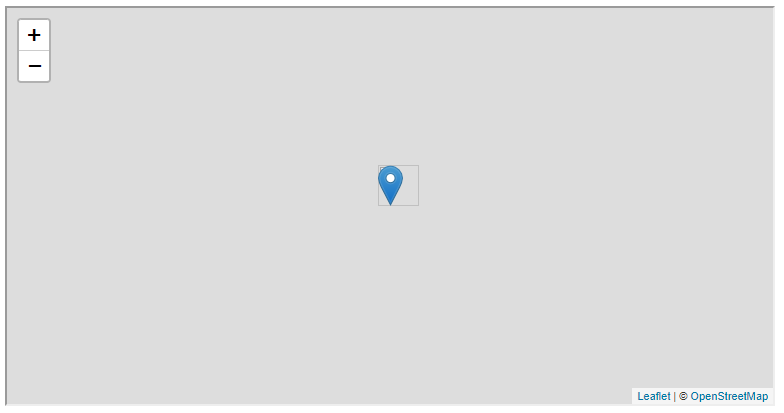


FIGURE 6.8: example-06-03.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-03.html)

The marker is actually a PNG image, and you can replace the default blue marker with any other image, which we learn how to do in Section [11.2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#adding-custom-marker). However, we cannot otherwise easily control the size and color of the marker, since this would require preparing a different PNG image each time. Therefore, it is sometimes more appropriate to use *circle markers* or *circles* instead of markers.

With both circle markers and circles, instead of adding a PNG image, vector circles are being drawn around the specified location. You can set the size (radius) and appearance of those circles, by passing an additional object with [options](https://leafletjs.com/reference-1.7.1.html#circlemarker), such as radius, color, and fillColor. The difference between circle markers and circles is in the way that the radius is set: in pixels (circle marker) or in meters (circle). For example, the following expression adds a circle marker 0.001 degrees to the west from where we placed the ordinary marker:

L.circleMarker(

[31.262218, 34.801472 - 0.001],

{radius: 50, color: "black", fillColor: "red"}

).addTo(map);

* Run the above expression in the console inside example-06-03.html. A circle marker with the specified color scheme should appear on screen.
* Zoom in and out; the circle size should remain constant on screen, at 50 pixels.

Remember that in circle markers, created with L.circleMarker, the radius property is given in pixels (e.g., 50), which means the circle marker maintains constant size on screen, irrespective of zoom level. In circles, created using L.circle, the radius is set in meters, which means the circle size maintains constant spatial extent, expanding or shrinking as we zoom in or out.

**6.6.3 Adding lines.**To add a *line* on our map, we use the L.polyline function. Since a line is composed of several points, we specify the series of point coordinates the line goes through as an array of coordinate arrays. The internal arrays, i.e., the point coordinates, are specified just like marker coordinates, in the [lat, lon] format. In the following example, we are constructing a line that has just two points, but there could be many more points when making a more complex line. The line is drawn in the order given by the array, from the first point to the last one.

We can specify the appearance of the line, much like with the circle marker (Section [6.6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-markers)), by setting various options such as line color and weight (i.e., width, in pixels). In case we are not passing any options, a default light blue 3px line will be drawn. In the following example, we override the default color and weight options, setting them to "red" and 10, respectively:

**let** line = L.polyline(

[[31.262705, 34.800514], [31.262053, 34.800782]],

{color: "red", weight: 10}

).addTo(map);

Note that the list of relevant options differs between different layer types, so once more you are referred to the [documentation](https://leafletjs.com/reference-1.7.1.html#polyline) to check which properties may be modified for each particular layer type. For example, the radius option which we used for a circle marker is irrelevant for lines.

The resulting map example-06-04.html, now with both a marker and a line, is shown in Figure [6.9](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-04).

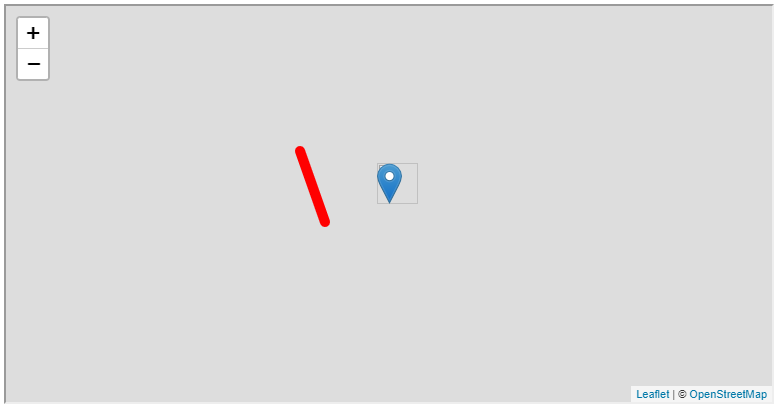


FIGURE 6.9: example-06-04.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-04.html)

It is also possible to draw a **multi-part** line (Table [7.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#tab:geojson-multi-part1)) with L.polyline, by passing an array of separate line segments: *an array* of arrays of arrays. We will not elaborate on this option here, because this type of more complex multi-part shapes are usually loaded from existing GeoJSON layers (Chapter [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1)).

**6.6.4 Adding polygons.**Adding a *polygon* is very similar to adding a line, only that we use the L.polygon function instead of L.polyline. Like L.polyline, the L.polygon function also accepts an array of point coordinates the polygon consists of. Again, the polygon is drawn in the given order, from first node to last. Note that the array is not expected to have the last point repeating the first one, to “close” the shape (unlike in the GeoJSON format; see Section [7.3.2.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#single-part-geometries)).

Like with circle markers, it is useful to define the border color and fill color of polygons, which can be done using the color and fillColor properties of the options object, respectively. For example, the following expression adds a polygon that has four nodes, with red border, yellow fill, and 4px border width:

**let** pol = L.polygon(

[

[31.263127, 34.803668],

[31.262503, 34.803089],

[31.261733, 34.803561],

[31.262448, 34.804752]

],

{color: "red", fillColor: "yellow", weight: 4}

).addTo(map);

The resulting map example-06-05.html, now with a marker, a line, and a polygon, is shown in Figure [6.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-05). Similarly to what we mentioned concerning lines (Section [6.6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-lines)), it is also possible to use L.polygon to add more complex polygons such as **multi-part** polygons or polygons with **holes** (Table [7.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#tab:geojson-multi-part1)). As mentioned previously, we will use the GeoJSON format for adding such complex shapes (Chapter [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1)).

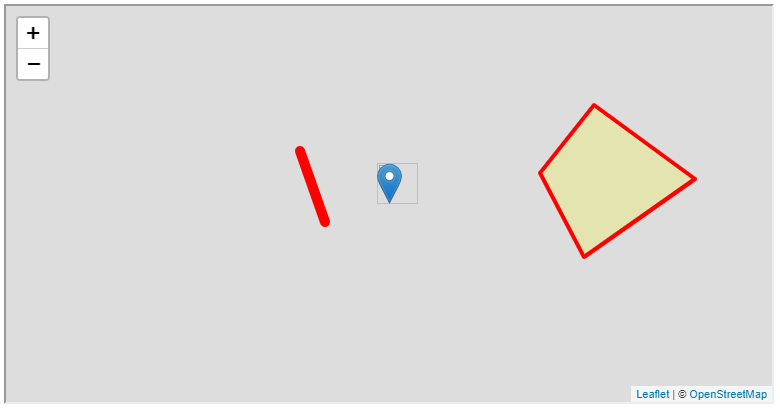


FIGURE 6.10: example-06-05.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-05.html)

**6.6.5 Other layer types.**So far we mentioned about half of the commonly used types of layers in Leaflet[62](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn62), as listed in the first six lines of Table [6.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#tab:leaflet-layer-types). We haven’t mentioned the rectangle layer, but this is practically a specific case of a polygon layer (Section [6.6.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-polygons)) so there is not much to say about it. You are invited to try the L.rectangle example from Leaflet documentation [on your own](https://leafletjs.com/reference-1.7.1.html#rectangle).

All of the latter layer types, except for tile layers, are vector layers drawn according to coordinate arrays. As mentioned above, the coordinate array method is useful for manually drawing simple shapes, but not very practical for complex vector layers, which are usually loaded from existing external sources. When working with complex predefined vector layers, the most useful type of Leaflet layer is L.geoJSON, specified using the GeoJSON format, which we learn about in Chapter [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1).

Leaflet also supports adding [**Web Map Service (WMS)**](https://en.wikipedia.org/wiki/Web_Map_Service) layers using the L.tileLayer.wms function, which are (usually) raster images, dynamically generated by a WMS server (unlike tile layers which are pre-built). Finally, Leaflet supports **grouping** layers using functions called L.layerGroup and L.featureGroup. We will use these later on, in Chapters [7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\geojson-1.html#geojson-1) and [10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\non-spatial-queries-1.html#non-spatial-queries-1)–[13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\collaborative-mapping.html#collaborative-mapping).

|  |  |
| --- | --- |
| TABLE 6.2: Commonly used Leaflet layer types | |
| **Layer** | **Function** |
| Tile layer | L.tileLayer |
| Marker | L.marker |
| CircleMarker | L.circleMarker |
| Circle | L.circle |
| Line | L.polyline |
| Polygon | L.polygon |
| Rectangle | L.rectangle |
| GeoJSON | L.geoJSON |
| Tile layer (WMS) | L.tileLayer.wms |
| Layer group | L.layerGroup |
| Feature group | L.featureGroup |

**6.7 Adding popups.**

**Popups** are informative messages which can be interactively opened and closed to get more information about the various features shown on a map. This is similar to the “identify” tool in GIS software. When the user clicks on a vector feature associated with a popup, an information box is displayed (Figure [6.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-06)). When the user clicks on the “X” (close) symbol on the top-right corner of the box, on any other element on the map, or on the **Esc** key, the information box disappears.

Popups are added on the map by *binding* them to a given layer. Binding can be done using the .bindPopup method that any Leaflet layer object has. The .bindPopup method accepts a text string with the popup content. For example, we can add a popup to the line layer named line we created in Section [6.6.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-lines), using the following expression:

line.bindPopup(

"This is the path from <b>our department</b> to the <b>library</b>."

);

Note that popup content can contain HTML tags. In this case, we just use the <b> tag to specify bold font (Section [1.6.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#font-formatting)), but you can use any HTML elements to add different type of content inside a popup, including links, lists, images, tables, videos, and so on. The resulting map with the popup is given in example-06-06.html[63](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn63). Clicking on the line opens a popup, as shown in Figure [6.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-06).

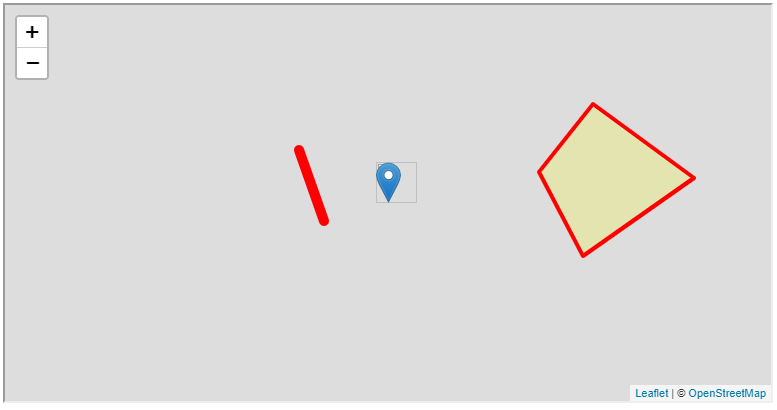


FIGURE 6.11: example-06-06.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-06.html)

* Try to bind a popup with the image of the Leaflet logo to the polygon shown in example-06-06.html (Figure [6.11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-06)), by running the appropriate expression in the console.
* Hint: use the code 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)) from example-01-04.html (Figure [1.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\html.html#fig:example-01-04)).

**6.8 Adding a description.**

In Section [2.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#example-map-description) we saw an example of adding map title and description using custom HTML and CSS code (Figure [2.13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\css.html#fig:example-02-11)). The Leaflet library has its own, simplified way to do the same task, using a function named L.control. With L.control, we can add custom map *controls* for different purposes, specified using custom HTML code. In the present example (example-06-07.html), we are going to use L.control to display a map description panel (Figure [6.13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-07)).

In example-06-07.html, we are using the L.control function to initialize a new map control named legend, and set its position to the bottom-left corner of the screen. The L.control function creates a new HTML element “above” the map, which can be filled with contents such as buttons or inputs (i.e., controls), map legends, or information boxes[64](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn64):

**let** legend = L.control({position: "bottomleft"});

In the present example, we want to fill the control with a map description, composed of few lines of text and an image (Figure [6.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:control-map-description)). We use the .onAdd method of the newly created control to define its contents and behavior. In this case, we are creating a <div> with HTML code which creates the map description:

legend.onAdd = **function**() {

**let** div = L.DomUtil.create("div", "legend");

div.innerHTML =

'<p><b>Simple shapes in Leaflet</b></p><hr>' +

'<p>This map shows an example of adding shapes ' +

'on a Leaflet map</p>' +

'The following shapes were added:<br>' +

'<p><ul>' +

'<li>A marker</li>' +

'<li>A line</li>' +

'<li>A polygon</li>' +

'</ul></p>' +

'The line layer has a <b>popup</b>. ' +

'Click on the line to see it!<hr>' +

'Created with the Leaflet library<br>' +

'<img src="images/leaflet.png">';

**return** div;

};

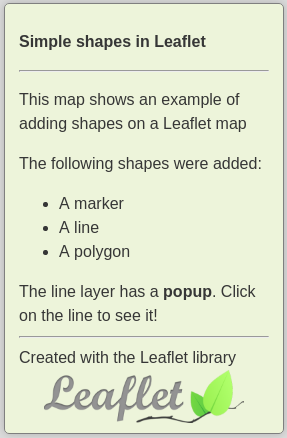


FIGURE 6.12: Map description

The above expression basically creates a <div> element with a class named "legend", then adds internal HTML content into that element. Adding the HTML code is done with the .innerHTML method, which we learned about in Section [4.3.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#modifying-elements). Note that we are using the + operators in order to split the assigned HTML code string into multiple lines and make the code more manageable and easier to read. Finally, the control is added to the map with .addTo method, just like we did for map layers in previous examples in this chapter:

legend.addTo(map);

The complete code for adding the map description is as follows:

**let** legend = L.control({position: "bottomleft"});

legend.onAdd = **function**() {

**let** div = L.DomUtil.create("div", "legend");

div.innerHTML =

'<p><b>Simple shapes in Leaflet</b></p><hr>' +

'<p>This map shows an example of adding shapes ' +

'on a Leaflet map</p>' +

'The following shapes were added:<br>' +

'<p><ul>' +

'<li>A marker</li>' +

'<li>A line</li>' +

'<li>A polygon</li>' +

'</ul></p>' +

'The line layer has a <b>popup</b>. ' +

'Click on the line to see it!<hr>' +

'Created with the Leaflet library<br>' +

'<img src="images/leaflet.png">';

**return** div;

};

legend.addTo(map);

We also need some CSS code to give our map description the final styling shown in Figure [6.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:control-map-description):

.legend {

**width**: 250px;

**font-size**: 16px;

**color**: #333333;

**font-family**: "Open Sans", Helvetica, sans-serif;

**padding**: 10px 14px;

**background-color**: rgba(245,245,220,0.8);

**box-shadow**: 0 0 15px rgba(0,0,0,0.2);

**border-radius**: 5px;

**border**: 1px solid grey;

}

.legend img {

**width**: 200px;

**margin**: auto;

**display**: block;

}

Finally, we need to have the leaflet.png image file in the images sub-directory inside the directory where the HTML document is, for the last expression in our HTML code to work. The expression adds the Leaflet library logo in the bottom of the map description (Figure [6.12](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:control-map-description)):

**<img** src="images/leaflet.png"**>**

The resulting map example-06-07.html, with the map description information box in the bottom-left corner, is shown in Figure [6.13](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-07).

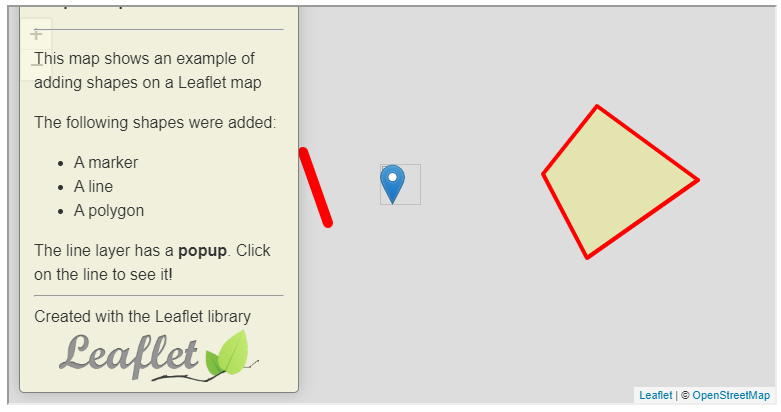


FIGURE 6.13: example-06-07.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-07.html)

**6.9 Introducing map events.**

In Section [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners), we learned how the browser fires events in response to user interactions, and that these events can be handled using JavaScript to make our web page interactive. We also mentioned the event object (Section [4.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#the-event-object)), which can be used to trigger specific responses according to event properties. Recall the mouse position in example-04-07.html (Figure [4.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#fig:example-04-07)), where mouse position was printed on screen thanks to the pageX and pageY properties of the event object. In Section [4.10](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#the-event-object), we also mentioned that *custom* event types and *custom* event object properties are often defined in JavaScript libraries for specific needs.

The Leaflet library defines specific event types and event properties appropriate for web mapping. These specific events can be captured for the entire map object, or for specific features the map contains. In this context, a popup is an example of a built-in event listener, with a pre-defined behavior: opening on click, closing when the “X” button or any other layer is clicked. Similarly, we can define and customize other Leaflet map interactive behaviors.

One of the most useful types of **map events** are "click" events, and the most important property of a map click event is the spatial location where the user clicked. When the user clicks on the map—in addition to the usual "click" event properties—the event object contains the latitude and longitude of where the click was made. The latitude and longitude are given in the special .latlng property of the [map event](https://leafletjs.com/reference-1.7.1.html#map-event) object. The .latlng event object property opens up the possibility of “capturing” the clicked coordinates and doing something useful with them.

For example, in Chapter [11](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#spatial-queries-1) we will use the clicked location to query specific features—that are *near* the clicked location—from a database (Figure [11.8](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\spatial-queries-1.html#fig:example-11-04)). In this section, we will see a simpler example, displaying the clicked coordinates inside a popup, which is opened on the clicked location itself (Figure [6.15](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-08)). To display the currently clicked location using a popup, we need to do several things:

* Add an empty popup object to our map
* Write a function that, once called, sets popup location, fills the popup with content, and opens it on the map
* Add an event listener stating that each time the map is clicked, the latter function is executed

Starting with the first item, an empty popup object can be created with the L.popup function, as follows. This is in contrast to the .bindPopup method we used earlier to bind a popup to an *existing* layer (Section [6.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#adding-popups)):

**let** popup = L.popup();

In the function that the event listener will refer to, hereby named onMapClick (see below), we will be using the event object parameter e. As mentioned above, when referring to map "click" events, the event object includes the .latlng property. The .latlng property is itself an object of type LatLng, which holds the longitude and latitude of the clicked location. This is basically a JavaScript object with two properties and two numeric values, such as the following one (plus some additional methods, which we don’t need to worry about at the moment):

{lat: 31.2596049045682, lng: 34.80215549468995}

A LatLng object can be created with L.latLng, and it can be used for specifying coordinates instead of a simple [lat, lon] array. For example, try running the following expression in the console in any of the above examples. This will add a marker at a location defined using a LatLng object:

L.marker(L.latLng(31.264, 34.802)).addTo(map);

Back to the event listener example. The LatLng object, representing the clicked location, will be used for two purposes:

* Determining **where** the popup should be opened
* Setting the popup **contents**

These are accomplished with two methods of the popup object, .setLatLng and .setContent. Finally, once popup placement and content are set, the popup will be opened using a third method named .openOn. The event listener function, hereby named onMapClick, thus takes the following form:

**function** onMapClick(e) {

popup

.setLatLng(e.latlng)

.setContent(

"You clicked the map at -<br>" +

"<b>lon:</b> " + e.latlng.lng + "<br>" +

"<b>lat:</b> " + e.latlng.lat

)

.openOn(map);

}

To better understand what this function does, try running the following code in the console in one of the Leaflet examples from this chapter:

**let** myLocation = L.latLng(31.264, 34.802);

L.popup()

.setLatLng(myLocation)

.setContent(

"You clicked the map at -<br>" +

"<b>lon:</b> " + myLocation.lng + "<br>" +

"<b>lat:</b> " + myLocation.lat

)

.openOn(map);

This is basically the same code as the body of the onMapClick function, only that instead of e.latlng we used a specific LatLng instance, named myLocation. As a result of running this code section, you should see a popup with the content shown in Figure [6.14](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:popup-clicked-location), opened at the location specified by L.latLng(31.264, 34.802).

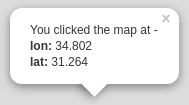


FIGURE 6.14: A popup displaying clicked location coordinates

Finally, we add an event listener (Section [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners)) specifying that the onMapClick function should be executed on map click. Note that in this case the element responding to the event is the entire map, so that any click inside the map triggers the onMapClick function[65](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fn65):

map.addEventListener("click", onMapClick);

The complete code we need to add to a basic Leaflet map, so that clicked coordinates are displayed in a popup, is shown on the following page:

**let** popup = L.popup();

**function** onMapClick(e) {

popup

.setLatLng(e.latlng)

.setContent(

"You clicked the map at -<br>" +

"<b>lon:</b> " + e.latlng.lng + "<br>" +

"<b>lat:</b> " + e.latlng.lat

)

.openOn(map);

}

map.addEventListener("click", onMapClick);

The resulting map example-06-08.html is shown in Figure [6.15](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-08).

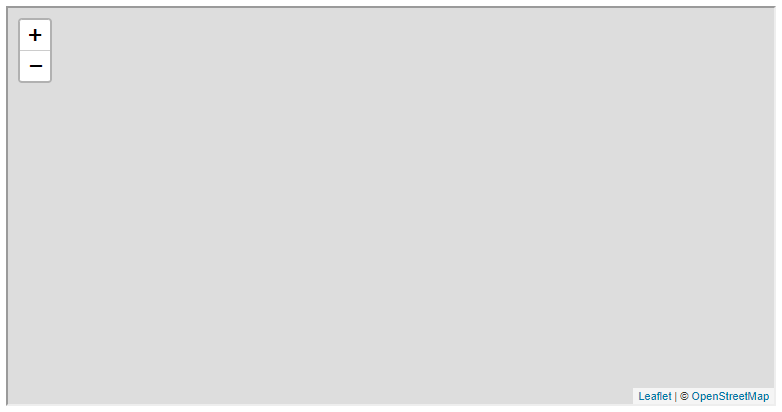


FIGURE 6.15: example-06-08.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\example-06-08.html)

* Open example-06-08.html (Figure [6.15](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-08)) and click anywhere on the map.
* As shown in Figure [6.15](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:example-06-08), the popup displays the e.latlng coordinates “as is,” with many irrelevant digits.
* The first six digits refer to sub-meter (~0.1 m) precision, which is more than enough for most types of applications.
* Modify the onMapClick function so that the coordinates are rounded and only the first *six* digits of longitude and latitude are displayed inside the popup. (Hint: you can search on Google for a JavaScript function to round numbers according to specified number of digits.)

**6.10 Exercise**

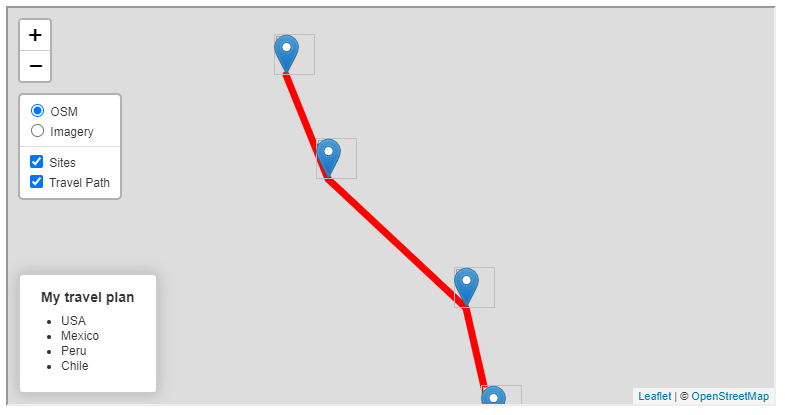
* Create a Leaflet map with the location of places you want to visit, and the path you want to travel along between those places.
* The map should have the following components:
  + A tile layer
  + Markers of the visited locations
  + A line representing the travel path
  + Popups for each marker, containing location names
  + A description box, with your name and the list of locations in the right order
* Optional: Check out the [*Layer Groups and Layers Control*](https://leafletjs.com/examples/layers-control/) tutorial, and try to add a layer control for showing or hiding the markers and the line layers, and for switching between two or more types of tile layers (Figure [6.16](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fig:solution-06)).
* 

FIGURE 6.16: solution-06.html [(Click to view this example on its own)](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\examples\solution-06.html)

**References**

Crickard III, Paul. 2014. *Leaflet.js Essentials*. Birmingham, UK: Packt Publishing Ltd.

Dincer, Alper, and Balkan Uraz. 2013. *Google Maps JavaScript API Cookbook*. Birmingham, UK: Packt Publishing Ltd.

Farkas, Gábor. 2016. *Mastering OpenLayers 3*. Birmingham, UK: Packt Publishing Ltd.

Gratier, Thomas, Paul Spencer, and Erik Hazzard. 2015. *OpenLayers 3: Beginner’s Guide*. Packt Publishing Ltd.

Janert, Philipp K. 2019. *D3 for the Impatient: Interactive Graphics for Programmers and Scientists*. Sebastopol, CA, USA: O’Reilly Media, Inc.

Langley, Peter J, and Antonio Santiago Perez. 2016. *OpenLayers 3.x Cookbook*. 2nd ed. Birmingham, UK: Packt Publishing Ltd.

Murray, Scott. 2017. *Interactive Data Visualization for the Web: An Introduction to Designing with D3*. Sebastopol, CA, USA: O’Reilly Media, Inc.

Newton, Thomas, and Oscar Villarreal. 2014. *Learning D3.js Mapping*. Birmingham, UK: Packt Publishing Ltd.

Rubalcava, Rene. 2015. *ArcGIS Web Development*. Shelter Island, NY, USA.: Manning Publications Co.

1. A good introduction to web mapping can be found in the *Web Maps 101* (<http://maptimeboston.github.io/web-maps-101/#0>) presentation by *Maptime Boston* (<https://maptimeboston.github.io/resources>).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref52)
2. Check out the *What is a web mapping API?* (<https://www.e-education.psu.edu/geog585/node/763>) article for additional information on pros and cons of different web-mapping libraries.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref53)
3. Not before![↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref54)
4. See Appendix [A](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\associated-files.html#associated-files) for complete list of files and directory structure; see online version (Section [0.7](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\index.html#online-version)) for downloading the complete set of files for the Leaflet library.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref55)
5. You can always to refer to the documentation (<https://leafletjs.com/reference-1.7.1.html>) for the complete list of options for L.map, or any other Leaflet function.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref56)
6. See the *Zoom levels* tutorial (<https://leafletjs.com/examples/zoom-levels/>) for more information on tile layer zoom levels and their implementation in Leaflet.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref57)
7. In fact, the default OpenStreetMap tile server is not a simple static server. It has a more efficient (and complicated) design, to conserve storage space and computation power. Tiles that are rarely viewed (in the ocean, etc.) are rendered *upon request*, instead of being stored on the server. See <https://wiki.openstreetmap.org/wiki/Tile_disk_usage> for more details.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref58)
8. Again, for the complete list of options see the Leaflet documentation (<https://leafletjs.com/reference-1.7.1.html#tilelayer>).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref59)
9. For more information, see the *Leaflet on Mobile* tutorial (<https://leafletjs.com/examples/mobile/>) and the *Using the viewport meta tag* article (<https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag>) by Mozilla.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref60)
10. Remember that the coordinates arrays that Leaflet accepts follow the [lat, lon] format, *not* the [lon, lat] format![↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref61)
11. The complete list of Leaflet layer types is given in the Leaflet documentation (<https://leafletjs.com/reference-1.7.1.html>).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref62)
12. See the Leaflet *Quick Start Tutorial* (<https://leafletjs.com/examples/quick-start/>) for another overview on adding simple layers with popups in Leaflet.[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref63)
13. In subsequent chapters, we are going to use L.control to display other types of controls, such as a map legend (Section [8.6](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#adding-a-legend)), a dynamically updated information box (Section [8.7.2](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\symbology-and-interactivity.html#dynamic-control-contents)), and a dropdown menu (Section [10.3](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\non-spatial-queries-1.html#creating-a-dropdown-menu)).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref64)
14. The .on method—which you may encounter in the Leaflet tutorials and other code examples—is identical to the .addEventListener method. Either one can be used to bind event listeners in Leaflet. In this book, we will be using .addEventListener for uniformity with the plain JavaScript syntax (Section [4.4](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\javascript-interactivity.html#event-listeners)).[↩︎](file:///C:\Users\User\Downloads\Telegram%20Desktop\Web_Gis\Web_Gis\js\leaflet.html#fnref65)