**Leaflet JS library. Create interactive maps.**



[**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)).

**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 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});

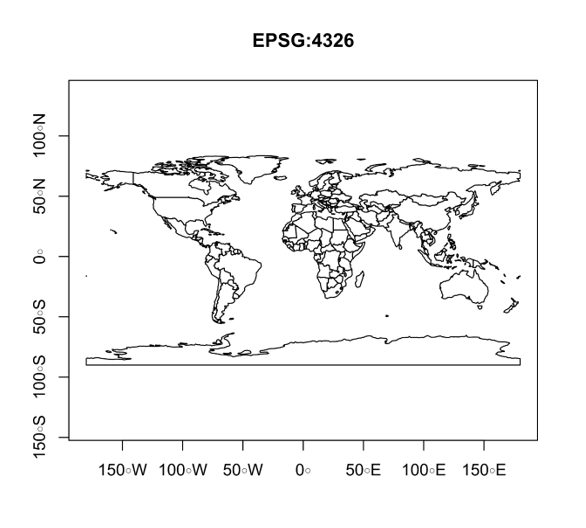
Note that the center option is specified in **geographic coordinates** (longitude and latitude) using an array of the form [lat, lon] rather than [lon, lat], that is, in the [Y, X] rather than the [X, Y] order. This may seem unintuitive to GIS users, but the [lat, lon] ordering is actually very common 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]



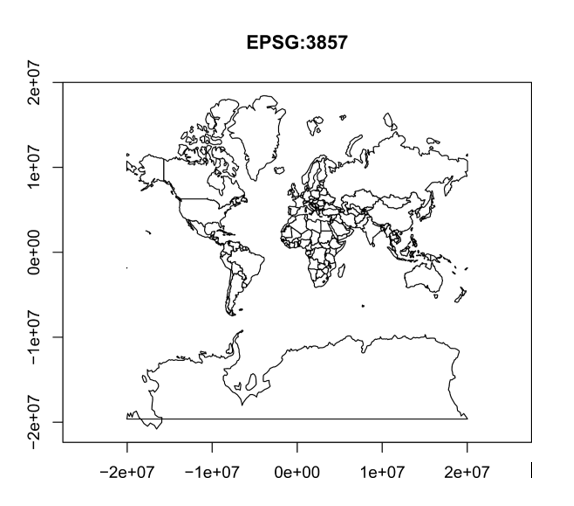


FIGURE 6.1: World map in the WGS84 (EPSG:4326) and Web mercator (EPSG:3857) projections

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.