

# Astrid Günther

# Leaflet

Eine  
Einig  
Open-Source-JavaScript-Bibliothek  
für mobil-freundliche digitale Karten

# Inhaltsverzeichnis

[Willkommen](#)

[Wichtiges zum Buch](#)

[Was ist Leaflet?](#)

[Was sollten Sie mitbringen?](#)

[Die Autorin](#)

[Eine Karte mit Leaflet erstellen](#)

[In diesem Kapitel werden wir ...](#)

[Wir beginnen mit einer einfachen Karte](#)

[Integrieren Sie die notwendigen JavaScript und Cascading Style Sheet \(CSS\) Dateien](#)

[Leaflet über ein CDN einbinden](#)

[Eine lokale Leaflet-Kopie einbinden](#)

[Leaflet performant einbinden – defer oder async](#)

[Was passiert genau, wenn eine Website mit einem <script>-Tag geladen wird?](#)

[Wie können Sie die Ladezeit positiv beeinflussen?](#)

[Was sollten Sie beim Einsatz von defer oder async mit Leaflet beachten?](#)

[Erstellen Sie ein HTML-Element in dem Ihre Karte angezeigt werden soll](#)

[Erstellen Sie das Karten-Objekt](#)

[Fügen Sie eine Schicht mit Kacheln – einen Tile-Layer – zum Karten-Objekt hinzu](#)

[Exkurs: Geographische Koordinaten](#)

[Das Koordinatensystem der Erde](#)

[Breitengrade](#)

[Längengrade](#)

[Schreibweisen von geografischen Koordinaten](#)

[Das Sexagesimalsystem](#)

[Die Dezimalschreibweise](#)

[Exkurs: Wie werden Landkarten auf einer Webseite angezeigt?](#)

[Grafikformate: Vektoren und Rastergrafiken](#)

[Vektoren](#)

[Rastergrafiken](#)

[Vektoren und Rastergrafiken für digitale Karten](#)

[Wir unterteilen die Welt in Kacheln](#)

[Wie weiß Leaflet welche der vielen Kacheln angezeigt werden sollen?](#)

[Schöne Kartenlayer](#)

[Thunderforest](#)

[Stamen](#)

[Images als Layer – Web-Map-Service](#)

[Eine einfache Leaflet-Karte mithilfe des Web-Map-Services erstellen](#)

[L.tileLayer.wms über L.tileLayer.wms](#)

[L.tileLayer.wms und L.tileLayer zusammen auf einer Karte](#)

[In diesem Kapitel haben wir ...](#)

[Die Karte mit Daten bestücken](#)

[In diesem Kapitel werden wir ...](#)  
[Punkte, Marker, Linien, Polygone, Rechtecke und Kreise als Leaflet Objekte](#)  
[Punkte und Marker](#)  
[Punkte](#)  
[Marker](#)  
[Objekte, die aus mehr als einem Punkte bestehen](#)  
[Linien](#)  
[Polygone](#)  
[Rechtecke und Kreise](#)  
[Rechtecke](#)  
[Kreise](#)  
[Mehrere Poly- Objekte auf einer Ebene](#)  
[Mehrere Polyline Objekte auf einer Ebene](#)  
[Mehrere Polygon Objekte auf einer Ebene](#)  
[Mehrere sich überschneidende Polygon Objekte auf einer Ebene](#)  
[Daten mit Layern gruppieren](#)  
[Layergruppen](#)  
[Featuregruppen](#)  
[Popups](#)  
[Mobil](#)  
[HTML und CSS](#)  
[JavaScript](#)  
[Ereignisse in Leaflet](#)  
[Eine benutzerdefinierte Funktionen](#)  
[Ein interaktives Beispiel](#)  
[In diesem Kapitel haben wir ...](#)  
[GeoJSON](#)  
[In diesem Kapitel werden wir ...](#)  
[Die Entwicklungsgeschichte von GeoJSON](#)  
[Warum ist das Format GeoJSON entstanden?](#)  
[XML](#)  
[JSON](#)  
[Und warum nun auch noch GeoJSON?](#)  
[GeoJSON erkunden](#)  
[Eine Geometrie](#)  
[Position](#)  
[Point](#)  
[Multipoint](#)  
[LineString](#)  
[MultiLineString](#)  
[Polygone](#)  
[MultiPolygon](#)  
[Mehrere Geometrien zusammenfassen - GeometryCollection](#)  
[Einer GeoJSON Geometrien Eigenschaften zuordnen](#)

[FeatureCollection](#)

[Die Grenzen von GeoJSON](#)

[GeoJson in Leaflet](#)

[Ein GeoJSON Feature in Leaflet einbinden](#)

[Eine GeoJSON FeatureCollection in Leaflet einbinden](#)

[GeoJSON aus Leaflet exportieren](#)

[Stilen](#)

[Beim Erstellen eines GeoJSON Layer einen Stil mitgeben](#)

[Ein komplexeres Beispiel: Eine andere Farbe beim Überrollen mit der Maus](#)

[Iterieren](#)

[OnEachFeature\(\) – Bearbeite jedes Feature](#)

[PointToLayer – Punkt zu Ebene](#)

[Filtern mit der Option filter](#)

[In diesem Kapitel haben wir ...](#)

[Heatmaps und Choroplethenkarten](#)

[In diesem Kapitel werden wir ...](#)

[Heatmaps](#)

[Was ist eine Heatmap?](#)

[Heatmaps in Leaflet – Dichte](#)

[Stile-Optionen](#)

[Nachfolgend sehen Sie einen Vergleich einer Ansichten mit unterschiedlichen Werten für die Optionen blur, gradient und maxZoom](#)

[Methoden](#)

[Die Methode addLatLng\(\)](#)

[Die Methoden addLatLng\(\), addLatLngs\(\) und setOptions\(\) in einem Beispiel](#)

[Marker](#)

[Heatmaps mit Leaflet – heatmap.js – Intensität](#)

[Dokumentation und Methoden](#)

[Interaktive Heatmaps](#)

[Animierte Heatmaps](#)

[Choroplethenkarte](#)

[Was genau ist eine Choroplethenkarte?](#)

[Choroplethenkarten in Leaflet](#)

[Open Data](#)

[Farben](#)

[Stile](#)

[Das vollständige Beispiel](#)

[Normalisierte Choroplethenkarten](#)

[In diesem Kapitel haben wir ...](#)

[Custom Markers](#)

[In diesem Kapitel werden wir ...](#)

[Ein individueller Marker auf Ihrer Karte](#)

[Eigenschaften eines individuellen Marker](#)

[Die Klasse L.Icon erweitern](#)

[Ein Marker Plugin](#)

[BeautifyIcon](#)

[Cluster](#)

[Optionen, Methoden und Ereignisse](#)

[Marker animieren](#)

[Hüpftende Marker](#)

[Animierte Marker](#)

[Ein Marker bewegt sich](#)

[Einen Marker in Bewegung setzen und wieder stoppen](#)

[Plugins kombinieren](#)

[Leaflet Data Visualization Framework \(DVF\)](#)

[Das Plugin Data Visualization Framework](#)

[Diagrammen als Marker](#)

[In diesem Kapitel haben wir ...](#)

[ESRI](#)

[In diesem Kapitel werden wir ...](#)

[L.esri.BasemapLayer erweitert L.TileLayer](#)

[Basemaps und optionale Layer von ESRI](#)

[Basemaps](#)

[Optionale Label Layer](#)

[Ein erstes Beispiel](#)

[Switch Basemaps](#)

[Shapefiles](#)

[Was genau verbirgt sich hinter dem Begriff Shapefile](#)

[Wie kommen Sie an ein Shapefiles](#)

[Wie binden Sie Shapefiles in Ihre Leaflet Karte ein?](#)

[Deutsche Verwaltungsgrenzen als Shapefile in der Leaflet Karte](#)

[Deutsche Verwaltungsgrenzen mit Optionen](#)

[Ein Pop-up-Fenster](#)

[Ein Pop-up-Fenster mit allen Informationen des Features](#)

[ESRI Services](#)

[L.esri.DynamicMapLayer](#)

[Geocoding](#)

[Nach Eingabe einer Adresse den passenden Ort auf der Karte finden](#)

[Mithilfe eines Parameters in der URL den passenden Ort finden](#)

[Nach Klick auf die Karte die passende Adresse ausgeben](#)

[Feature Services](#)

[Attribute](#)

[Abstand visualisieren](#)

[In diesem Kapitel haben wir ...](#)

[Routing](#)

[In diesem Kapitel werden wir ...](#)

[Allgemeines zum Thema Routing...](#)

[Leaflet Routing Maching](#)

[Options](#)

[Geocoding und Routing](#)

[In diesem Kapitel haben wir ...](#)

[Auf Wiedersehen](#)

[Impressum](#)

# Willkommen

Das Arbeiten mit Geodaten und Landkarten hat durch das globale Positionsbestimmungssystem GPS immer mehr an Relevanz gewonnen. Viele Anwendungen bieten heute sogar online geografische Informationen in Echtzeit an. So finden Sie auch im Internet immer mehr digitale Karten und Anwendungen die mit Geodaten arbeiten.

Geodaten sind Informationen, die eine Zuordnung zu einer räumlichen Lage besitzen. 90 % aller Daten können einem Ort zugeordnet werden und sind somit Geodaten.

Sie haben dieses Buch sicherlich gekauft, weil auch Sie eine digitale Karte anzeigen möchten - höchstwahrscheinlich auf einer Website. Die Grundlagen zu dem, was ich in diesem Buch geschrieben habe, finden Sie alle in den Dokumentationen zu Leaflet oder den jeweiligen Plugins öffentlich im Internet. Diese Dokumentationen habe ich im Buch verlinkt. Warum habe ich trotz der guten Dokumentation dieses Buch geschrieben? Ich habe das Buch geschrieben, weil ich manchmal gerne mit Beispielen lerne. Oft komme ich hierbei auf Ideen, die trockene Dokumentationen nicht hergeben. Vielleicht geht es Ihnen ja auch so und dieses Buch bringt Ihnen einen Mehrwert zu den vorhandenen Dokumentationen. Die Quellcode Dateien zu den im Buch verwendeten Beispielen finden Sie auf der Website <https://astrid-guenther.de>.

Außerdem finde ich es immer sehr hilfreich, wenn ich ein bisschen über den Tellerrand hinaus blicke. Hier im Buch finden Sie ein paar solcher Blicke. Wenn es Sie interessiert, können Sie mit mir etwas tiefer in die Welt der geografische Koordinaten blicken. Ich erkläre Ihnen auch die unterschiedlichen Techniken beim Erstellen der Grafiken für die digitalen Karten. Außerdem sehe ich mir mit Ihnen GeoJSON, über die reine Anwendung hinaus, etwas genauer an. Ich zeigen Ihnen wie Sie der Karte mit individuellen Markern eine persönliche Note geben können. Neben diesen grundlegenden Elementen kommen die Visualisierung der Daten mit Heatmaps, Geocoding und Routing nicht zu kurz.

## Wichtiges zum Buch

Ich erkläre Ihnen hier Schritt für Schritt alles das, was ich rund um Leaflet als wichtig ansehe – also alles das, was Sie als Entwickler brauchen, um kreativ arbeiten zu können.

Ich hoffe, dass Ihnen meine Art zu schreiben gefällt. Ich persönlich hätte mir genau dieses Buch zum Start mit Leaflet gewünscht.

# Was ist Leaflet?

Leaflet.js ist eine Open Source JavaScript-Bibliothek, die Ihnen das Bereitstellen von Karten auf Ihrer Webseite einfach macht. Open Source bedeutet, dass der Programmcode einsehbar ist. Jeder mit den notwendigen Kenntnissen kann prüfen, wie die Anwendung genau funktioniert. Und was noch wichtiger ist: Jeder kann Leaflet verwenden, an seine Bedürfnisse anpassen und verbessern.

Was müssen Sie tun, wenn Sie Leaflet auf Ihrer Website einsetzen möchten? Im Grunde genommen müssen Sie nur zwei Dateien – eine JavaScript Datei und die dazugehörige CSS-Datei einbinden. Wie Sie das genau machen erkläre ich Ihnen im Kapitel *Integrieren Sie die notwendigen JavaScript und Cascading Style Sheet (CSS) Dateien*. ganz genau.

Das Einbinden der Dateien und bietet Ihnen den Zugriff auf eine Reihe von Funktionen Mithilfe dieser Funktionen können sie unkompliziert eine digitale Karte auf Ihrer Website integrieren. Leaflet unterstützt alle modernen Browser – auch die mobilen Versionen. Das bedeute, dass Sie Ihre Karte so ziemlich überall einsetzen können.

Leaflet selbst legt den Schwerpunkt auf die einfache Bedienbarkeit und die Performance. Außerdem ist es den Entwicklern von Leaflet wichtig, dass die JavaScript Bibliothek unkompliziert von anderen Entwicklern mit einem Plugin erweitert werden kann. Die Programmierschnittstelle ist sehr gut dokumentiert. Als Erweiterungsprogrammierer muss man keine Geheimnisse lüften, um Leaflet erfolgreich mit einem Plugin zu erweitern. Dies ist meiner Meinung nach eine der bedeutendsten Stärken von Leaflet.

Viele namhafte Unternehmen setzten Leaflet ein. Darunter sind Namen wie Flickr, Github, Pinterest, Wikimedia und Spiegel.

Out-of-the-box können Sie mit Leaflet Marker, Popups, Linien und Formen auf unterschiedlichen Ebenen auf Ihrer Karte anzuzeigen. Sie können Zoomen, Entfernung berechnen und das Zentrum der Karte zu bestimmten Koordinaten schieben.

# Was sollten Sie mitbringen?

Ich gehe davon aus, dass Sie über grundlegende HTML und CSS Kenntnisse verfügen. Sie sollten auf alle Fälle wissen, wie Sie Cascading Style Sheets (CSS) und ein JavaScript Skript in ein HTML-Dokument einbinden und wie Sie mit einfachen HTML-Elementen arbeiten. Für das Verständnis der Beispiele sind darüber hinaus grundlegende Javascript Kenntnisse hilfreich.

# **Die Autorin**

Ich habe über 30 Jahre im öffentlichen Dienst gearbeitet. In dieser Zeit habe ich nebenberuflich das Abitur nachgeholt und im Anschluss ein Studium mit dem Abschluss Master of Computer Science bei der Fernuni Hagen abgeschlossen.

Heute arbeite ich freiberufllich unter anderem mit Leaflet, lebe mit meinem Mann, meiner Tochter, meinem Hund und meiner Katze in einem kleinen Dorf in der Eifel und bin passionierte Joggerin.

Fragen zum Buch, konstruktive Kritik, Anregungen und Hinweise auf Tippfehler können gerne per E-Mail an die E-Mail-Adresse [info@astrid-guenther.de](mailto:info@astrid-guenther.de) gesandt werden.

# Eine Karte mit Leaflet erstellen

## In diesem Kapitel werden wir ...

Zunächst zeige ich Ihnen, wie Sie in vier einfachen Schritten eine *digitale Karte* mit Leaflet in ein HTML Dokument einbinden. Mit Leaflet müssen Sie dafür nicht wissen, was geografische Koordinaten sind und wie die Bilddateien für die Karten erstellt werden. Da dies aber als Hintergrundwissen bei einer eventuellen Fehlersuche hilfreich sein kann, habe ich einen Theorie-Teil in dieses Kapitel eingefügt. Hier erkläre ich Ihnen Wichtiges zum Thema geografische Koordinaten und die übliche Vorgehensweise beim Erstellen und Zuordnen der Imagedateien für diese Karten – die *Kachel-Technik*. Zum Abschluss lernen Sie dann noch eine Alternative zur Kachel-Technik kennen, den *Web-Map-Service*.

## Wir beginnen mit einer einfachen Karte

Um eine Karte mit Leaflet auf einer Internetseite anzuzeigen, reichen wenige Schritte aus:

1. Integrieren Sie die notwendigen JavaScript und Cascading Style Sheet (CSS) Dateien.
2. Erstellen Sie ein HTML-Element – üblicherweise ein `<div>`-Element – in dem Ihre Karte angezeigt werden soll.
3. Erstellen Sie das JavaScript Karten-Objekt.
4. Fügen Sie eine Schicht mit Kacheln – einen Tile-Layer – zum Karten-Objekt hinzu.

CSS ist eine Stylesheet-Sprache für digitale Dokumente. Stylesheet-Sprachen werden verwendet, um Dokumente und Benutzeroberflächen zu gestalten. Dabei ist ein Stylesheet mit einer Formatvorlage zu vergleichen. Grundidee hierbei ist die Trennung von Dateninhalten und Design. Zusammen mit HTML und dem Document Object ModelOM (DOM) ist CSS eines der wichtigsten Elemente im Internet. CSS ist ein so genannter lebendiger Standard und wird vom World Wide Web Consortium (W3C) permanent weiterentwickelt.

Noch vor Schritt 1 erstellen wir eine einfache HTML-Datei. Diese Datei ist die Grundlage für die Beispiele hier im Buch. Den Programmcode für die einfache HTML-Datei sehen Sie im nachfolgenden Programmcodebeispiel. Fügen Sie diesen Programmcode in eine Datei ein und speichern diese Datei ab.

```
<!DOCTYPE HTML>
<html>
<head>
```

```
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
</head>
<body>
</body>
</html>
```

Ein Aufruf dieser Datei in Ihrem Browser öffnet zunächst nur ein leeres Browser-Fenster. Nur der Titel der Seite ist in der Titelleiste des Browsers abzulesen. In den nächsten vier Kapiteln erfahren Sie, wie Sie die digitale Landkarte in das Fenster bekommen. Vier Kapitel hört sich aufwendiger an. Das ist es aber nicht. Fangen Sie an und Sie werden sehen, dass Sie schon in wenigen Minuten die Karte präsentieren können.

## Integrieren Sie die notwendigen JavaScript und Cascading Style Sheet (CSS) Dateien

Sie haben zwei Möglichkeiten die notwendigen Dateien in Ihr HTML-Dokument zu integrieren.

1. Binden Sie die Dateien über ein Content Delivery Network (CDN) in Ihr HTML-Dokument ein.
2. Kopieren Sie die Dateien und binden Sie die lokale Kopie in Ihr HTML-Dokument ein.

■ Ein Content Delivery Network oder Content Distribution Network (CDN) ist ein Netz von Servern, die über das Internet verbundenen sind. Diese Server bieten Inhalte zum Download an.

### Leaflet über ein CDN einbinden

Sie können Leaflet mithilfe eines CDN nutzen. So müssen Sie die Dateien nicht selbst herunterladen. Es ist lediglich eine Verlinkung nötig. Mit der richtigen Verlinkung wird Leaflet automatisch über das CDN heruntergeladen, wenn ein Website Besucher Ihre Website aufruft. Im nachfolgenden Programmcodebeispiel sehen Sie den Text für die Verlinkung zum Aufruf der Leaflet-Version 1.2.0 in fett formatiert.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.2.0/dist/leaflet.css"/>
<script src="https://unpkg.com/leaflet@1.2.0/dist/leaflet.js"></script>
</head>
<body>
</body>
</html>
```

Welche Leaflet-Versionen Ihnen über das CDN zur Verfügung stehen, können Sie jederzeit unter der Adresse <http://leafletjs.com/download.html> nachlesen.

Auch wenn Leaflet nun automatisch über das CDN heruntergeladen wird: Ein Aufruf dieser Datei in Ihrem Browser öffnet immer noch ein leeres Browser-Fenster. Erst im letzten Schritt, im Kapitel *Fügen Sie eine Schicht mit Kacheln – einen Tile-Layer – zum Karten-Objekt hinzu*, wird die Karte sichtbar.

Fast alle modernen Browser unterstützen Subresource Integrity (SRI) und auch Leaflet verwendet diese Sicherheitsfunktion. SRI steht für die Überprüfung der Integrität von Dateien die im Internet verwendet werden. Mithilfe dieser Funktion können Sie sich davor schützen, dass nachträglich veränderte Dateien auf Ihrem Server ausgeführt werden. Die Dateien könnten beispielsweise bei der Übertragung im Internet verändert werden. Wenn Sie SRI nutzen, erhalten Sie immer exakt die Version, die Ihnen auch ausgeliefert werden sollte – oder eine Fehlermeldung. Dabei wird auf eine Prüfsumme – also einen Hash-Wert – zurückgegriffen. Diese Prüfsumme macht die Datei eindeutig erkennbar. Um potenzielle Sicherheitsprobleme zu vermeiden, empfehle ich Ihnen SRI zu nutzen, wenn Sie Leaflet über ein CDN einbinden. Wie Sie dies genau handhaben, finden Sie unter der Adresse <http://leafletjs.com/download.html>.

## Eine lokale Leaflet-Kopie einbinden

Eine zweite – vielleicht etwas kompliziertere – Möglichkeit ist, die Leaflet-Dateien selber herunterzuladen. Die aktuellen Versionen finden Sie unter der Adresse <http://leafletjs.com/download.html>. Diese müssen Sie anschließend auf dem eigenen Server verfügbar machen und in Ihre Website einbinden. Diese Variante hat den Vorteil, dass Sie nicht von einem CDN Server abhängig sind. Somit haben Sie mehr Sicherheit. Sie wissen genau, welche Datei Sie auf Ihrem Server abgelegt haben und haben Einfluss auf die Konfiguration Ihres Servers. Nachteilig ist dabei allerdings, dass Sie sich selbst um alles kümmern müssen. Sie müssen selbst die Dateien kopieren und sicherstellen, dass Sie die passende Version von Leaflet nutzen.

Im nachfolgenden Codebeispiel sehen Sie – fett hervorgehoben – das Einbinden von Leaflet unter der Annahme, dass Sie die heruntergeladenen Dateien - relativ zu Ihrem HTML-Dokument ein Verzeichnis höher - im Unterverzeichnis `/leaflet` auf Ihrem Webserver abgelegt haben.

Wenn Sie mit relativen Pfadangaben arbeiten, setzen Sie die Links innerhalb eines Projektes mit Hilfe von Punkten. Der große Vorteil von relativen Pfadangaben ist, dass Sie ein Projekt jederzeit in ein anderes Verzeichnis kopieren können ohne Pfadangaben korrigieren zu müssen. Bei einer relativen Pfadangabe bedeuten die Punkte folgendes:

Ein Punkt gefolgt von einem Schrägstrich (`./datei.html`) zeigen auf eine Datei, die sich im gleichen Verzeichnis befindet.

Zwei Punkte gefolgt von einem Schrägstrich (`../datei.html`) zeigen auf eine Datei, die sich ein Verzeichnis höher befindet.

Zweimal zwei Punkte gefolgt von einem Schrägstrich hintereinander (`.../datei.html`) zeigen auf eine Datei, die sich zwei Verzeichnisse höher befindet.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
</body>
</html>
<!--index_998a.html-->
```

Auch wenn Leaflet nun geladen wird, zeigt ein Aufruf dieser Datei in Ihrem Browser immer noch ein leeres Browser-Fenster. Erst im letzten Schritt - im Kapitel Fügen Sie eine Schicht mit Kacheln – einen Tile-Layer – zum Karten-Objekt wird die Karte sichtbar.

Velleicht fragen Sie sich, warum ich den Link zur CSS-Datei mithilfe eines selbst-schließenden Tags – also einem *unary* Tag – geschrieben habe, aber für das Tag, in dem das Skript eingebunden wird, zwei separate Tags – also ein *binary* Tag – verwendet habe.

```
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
```

HTML unterscheidet zwischen Tags, die nie Inhalt enthalten können – nämlich den *void*-Tags –, und solchen, die prinzipiell Inhalt enthalten können. Im ersten Fall muss ein selbst-schließendes *unary* Tag verwendet werden. Hierzu gehört das `<link>`-Tag. Im zweiten Fall darf kein selbst-schließendes *unary* Tag benutzt werden – auch dann nicht, wenn das Tag tatsächlich leer ist. Hierzu gehört das `<script>`.

## Leaflet performant einbinden – defer oder async

In diesem Kapitel erkläre ich Ihnen, wie Sie Leaflet in Ihre Website einbinden können, ohne den Ladeprozess der Webseite zu unterbrechen. Falls Sie noch unsicher in der Anwendung von JavaScript sind, überspringen Sie dieses Kapitel am besten erst einmal. Das Beachten der Performance sollten Sie erst angehen, wenn Sie die ersten Karten selbst erstellt haben.

### Was passiert genau, wenn eine Website mit einem `<script>`-Tag geladen wird?

Sehen wir uns zunächst einmal an, was genau passiert, wenn ein Browser eine Website mit einem `<script>`-Tag lädt.

1. Als Erstes lädt der Browser den Text der HTML-Seite.

2. Als Nächstes beginnt er, den HTML-Code zu analysieren, also zu parsen.
3. Nun trifft der Parser auf das <script>-Tag, welches auf eine externe Skript-Datei verweist.
4. Der Browser fordert die Skript-Datei an. Einstweilen blockiert und stoppt der Parser seine Arbeit.
5. Je nach Größe der Datei ist das Skript nach einiger Zeit vollständig heruntergeladen und wird anschließend ausgeführt.
6. Nun endlich kann der Parser seine Arbeit fortsetzen und den Rest des HTML-Dokuments analysieren.

Wenn Sie sich diese Abfolge ansehen, können Sie sich vorstellen, dass Punkt vier das performante Laden der Website negativ beeinflusst. Der Ladevorgang der Website macht praktisch eine Pause. Solange bis alle Skripte heruntergeladen sind, passiert nichts mehr. Und wenn es eine Sache gibt, die Website-Besucher und Suchmaschinen nicht mögen, dann ist dies die Wartezeit beim Aufbau der Website.

### **Wie können Sie die Ladezeit positiv beeinflussen?**

Um das im vorherigen Abschnitt beschriebene Problem zu umgehen wurde früher oft empfohlen, den JavaScript-Code möglichst nah am schließenden <body>-Tag in die Website zu integrieren. Zu dieser Empfehlung gibt es mit HTML5 zwei gute Alternativen – nämlich die Attribute defer und async.

Sofern Sie das Attribut defer verwenden, wird das Skript ausgeführt, wenn das HTML-Dokument geladen und für die Ansicht umgewandelt - also geparsed - ist. Zum anderen können Sie das Attribut async einsetzen. Mit async wird Ihr Skript asynchron mit dem HTML-Dokument ausgeführt. Wenn Sie keines dieser Attribute explizit angegeben, wird erst das vollständige Skript geladen und ausgeführt und erst dann wird das Laden und Parsen des HTML-Dokuments fortgesetzt.

### **Was sollten Sie beim Einsatz von defer oder async mit Leaflet beachten?**

Wenn Sie Ihre Karte auf Ihrer Website anzeigen, werden Sie nicht nur das Leaflet-Skript laden. Sie werden später noch eigenen JavaScript-Code schreiben. Dieser eigene Code setzt das Laden des Leaflet-Skripts voraus. Aus diesem Grund müssen Sie sicherstellen, dass die Leaflet Bibliothek vollständig geladen ist, bevor Ihr eigener Code ausgeführt wird. Dies können Sie mithilfe des Eventhandlers load. Ich zeige Ihnen dies hier an einem ganz einfachen Beispiel.

Obwohl Ihr eigenes Skript voraussetzt, dass Leaflet vollständig geladen ist, können Sie das Attribut async verwenden. Sehen Sie selbst: Das folgende Beispiel zeigt es

Ihnen.

```
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.2.0/dist/leaflet.css"/>
</head>
<body>
<div id="map" style="width: 600px; height: 400px"></div>
<script src="mymap_99.js" async></script>
<script src="https://unpkg.com/leaflet@1.2.0/dist/leaflet.js" async></script>
</body>
</html>
<!--mymap_99.html-->
```

In ihrem eigenen Skript `mymap_99.js` müssen Sie mithilfe von `window.addEventListener('load', function() ... )` das Laden des vollständigen HTML-Dokuments abwarten.

```
window.addEventListener('load', function()
{
var map = L.map('map',
{
center: [50.27264, 7.26469],
zoom: 10
});
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
},
false);
<!--mymap_99.js-->
```

Alle weiteren Beispiele hier im Buch habe ich ohne das Attribut `async` erstellt, weil ich den Schwerpunkt auf die Verwendung von Leaflet selbst setzen wollte.

## Erstellen Sie ein HTML-Element in dem Ihre Karte angezeigt werden soll

Das Einfügen eines HTML-Elements in unser Grundgerüst dürfte für Sie kein Problem darstellen. Der Vollständigkeit halber habe ich diesen Schritt hier trotzdem eingefügt.

Setzen Sie ein `<div>`-Element mit einer bestimmten `ID` an die Stelle in Ihrem HTML-Dokument, an der Sie Ihre Karte anzeigen möchten. Stellen Sie dabei sicher, dass der Kartencansteller eine definierte Höhe hat.

Der einfachste Weg einem HTML-Element eine feste Höhe zuzuordnen, ist das `style`-Attribut – also direkt im HTML-Element selbst. Weil hier im Buch Leaflet das Hauptthema ist, verwende ich für das Einbinden von Stylesheets in den Beispielen diese einfache Art. Durch das direkte Festlegen von Formaten gehen allerdings im praktischen Einsatz viele Vorteile verloren. Alternative Varianten zum Einbinden von Stylesheets finden Sie unter anderem unter der Adresse [https://wiki.selfhtml.org/wiki/CSS/Einbindung#Stylesheets\\_in\\_HTML\\_einbinden](https://wiki.selfhtml.org/wiki/CSS/Einbindung#Stylesheets_in_HTML_einbinden).

Im nachfolgenden Programmcodeausschnitt sehen Sie die relevante Zeile fett formatiert.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css"/>
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"></script>
</head>
<body>
<div style="height: 180px;" id="mapid"></div>
</body>
</html>
<!--index_997.html-->
```

So, nun ist das HTML-Dokument bereit ein Leaflet Kartenobjekt zu initialisieren und interessante Dinge mit ihm anzustellen.

## Erstellen Sie das Karten-Objekt

Nun wird es spannend. Wir erstellen das erste Skript. Dabei beginnen wir mit dem Erstellen des Karten-Objektes. Im nachfolgenden Programmcodeausschnitt sehen Sie die erste Zeile des Skripts fett formatiert.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css"/>
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"></script>
</head>
<body>
<div style="height: 180px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 13);
</script>
</body>
</html>
<!--index_996.html-->
```

Was haben wir genau gemacht? Wir haben mit dem Befehl `var mymap = L.map('mapid')` ein neues Objekt – oder eine neue Instanz – der Klasse `map` erstellt und diese `mymap` genannt.

Sie fragen sich nun vielleicht, wie wir eine neue Instanz ohne die Verwendung des Schlüsselwortes `new` erstellen könnten? Die Antwort ist einfach: Die Leaflet-Klassen sind mit einem Großbuchstaben – beispielsweise `L.Map` – benannt und diese müssen mit `new` erstellt werden. Es gibt aber Shortcuts mit Kleinbuchstaben – `L.map` – die aus Bequemlichkeitsgründen von den Leaflet-Programmierern für Sie erstellt wurden. Leaflet setzt hier das Entwurfsmuster Fabrikmethode(englisch factory method) ein. Das Muster beschreibt, wie ein Objekt durch Aufruf einer Methode anstatt durch direkten Aufruf eines Konstruktors erzeugt wird.

Wollen Sie sich dies selbst ansehen? Die Funktion `L.map()` der Klasse `L.Map` finden Sie auf Github ganz am Ende in der Datei `map.js`.

Das Festlegen des Kartenmittelpunktes mithilfe der Koordinaten [50.27264, 7.26469] und der Methode `setView()` und die Angabe der Zoomstufe 13 ist optional. Ich empfehle Ihnen, diese Werte immer mitzugeben. Denn: Es ist für jeden ärgerlich eine Karte zu sehen, die die ganze Welt anzeigt – die relevanten Daten befinden sich aber alle in Gering, einem kleinen Dorf in der deutschen Eifel.

Sagen Ihnen die Koordinaten in der Form [50.27264, 7.26469] nichts und möchten Sie gerne mehr zum Thema geografische Koordinaten erfahren? Dann lesen den Exkurs im Kapitel Exkurs: Geographische Koordinaten.

Sie verfügen nun über ein Leaflet Karten-Objekt, mit dem Sie eine Karte anzeigen können. Sie müssen dem Karten-Objekt noch mitteilen, welches Kartenbild - also welche Grafiken - es anzeigen soll. Dies tun Sie, indem Sie eine Schicht mit Kacheln, also einen Tile-Layer, zum Karten-Objekt hinzufügen. Wie Sie dies genau tun, zeige ich Ihnen im nächsten Kapitel.

## Fügen Sie eine Schicht mit Kacheln – einen Tile-Layer – zum Karten-Objekt hinzu

Der letzte Schritt beim Erstellen der Karte ist das Hinzufügen der Kachel-Schicht. Diese Schicht – oder dieser Layer – kann als eine Art Basiskarte angesehen werden. Es handelt sich um die Grafiken, auf der die Geoobjekte, die wir hier im Buch erarbeiten, dargestellt werden. Also die Imagedateien.

Kacheln zum Anzeigen in einem digitalen Kartenobjekt werden als Service von unterschiedlichen Providern angeboten. Im nächsten Kapitel werde ich Ihnen genauer erläutert, dass diese Kacheln normalerweise als 256 Pixel x 256 Pixel Images angeboten werden und warum die URL zum Aufruf der Kacheln die etwas kryptisch wirkenden Zeichen `/z/x/y.png` enthält.

Ich verwende hier das Angebot von OpenStreetMap zur Darstellung der Karte. Den Programmcode zum Einbinden der Imagedateien vom OpenStreetMap Tile-Server habe ich für Sie im nachfolgenden Programmcodebeispiel fett hervorgehoben. Die rechtlichen Voraussetzungen zur Verwendung der Kacheln des Openstreetmap-Servers finden Sie unter der Adresse <https://operations.osmfoundation.org/policies/tiles>.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css"/>
```

```
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"></script>
</head>
<body>
<div style="height: 180px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 13);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
</script>
</body>
</html>
<!--index_995.html-->
```

Was haben wir genau gemacht? Wir haben ein `TileLayer` Objekt erstellt und diesem die URL des OpenStreetMap-Servers übergeben. Außerdem haben wir die Methode `addTo()` aufgerufen und dieser Methode unser Karten-Objekt `mymap` als Parameter übergeben. So weiß Leaflet nun genau, welche Bilddateien es wo abrufen soll und kann die Kartenschicht zeichnen.

Leaflet ist so programmiert, dass Sie die verschiedenen Methoden verketten können. Dies ist möglich, weil die unterschiedlichen Methoden Objekte zurückgeben, die wieder Funktionen enthalten.

So konnten wir `.addTo(mymap)` einfach an `L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')` anhängen.

Alternativ hätten wir zuerst ein `TileLayer` Objekt erstellen müssen und hätten erst im nächsten Schritt die Methode `addTo()` aufrufen können

```
var x = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png');
x.addTo(mymap);
```

Fertig! Sie haben nun eine vollständige Karte erstellt. Zählen Sie nach: In diesen vier Schritten haben Sie gerade einmal fünf Zeilen Programmcode eingegeben.

Standardmäßig – wir haben ja bisher noch keine Optionen übergeben – sind alle Maus- und Touch-Interaktionen auf der Karte aktiviert. Sie können die Karte vergrößern und verkleinern und in der rechten unteren Ecke befindet sich ein Hinweis darauf, dass die Karte mit Leaflet erstellt wurde.

Sie können nun die ganze Welt auf dieser Karte erkunden.

In der nachfolgenden Abbildung sehen Sie diese Karte – so sollte diese bei Ihnen aussehen, wenn Sie meinem Beispiel gefolgt sind.



Bevor wir die Karte nun weiter bearbeiten, sehen wir uns ein bisschen Theorie an. Fall Sie keine Theorie mögen, können sie sofort praktisch im Kapitel *Die Karte mit Daten bestücken* weitermachen.

## Exkurs: Geographische Koordinaten

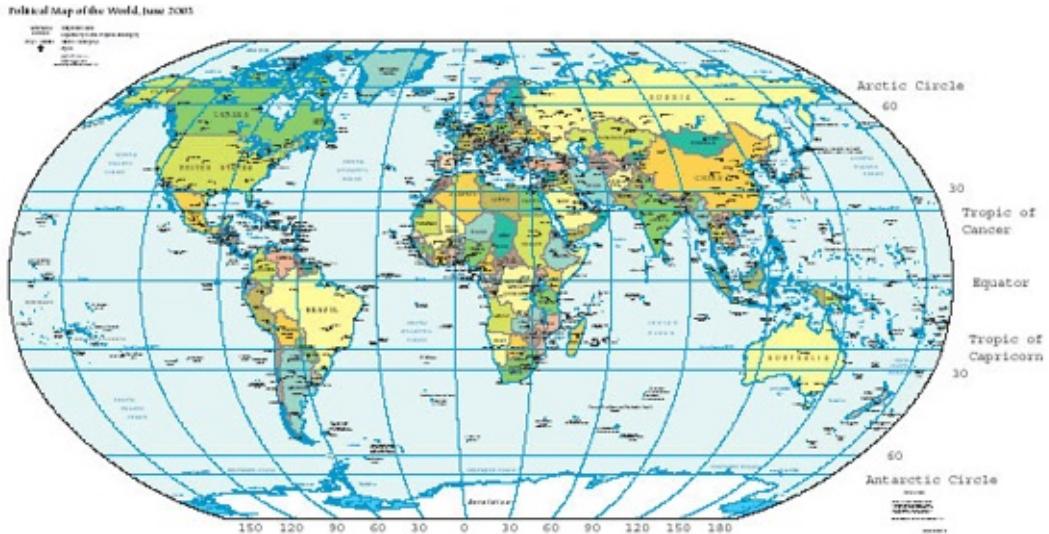
Mithilfe von Längen- und Breitengraden können Sie die genaue Position eines jedes Punktes auf der Erdoberfläche angeben.

### Das Koordinatensystem der Erde

Das Grad-Netz der Erde ist ein gedachtes Koordinatensystem auf der Erdoberfläche mit sich rechtwinklig schneidenden Längen- und Breitenkreisen. Zum Aufbau dieses Koordinatensystems wird unser Erdball zunächst in 180 Breitenkreise und 360 Längenkreise eingeteilt.

- Die Breitengrade oder Breitenkreise verlaufen parallel zum Äquator.
- Die Längengrade oder Längenkreise verbinden Nord- und Südpol.

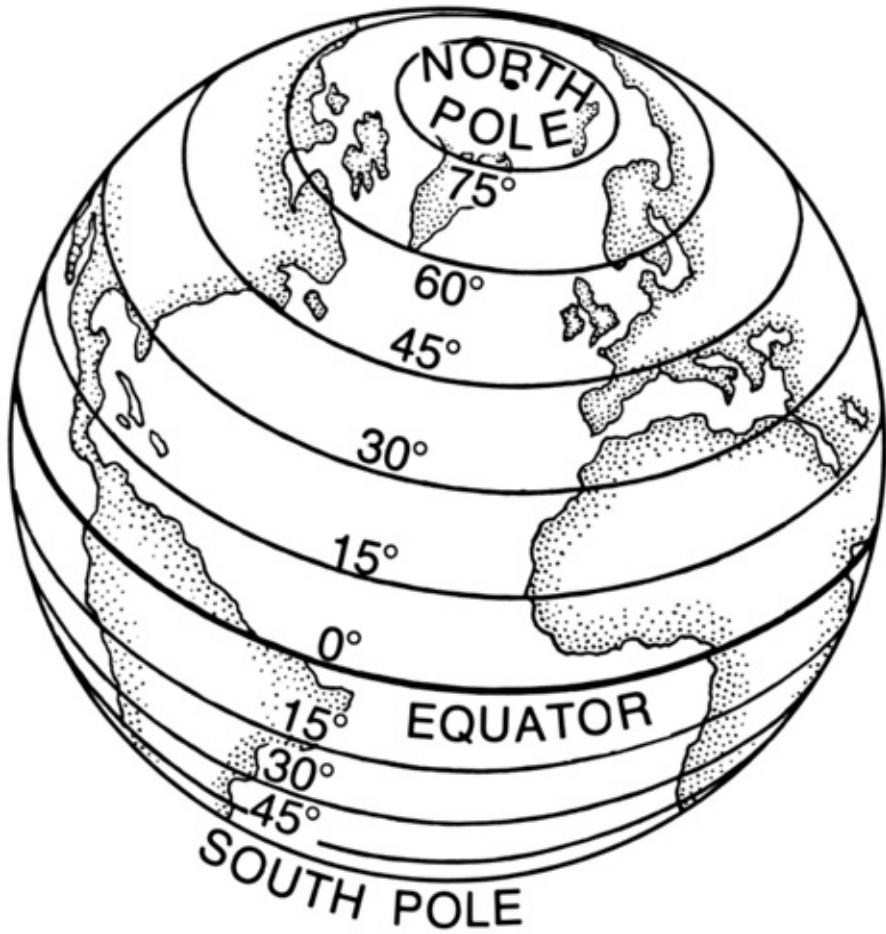
So entsteht ein grobmaschiges Gitter, anhand dessen jeder die ungefähre Position auf der Erdoberfläche bestimmen kann.



Um die Genauigkeit zu erhöhen, wird jeder Breiten- und Längengrad weiter unterteilt.

## Breitengrade

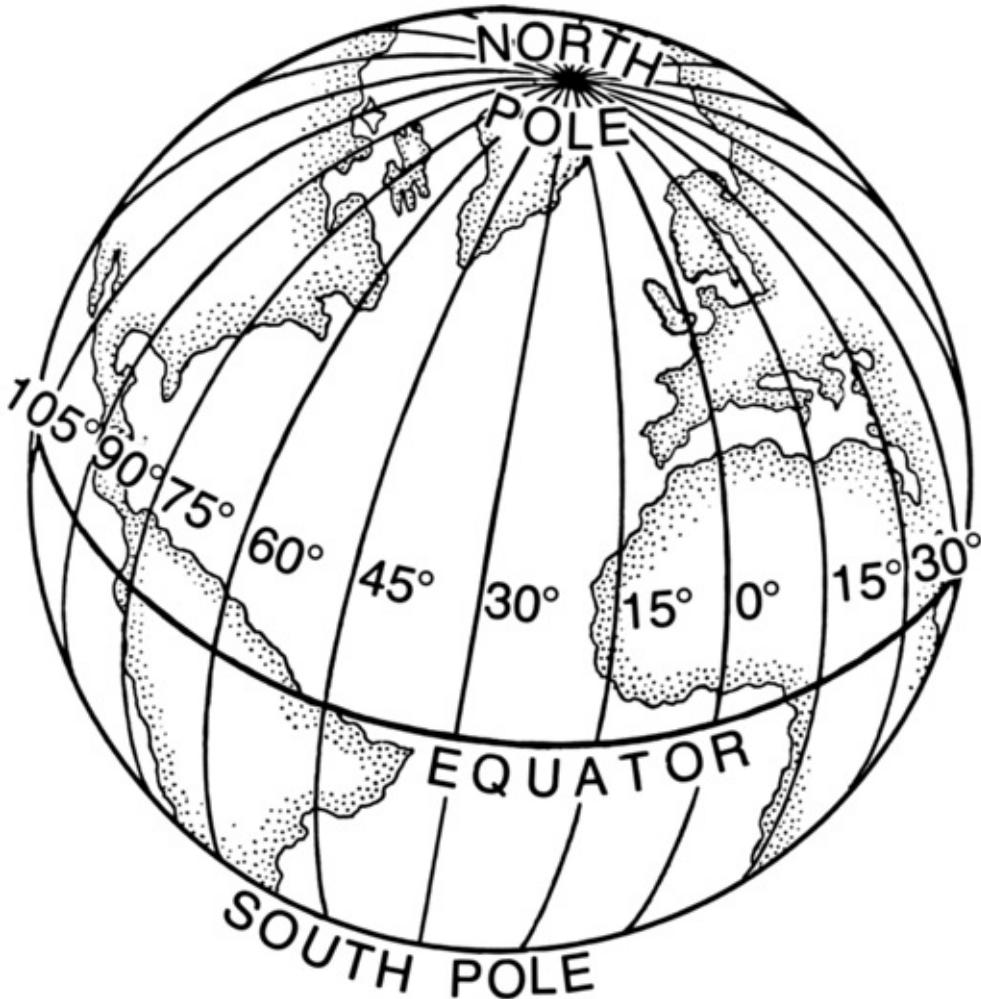
Die Breitengrade verlaufen von Osten nach Westen. Vielleicht wissen Sie noch aus dem Erdkundeunterricht in der Schule, dass der Äquator im rechten Winkel zur Erdachse verläuft. Er liegt etwa in der Mitte zwischen Nord- und Südpol. Im geografischen Koordinatensystem gilt er als Ausgangspunkt für die Berechnung der Breitenkreise und ihm wird ein Winkel von  $0^\circ$  zugeordnet.



*Breitengrade (Latitude) - By Pearson Scott Foresman [Public domain], via Wikimedia Commons*

## Längengrade

Die Längengrade auf der Erde verlaufen von Norden nach Süden. Sie umspannen die Erde praktisch. Eine Längenkreishälfte wird als Meridian bezeichnet. Die Längenkreise haben keinen natürlichen Nullpunkt. Heute gilt der Meridian, der den Londoner Stadtteil Greenwich durchläuft, als Nullmeridian und somit als Ausgangspunkt für die Berechnung der Längengrade.



*Längengrade (Longitude) - By Pearson Scott Foresman [Public domain], via Wikimedia Commons*

## Schreibweisen von geografischen Koordinaten

Bei der Angabe von geographischen Koordinaten wird heute normalerweise eine von zwei Schreibweisen verwendet: Entweder das Sexagesimalsystem, welches von Wikipedia verwendet wird, oder die Dezimalschreibweise, die von Computerprogrammen bevorzugt wird. Im Laufe unserer Geschichte haben sich allerdings eine Menge mehr unterschiedlicher Systeme entwickelt.

Falls Sie einmal in die Verlegenheit kommen sollten und eine Koordinate aus einem System in ein anderes umrechnen müssen, kann ich Ihnen die Website <https://www.deineberge.de/Rechner/Koordinaten/Dezimal/> empfehlen, weil diese das Umrechnen zwischen vielen verschiedenen Systemen unterstützt.

## Das Sexagesimalsystem

Das Sexagesimalsystem ist die traditionelle Schreibweise. Dieses System heißt Sexagesimal, weil ein Grad eines Breitengrades 60 Minuten entspricht. Somit basiert das Sexagesimalsystem auf der Zahl 60. Und der lateinische Name der Zahl 60 ist *sexagesimus*.

Jeder Breiten- und Längengrad wird in 60 Minuten mit je 60 Sekunden unterteilt. Eine Koordinate besteht somit aus drei Teilen.

- Der erste Teil gibt die Längen- und Breitengrade als Winkel in Grad ( $^{\circ}$ ) an. Die Angabe ist ganzzahlig und liegt beim Längengrad zwischen -180 und +180 und beim Breitengrad zwischen -90 und +90.

Dabei steht beim Längengrad  
 $-90^{\circ}$  für die Angabe  $90^{\circ}$  **Süd**  
und  
 $+90^{\circ}$  für die Angabe  $90^{\circ}$  **Nord**.

Die Breite wird entsprechend in  $-180^{\circ}$  bis  $+180^{\circ}$  angegeben, anstelle von  $180^{\circ}$  **West** bis  $180^{\circ}$  **Ost**.

- Der zweite Teil gibt die Minuten an. Die Minuten werden durch eine Prime ('') gekennzeichnet. Jeder Grad hat 60 Minuten. Das bedeutet, dass diese Zahl nicht kleiner als 0 sein darf und kleiner als 60 sein muss.
- Der dritte Teil gibt die Sekunden an. Jede Minute hat 60 Sekunden, die anhand einer Doppelprime ( '') erkennbar sind. Genau wie bei den Minuten gilt also auch hier: Die Sekundenzahl darf nicht kleiner als 0 sein und muss kleiner als 60 sein.

Eine Breitenminute entspricht auf der Erdoberfläche einer Strecke von circa 1,852 Kilometern. Die Strecke, die einer Längenminute entspricht, beträgt am Äquator ebenfalls 1,852 Kilometer, verringert sich aber zum Pol hin auf 0 Kilometer.

So hat beispielsweise die Zugspitze die Koordinaten  $47^{\circ} 25' 16''$ ,  $10^{\circ} 59' 7''$ . Einem Ort, der auf dem westlichen Teil der Südhalbkugel liegt, könnten die Koordinaten  $-11^{\circ} 27' 30''$ ,  $-72^{\circ} 47' 23''$  zugeordnet werden.

## Die Dezimalschreibweise

Parallel zum traditionell gebräuchlichen Sexagesimalsystem hat sich die Angabe der Koordinaten im Dezimalsystem etabliert. Das Dezimalsystem basiert auf der Zahl 10. Dieses System wird vor allem deshalb von Computern gerne benutzt, weil es sich

damit recht unkompliziert rechnen lässt.

Die Genauigkeit einer Koordinate in der Dezimalschreibweise hängt sehr von der Anzahl der Nachkommastellen ab. Bei nur zwei Nachkommastellen ergibt sich eine mögliche Abweichung von bis zu einem Kilometer, bei vier Stellen nach dem Komma sind es nur noch zehn Meter Abweichung und sechs Nachkommastellen entsprechen einer Genauigkeit von einem Meter.

Orten auf der West- und Südhalbkugel wird in der Regel ein Minus (-) vorangestellt. Die Breite wird in Dezimalgrad von  $-90^\circ$  bis  $+90^\circ$  angegeben. Dabei steht  $-90^\circ$  für die Angabe  $90^\circ$  **Süd** und  $+90^\circ$  für die Angabe  $90^\circ$  **Nord**. Die Breite wird entsprechend in  $-180^\circ$  bis  $+180^\circ$  angegeben, anstelle von  $180^\circ$  **West** bis  $180^\circ$  **Ost**.

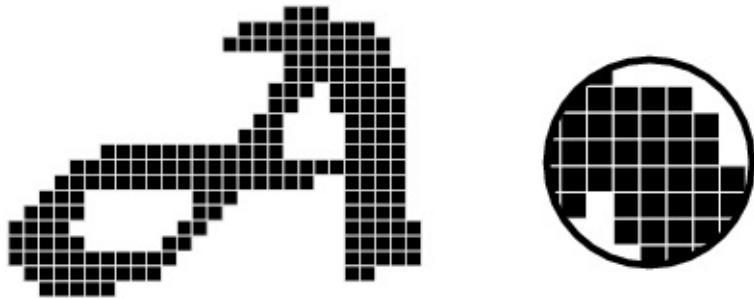
Beispielsweise werden im Dezimalsystem die Koordinaten der Zugspitze mit 47.4211, 10.9852 angegeben. Einem Ort, der auf dem westlichen Teil der Südhalbkugel liegt, könnten die Koordinaten -13.163333, -72.545556 zugeordnet werden.

## Exkurs: Wie werden Landkarten auf einer Webseite angezeigt?

Eine Karte ist im Grunde genommen nichts anderes als die Darstellung einer Abbildung oder einer Grafik. Abbildungen oder Grafiken müssen, damit sie von Computern verarbeitet werden können, in einem Grafikformat gespeichert werden. Bevor wir uns genau ansehen, wie die Grafiken für Landkarten erstellt werden, erkläre ich Ihnen nachfolgend kurz die wesentlichen Unterschiede dieser beiden Formate.

## Grafikformate: Vektoren und Rastergrafiken

Ein Grafikformat ist ein Dateiformat, das den Aufbau einer Bilddatei beschreibt. Bei den Grafikformaten können Sie alles in allem zwischen Vektorgrafiken und Rastergrafiken unterscheiden. Im nächsten Bild sehen Sie oben eine Vektorgrafik und unten eine Rastergrafik.



## Vektoren

Vektorgrafiken basieren, im Gegensatz zu Rastergrafiken, nicht auf einem Pixelraster, indem jedem Bildpunkt ein Farbwert zugeordnet ist. Vektorgrafiken basieren auf einer Formel, die die Elemente, aus denen das Bild aufgebaut ist, genau beschreibt. Ein Kreis kann in einer Vektorgrafik anhand des Mittelpunktes, des Radiuses, der Linienstärke und der Farbe vollständig beschrieben werden. Deshalb müssen auch nur diese Parameter gespeichert werden. Je nach Bildgröße benötigen Vektorgrafiken daher oft weniger Speicherplatzbedarf als Rastergrafiken. Außerdem können sie im Gegensatz zur Rastergrafik stufenlos und verlustfrei skaliert, also vergrößert oder verkleinert werden.

## Rastergrafiken

Rastergrafiken kennen Sie sicherlich auch unter dem Namen Pixelgrafik oder Bitmap. Dieses Format beschreibt die Bilder in Form einer Anordnung von Pixeln als Raster. Pixel sind im Grunde genommen nichts anderes als Bildpunkten, denen eine Farbe zugeordnet ist. Anders als bei Vektorgrafiken ist die Bildgröße – die Breite und Höhe gemessen in Pixeln – und die Farbtiefe – die maximale Anzahl an Farben – ein wesentliches Merkmal des Bildes. Eine Rastergrafik kann nicht stufenlos und verlustfrei vergrößert werden.

## Vektoren und Rastergrafiken für digitale Karten

Karten sollen intuitiv und einfach bedienbar sein. Idealerweise ist jeder Ausschnitt der Karte in jeder Auflösung schnell abrufbar.

Theoretisch ist dies für Vektorkarten möglich. Praktisch kostet es aber sehr viel Rechenzeit. Abgesehen von Satellitenaufnahmen oder Luftbildern, die nichts anderes als ein Foto sind, sind Karten in der Regel keine Rastergrafiken. Die Informationen anhand derer die Karte erstellt wird, werden als Daten gespeichert. Diese Daten entsprechen eher den Daten, mit denen Vektorgrafiken erstellt werden. Eine Straße wird beispielsweise mithilfe einer Anzahl von Punkten, die miteinander verbunden sind, dargestellt. Zusätzlich werden mit diesen Punkten Eigenschaften abgespeichert. Eine Eigenschaft kann der Straßenname sein – eine andere Eigenschaft kann der Straßenbelag sein.

Leider ist die Darstellung dieser Informationen auf einer Webseite in einem Vektorformat aber schwierig. Nicht alle Browser können gut mit Vektorgrafiken umgehen. Außerdem gibt es viele Geodaten, die große Bereiche auf der Erde abdecken. Diese müssen bei der Verwendung eines Vektorformates auch dann verarbeitet werden, wenn Sie sich nur einen kleinen Bereich in Deutschland ansehen möchten. Mit Rastergrafiken hat kein Browser Probleme. So ziemlich jedem Browser kann eine Rasterkarte anstandslos auf einem Bildschirm anzeigen.

Das Problem bei der Bereitstellung von geographischen Informationen als Rastergrafik ist, dass eine gute Bildqualität eine hohe Auflösung voraussetzt. Dies hat zur Folge, dass die Grafikdateien sehr groß werden. Bilddateien, die über das Internet geladen und im Browser angezeigt werden, sollten aber so klein wie möglich sein.

Aus diesem Grund wird die Karte für kleine Ausschnitte im Vorfeld berechnet und in einem Rasterformat gespeichert. Als Rasterformat wird PNG verwendet. Wie dies genau gemacht wird, erkläre ich Ihnen im nächsten Kapitel.

## **Wir unterteilen die Welt in Kacheln**

Um eine Karte anzuzeigen, wird die Welt also in Ausschnitte, genau genommen in Quadrate zerlegt. Die Quadrate werden Tiles, das ist das englische Wort für Kacheln, genannt. Jedes Quadrat ist exakt 256 Pixel x 256 Pixel groß.

Nicht nur OpenStreetMap, auch die Google Maps API unterteilt ihr Kartenbilder in Kacheln. Wenn Sie die Website <https://www.google.de/maps> aufrufen und eine andere Vergrößerungsstufe wählen, wird ermittelt, welche Daten erforderlich sind. Diese Daten werden dann in einen Satz mit Kacheln übersetzt und angezeigt.

Dabei bildet die Zoom-Stufe 0 die ganze Welt auf ein Quadrat ab. Teilt man den Erdumfang von 40.038 Kilometern durch die 256 Pixel der Kachel sieht man im Ergebnis, dass ein Pixel 156,4 Kilometer darstellt. Das ist noch nicht sehr detailliert. Bis Zoom-Stufe 19 ändert sich eine ganze Menge. In der nachfolgenden Tabelle sehen Sie, dass bei Zoom-Stufe 19 ein Pixel einem Bereich von 0,3 Metern auf der Erde entspricht. Damit kann man schon etwas anfangen!

<b>Zoom-Stufe</b>	<b>Kachel-Anzahl</b>	<b>Kachel-Breite entspricht</b>	<b>Ein Pixel entspricht</b>
0	1	40.038 Kilometer	156 Kilometer
1	4	20.019 Kilometer	78 Kilometer
2	16	10.009 Kilometer	39 Kilometer
3	64	5.004 Kilometer	19,5 Kilometer
4	256	2502 Kilometer	9,8 Kilometer
..	..	..	..
15	1 Milliarde	1224 Meter	4,8 Meter
16	4 Milliarden	612 Meter	2,4 Meter
17	17 Milliarden	306 Meter	1,2 Meter
18	69 Milliarden	153 Meter	0,6 Meter
19	275 Milliarden	76 Meter	0,3 Meter

Die vollständige Tabelle können Sie unter der Adresse [http://wiki.openstreetmap.org/wiki/Zoom\\_levels](http://wiki.openstreetmap.org/wiki/Zoom_levels) mit weiteren Angaben im Internet abrufen.

Vielleicht probieren Sie nun das Zoomen im vorangegangene Beispiel aus und wundern sich, dass Sie die Karte nur bis zur Zoom-Stufe 18 vergrößern können. Das liegt daran, dass bei dieser OpenStreetMap Karte standardmäßig die Option `maxZoom` mit 18 gesetzt ist. Sie können diese Option jedoch überschreiben. Wie das geht sehen Sie im nachfolgenden Programmcodebeispiel fett formatiert. Weitere Informationen finden Sie im Kapitel zur Karte von Stamen.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 180px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 13);
console.log(mymap);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {minZoom: 0, maxZoom: 19})
.addTo(mymap);

</script>
</body>
</html>
<!--index_995a.html-->
```

Vielleicht sind Sie es gewohnt, bei der Darstellung von Landkarten in den Zahlen eines Maßstabs zu denken? Bei digitalen Karten gibt es keinen Maßstab im Sinne einer Papierkarte, weil die Druckauflösung nicht bekannt ist und ein Maßstab hiervon abhängt. Ein Maßstab kann immer nur

■ relativ zur Auflösung angegeben werden.

## Wie weiß Leaflet welche der vielen Kacheln angezeigt werden sollen?

Nun haben wir jede Menge Kacheln und möchten mit diesen eine digitale Karte auf unserer Website anzeigen. Woher weiß Leaflet, welche Kacheln es vom verlinkten Server laden und an welcher Stelle es diese anzeigen soll? Dazu sehen wir uns zunächst einmal an, wie die Kacheln genau erstellt werden.

Um ein Bild von einer Karte in kleine überschaubare Abschnitte zu teilen, unterscheidet der Server, der die Kacheln erzeugt, zwischen verschiedenen Zoom-Stufen und für jede Zoom-Stufe erstellt er ein eigenes Set von Kacheln – praktisch eine eigene Ebene.

Da der Standard für die Größe der Kacheln 256 Pixel x 256 Pixel beträgt, ist bei der Zoom-Stufe 0 die gesamte Welt in einer einzigen 256 Pixel x 256 Pixel großen Kachel enthalten. In der Tabelle im vorherigen Kapitel konnten Sie ja schon erkennen, dass jede Erhöhung der Zoom-Stufe auch die Anzahl der anzugezeigenden Kacheln erhöht.

Um die Kacheln in der richtigen Weise zu benutzen, muss es ein Muster geben, das befolgt werden kann, um sicherzustellen, dass die richtige Kachel vom Server geladen werden und vom Browser des Clients an der richtigen Stelle angezeigt werden.

Im Kapitel Fügen Sie eine Schicht mit Kacheln – einen Tile-Layer – zum Karten-Objekt hinzu hatten wir die URL für den Tile Server mit `http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png` angegeben.

Der Teil `{z}/{x}/{y}` des Pfades zur PNG-Datei enthält Variablen aus denen der Namen der Bilddatei berechnet werden kann.

- `{z}` bezeichnet die zu ladende Zoom-Stufe.
- `{x}` bezeichnet die Position auf der x-Achse der Kachel.
- `{y}` bezeichnet die Position auf der y-Achse.
- `{s}` steht für eine optionale Subdomain.

Zum Beispiel wird das Bild für die niedrigste Zoom-Stufe – also das Bild welches den größten Bereich pro Pixel anzeigt – unter dem Dateinamen `0/0/0.png` abgespeichert.



Die vollständige URL dieses Kachelbildes auf dem Openstreetmap Server ist <http://a.tile.openstreetmap.org/0/0/0.png>. Probieren Sie es, klicken Sie den Link an oder öffnen Sie selbst die Adresse <http://a.tile.openstreetmap.org/0/0/0.png> in Ihrem Internetbrowser.

Tiefer gehend können Sie das Thema auf der Website von OpenStreetMap, genau unter der Adresse [http://wiki.openstreetmap.org/wiki/Slippy\\_map\\_tilenames](http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames), nachlesen.

Bei der Zoom-Stufe 1 sind die Kacheln, wie in der nachfolgenden Grafik dargestellt, angeordnet.

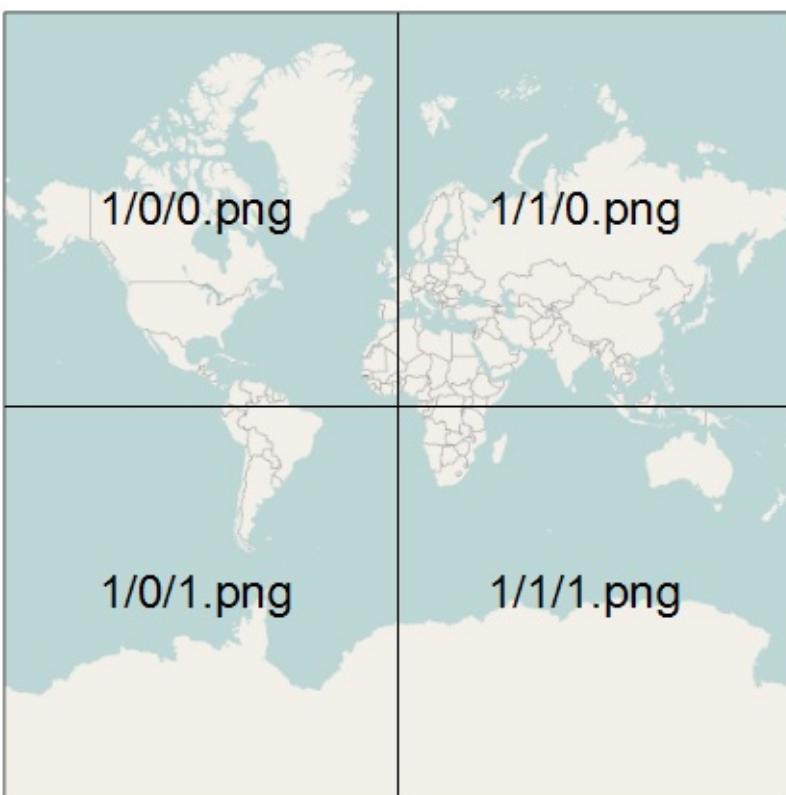


Abbildung 7: Zoom Level 1. © OpenStreetMap contributors

Unter der Adresse <http://a.tile.openstreetmap.org/1/0/0.png> finden sie die Grafik, die sich in der Abbildung links oben befindet.

# Schöne Kartenlayer

Nachdem das Erstellen der ersten Karte so einfach vonstatten ging fragen Sie sich sicher, ob es genauso einfach ist eine alternative Darstellung – also Kacheln eines anderen Providers – zu verwenden. Die Antwort ist: Ja, es ist genauso einfach!

Ich zeige Ihnen dies hier anhand von zwei weiteren Providern, nämlich Thunderforest und Stamen.

Mögen Sie die Karten von GoogleMaps und möchten Sie gerne die Kacheln von Google für Ihre digitale Karte nutzen? Wenn Sie dies zusammen mit Leaflet tun möchten, können Sie dies mithilfe des Plugins L.GridLayer.GoogleMutant.

## Thunderforest

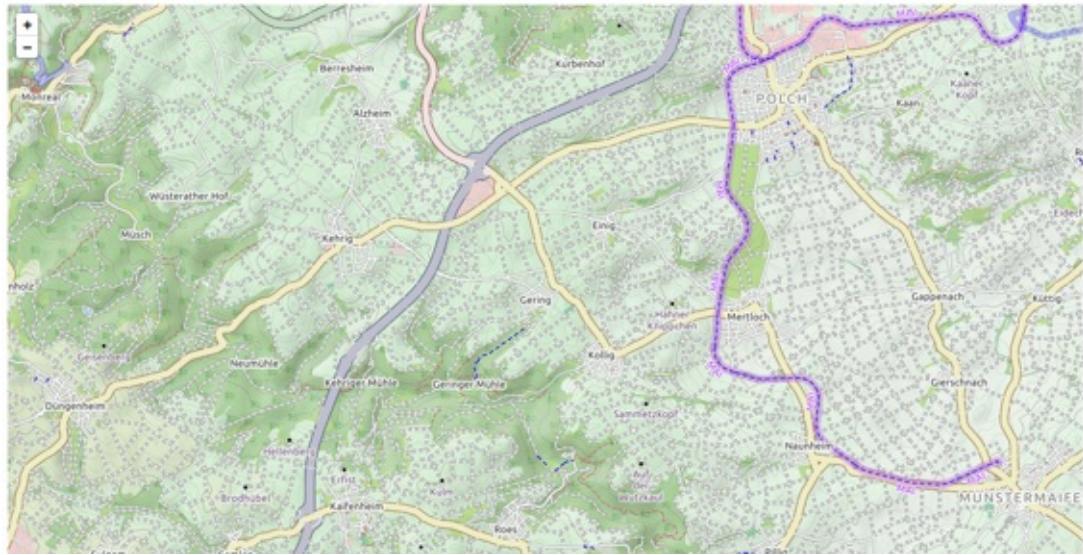
Thunderforest bietet Ihnen gleich neun verschiedene Kachel-Varianten. Sie erreichen die Kacheln alle über die gleiche URL, lediglich das Unterverzeichnis muss angepasst werden.

Um Kacheln von Thunderforest zu verwenden, müssen Sie ein Zugriffstoken anfordern. Dieses Token können Sie über die Adresse <https://www.thunderforest.com/docs/apikeys/> selbst erstellen. Wenn Sie ihre Karte erstellen, hängen Sie dieses Zugriffstoken einfach an das Ende der URL des Tile-Servers an. Zum Beispiel so:

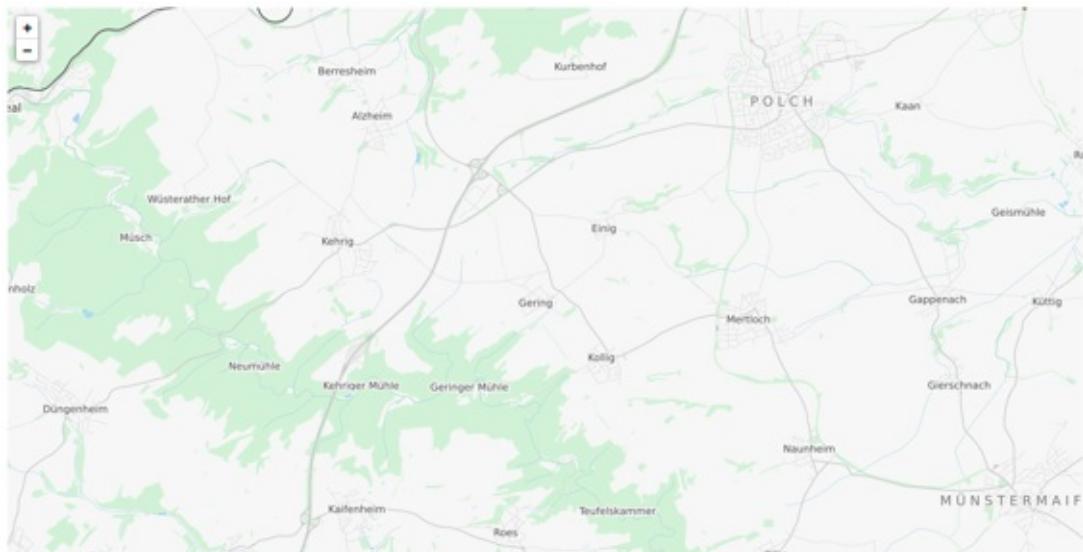
`https://{s}.tile.thunderforest.com/cycle/{z}/{x}/{y}.png?apikey=YourApiKey`

Die Kacheln der OpenCyclemap finden Sie beispielsweise unter der Adresse <https://{s}.tile.thunderforest.com/cycle/{z}/{x}/{y}.png?apikey=YourApiKey> abgelegt. Die Transportvariante finden Sie unter der Adresse <https://{s}.tile.thunderforest.com/transport/{z}/{x}/{y}.png?apikey=YourApiKey>.

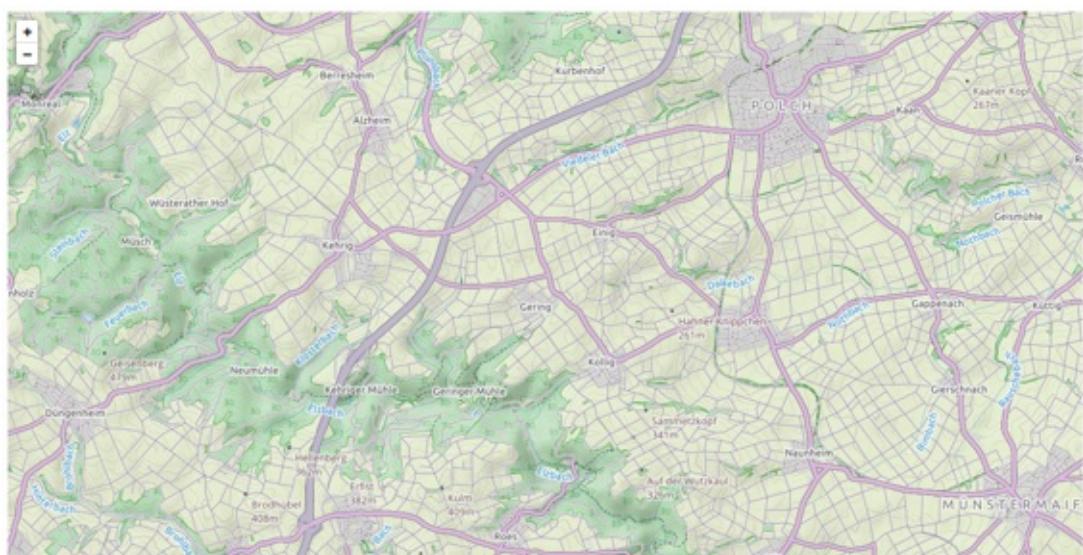
Die nachfolgende Tabelle zeigt Ihnen eine Übersicht über die verschiedenen Kartenstile von Thunderforest.



cycle



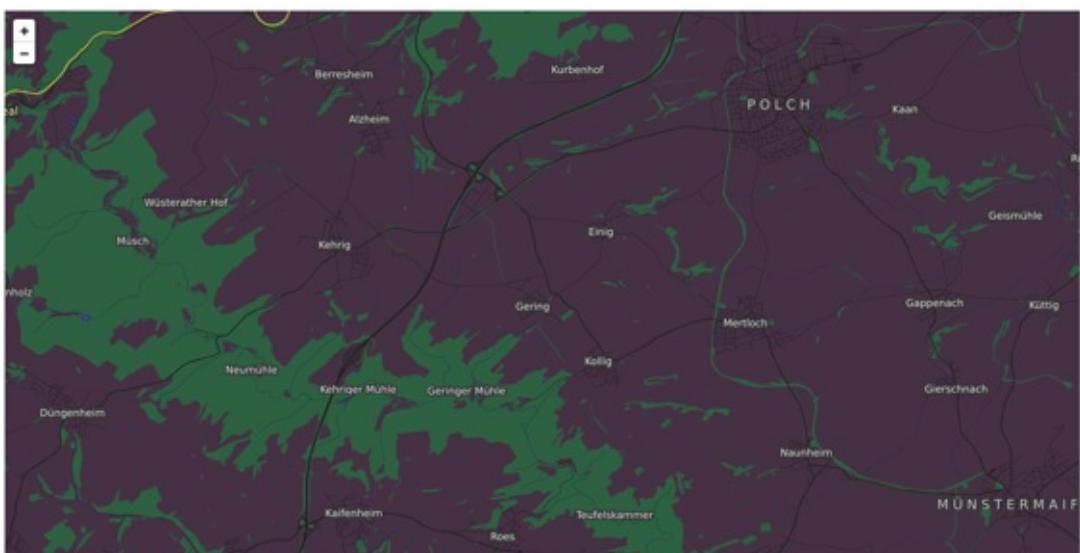
transport



## landscape



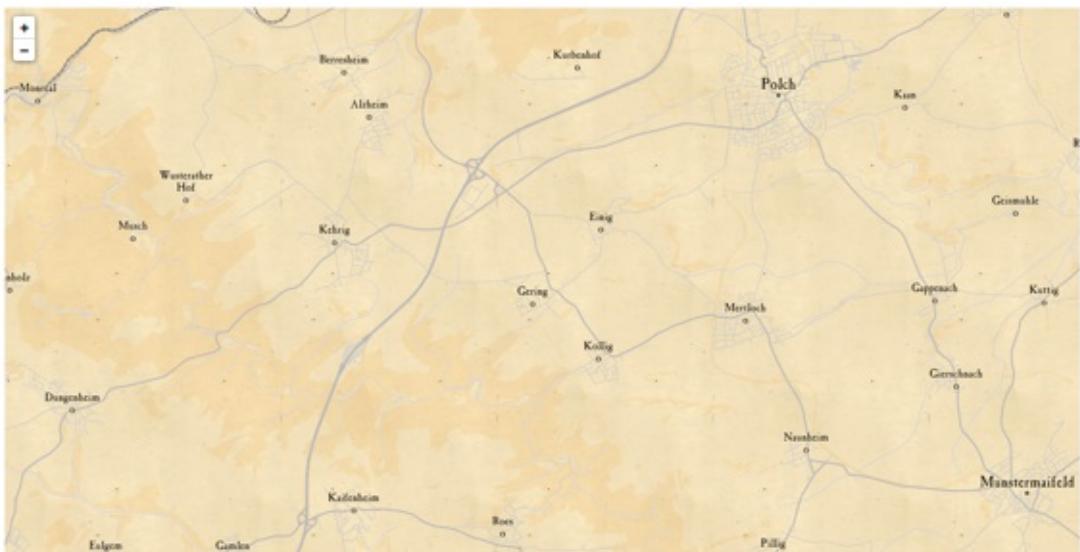
outdoors



transport-dark



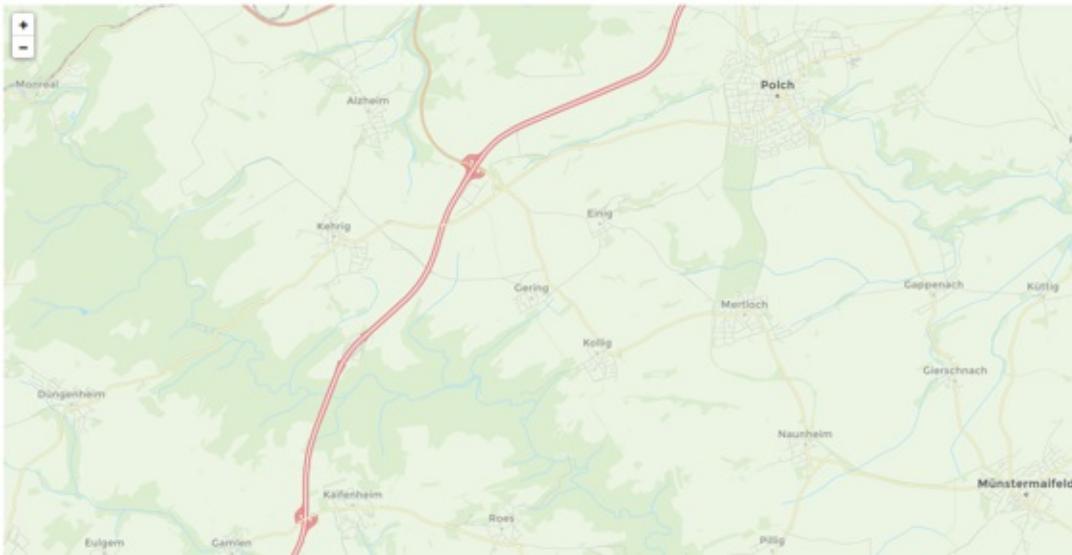
spinal-map



pioneer



## mobile-atlas



## neigborhood

Wenn Sie Thunderforest verwenden möchten, müssen Sie unser bisheriges Beispiel nun in einer Zeile abändern. Sie müssen als Tile Layer nur die im Beispiel zu sehende URL angeben. Der nachfolgende Programmcode zeigt Ihnen ein vollständiges Beispiel.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 13);

L.tileLayer('https://{s}.tile.thunderforest.com/landscape/{z}/{x}/{y}
.png?apikey=MeinZugriffstoken').addTo(mymap);

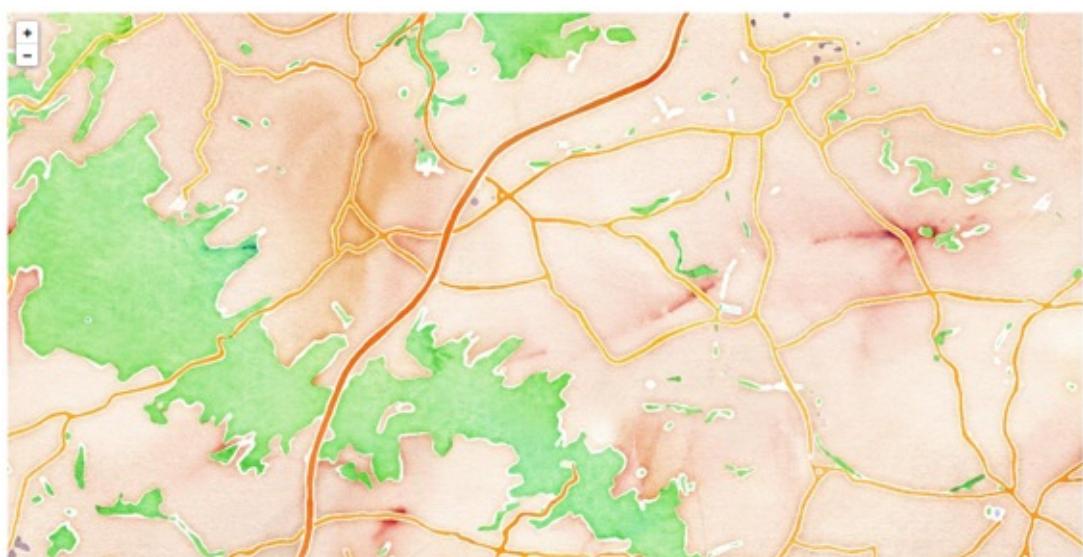
</script>
</body>
</html>
<!--index_994.html-->
```

## Stamen

Stamen legt den Schwerpunkt auf gutes Design. Informationen zu den Karten von Stamen finden Sie auf der Website <http://maps.stamen.com/>. Die nachfolgende Tabelle zeigt Ihnen drei Kartenstile von Stamen.



toner



watercolor



terrain

Beim Einbinden einer Karte von Stamen müssen Sie zusätzlich eine JavaScript Datei einbinden. Wie Sie den `StamenTileLayer` genau nutzen, können Sie im nachfolgenden Programmcodebeispiel ablesen.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script type="text/javascript" src="http://maps.stamen.com/js/tile.stamen.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 13);
var layer = new L.StamenTileLayer("watercolor");
mymap.addLayer(layer);
</script>
</body>
</html>
<!--index_993.html-->
```

## Achtung:

Der `StamenTileLayer` unterstützt nicht alle Zoom-Stufen. Wenn Sie den Typ `watercolor` verwenden, sehen Sie zum Beispiel mit der Zoom-Stufe 19 eine leere graue Fläche. Um dies zu verhindern können Sie die Optionen des `StamenTileLayer` überschreiben.

- Setzen Sie dafür nach der Instanziierung die Options `maxZoom` auf 19. So bleibt die Zoom-Stufe 19 als Ebene auf der Karte erhalten.
  - Setzen Sie dann aber die Option `maxNativeZoom` auf 18.

Dies bewirkt, dass Leaflet nicht versucht, Kachel für eine Zoom-Stufe 19 zu laden. Stattdessen benutze Leaflet auch bei Zoom-Stufe 19 die Kacheln der Zoom-Stufe 18 – skaliert diese aber auf die Größe der Zoom-Stufe 19.

```
...
var layer = new L.StamenTileLayer("watercolor");
layer.options.maxZoom = 19;
layer.options.maxNativeZoom = 18;
...
```

ESRI ist ein weiterer Anbieter von Basiskarten. Was ESRI genau ist und wie Sie die Karten dieses Institus einbinden können erkläre ich Ihnen im Kapitel ESRI.

Haben Sie noch nicht den Kartenstil gefunden, den Sie suchen oder sind Sie einfach nur neugierig, welche Karten sonst noch angeboten werden? Verweise auf weitere Tile-Server-Provider finden Sie unter der Adresse: <http://wiki.openstreetmap.org/wiki/Tiles>

## Images als Layer – Web-Map-Service

Sie haben eine gute Satellitenaufnahme und möchten diese als Schicht in Ihrer Karte anzeigen. Vielleicht denken Sie auch an die Wetterwarnkarten des Deutschen Wetterdienstes, die im Grunde genommen nur aus eingefärbten Polygonen bestehen. Ein Umwandeln dieser Grafikdateien in 275 Milliarden Kacheln, wie es im vorherigen Kapitel beschriebenen wurde, wäre zwar möglich – Sie können sich aber vorstellen, dass es für diese Aufgabenstellungen adäquatere Techniken gibt.

## Eine einfache Leaflet-Karte mithilfe des Web-Map-Services erstellen

Eine Alternative zur schon beschriebenen Kachel-Technik ist der Web-Map-Service (WMS). Der WMS ist ein Spezialfall eines Web Services. Dieser Service bietet Ihnen eine Schnittstelle zum Abrufen von Landkartenausschnitten über das Internet.

Ein WMS bietet drei Funktionen, die von einem Benutzer angefragt werden können. Die Funktionen `GetCapabilities` und `GetFeatureInfo` können wir hier vernachlässigen. Diese sind für die Anzeige der Karte nicht relevant. Die Funktion `GetMap` ist die, die wir uns genauer ansehen und die von Leaflet angewendet wird. Bei einem Aufruf von `GetMap` liefert der WMS ein georeferenziertes Rasterbild.

Bei einem georeferenzierten Rasterbild handelt es sich um eine Bilddatei, der raumbezogene Informationen hinzugefügt wurden. Das hört sich zunächst einmal sehr theoretisch an. Praktisch können Sie sich den Vorgang der Georeferenzierung so veranschaulichen: Stellen Sie sich vor, dass das Bild auf einen Bereich auf der Erde gelegt wird. Gleichzeitig wird das Gradnetz der Erde dieses Bereichs mit dem Bild verbunden. Im Ergebnis wird also jedem Pixel des Bildes eine Koordinate – in Relation zum Gradnetz der Erde – zugewiesen. Georeferenzierung kennen Sie vielleicht auch unter dem Begriff Geokodierung, Geotagging oder Verortung.

Innerhalb des GetMap Aufrufs können Sie Optionen auswählen. Zum Beispiel können Sie angeben,

- welches Koordinatensystem zugrundelegt werden soll,
- welchen Kartenausschnitt Sie sehen möchten,
- wie groß der Kartenausschnitt sein soll oder
- welches AusgabefORMAT Sie gerne hätten.

Mit folgendem URL-Abruf erhalten Sie beispielsweise ein speziell zusammengestelltes Bild vom GeoWebservice des Deutschen Wetterdienstes.

```
https://maps.dwd.de/geoserver/dwd/wms  
?service=WMS&version=1.1.1  
&request=GetMap&layers=dwd:Warnungen_Landkreise  
&bbox=6.15,51.76,14.90,55.01  
&width=512  
&height=418  
&srs=EPSG:4326  
&format=image%2Fjpeg  
&CQL_FILTER=EC_I%20IN%20('51','52')
```

Probieren Sie es aus: Der Aufruf der URL im Browser produziert eine Karte mit allen momentan ausgegebenen gültigen Windwarnungen der Kategorie 51 (Windböen) und 52 (Sturmböen) für Norddeutschland. Ausgegeben im JPG-Format. Sie sehen allerdings nur dann ein Bild, wenn tatsächlich Wetterwarnungen vorhanden sind.

Eine Anleitung zur Nutzung des GeoWebservices des Deutschen Wetterdienstes finden Sie unter der Adresse  
[http://www.dwd.de/DE/wetter/warnungen\\_aktuell/objekt\\_einbindung/einbindung\\_karten\\_gewebs\\_\\_blob=publicationFile&v=2](http://www.dwd.de/DE/wetter/warnungen_aktuell/objekt_einbindung/einbindung_karten_gewebs__blob=publicationFile&v=2).

Detaillierte technische Informationen zum Web Mapping Service (WMS) allgemein finden Sie unter der Adresse <http://www.opengeospatial.org/standards/wms> im Internet.

Ausführliche Informationen zu den möglichen Funktionen eines Geoservers finden Sie unter <http://docs.geoserver.org>.

Ich möchte Sie hier an dieser Stelle nicht mit trockenen Dokumentationen von Web Services langweilen. Viel lieber zeige ich Ihnen ein praktisches Beispiel. Im nachfolgenden Programmcodeausschnitt sehen Sie die wesentlichen Zeilen fett hervorgehoben.

```
<!DOCTYPE HTML>  
<html>  
<head>  
<title>Eine OpenStreetMap Karte mit Leaflet</title>  
<link rel="stylesheet" href="../leaflet/leaflet.css" />  
<script src="../leaflet/leaflet.js"></script>  
</head>  
<body>
```

```

<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
var dwd = L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms", {
  layers: 'dwd:bluemarble',
}).addTo(mymap);
</script>
</body>
</html>
<!--index_992.html-->

```

Wenn Sie dieses Beispiel mit dem Laden eines `L.tileLayer` ohne WMS vergleichen, ist eigentlich nur eine Zeile anders.

Anstelle der Zeile

`L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);`

haben wir

```

L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms", {
  layers: 'dwd:bluemarble',
}).addTo(mymap);

```

eingefügt.

Wichtig ist, dass Sie dem Aufruf `L.tileLayer.wms`

- die richtige Adresse zum WMS Service mitgeben und
- die Option `layers` passend setzen.

Für alle anderen Parameter setzt Leaflet, oder der Service selbst, Standardwerte ein – falls Sie nichts Spezielles angeben ...

#### **Hinweis:**

Die Leaflet Klasse `L.tileLayer.wms` können Sie sich hier auf Github ansehen.

Möchten Sie wissen, was vom WMS-Service geliefert wird? Dann öffnen Sie doch die HTML-Datei des vorherigen Beispiels in Ihrem Browser. Mit dem Layer `dwd:bluemarble` können Sie ein Satellitenbild zu Ihrer Karte hinzufügen. Wie das genau aussieht, sehen Sie im nachfolgenden Bild.



## L.tileLayer.wms über L.tileLayer.wms

Das Schöne an WMS-Layern ist, das Sie diese übereinander legen können. Das nachfolgende Beispiel enthält Programmcode, der im Ergebnis gleichzeitig drei WMS-Layer übereinander anzeigt.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

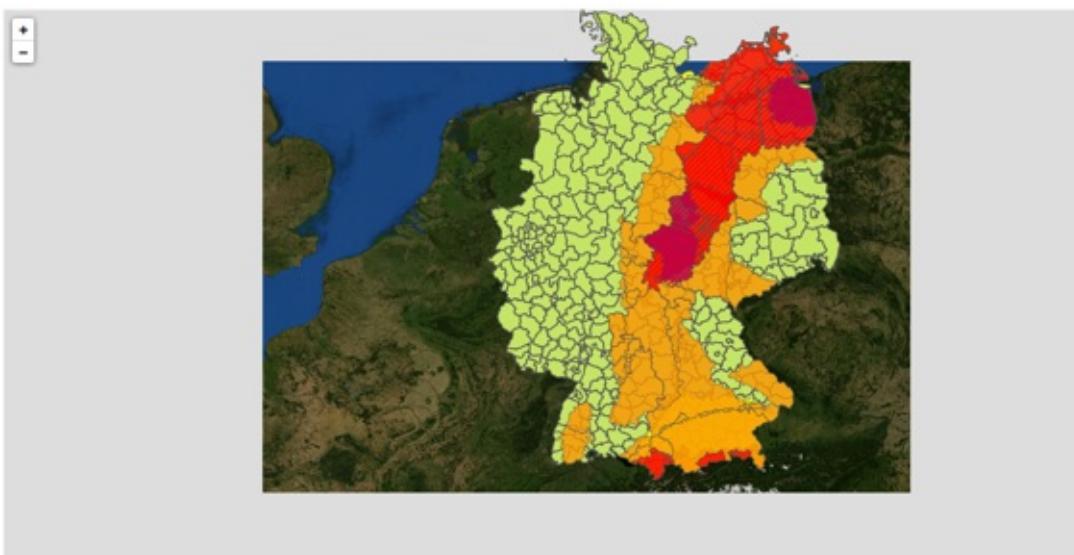
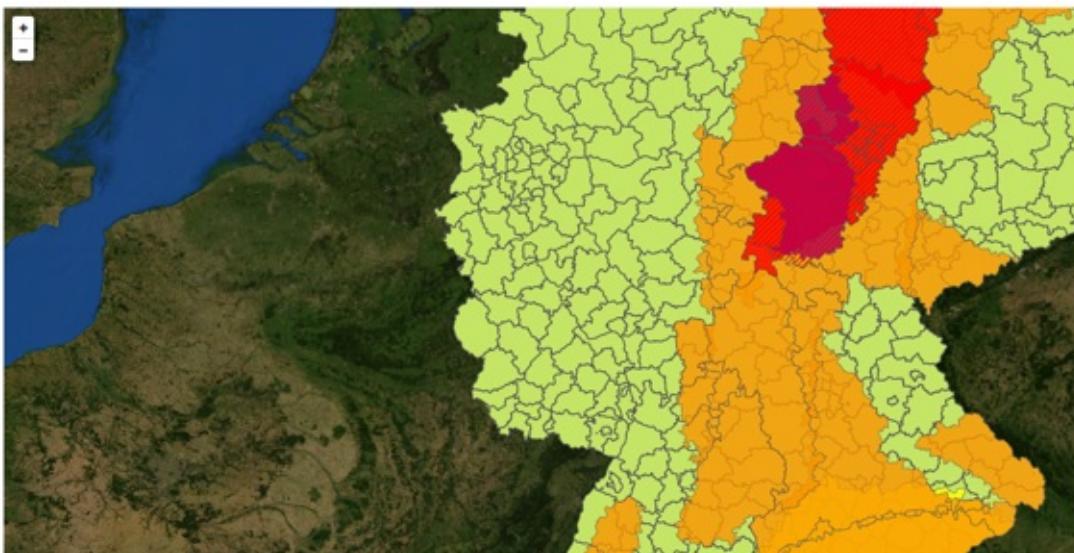
L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
  transparent: true,
  layers: 'dwd:bluemarble',
}).addTo(mymap);

L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
  format: 'image/png',
  transparent: true,
  layers: 'dwd:Warngebiete_Kreise'
}).addTo(mymap);

L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
  format: 'image/png',
  transparent: true, layers: 'dwd:Warnungen_Gemeinden_vereinigt'
}).addTo(mymap);

</script>
</body>
</html>
<!--index_991.html-->
```

Dieses Beispiel ist meiner Meinung nach selbsterklärend. Wichtig ist, dass Sie die Option `transparent` mit `true` übergeben. Andernfalls sehen Sie nur einen – nämlich den obersten – Layer. Bereiche, die nicht mit Daten gefüllt sind, werden weiß gezeichnet. Außerdem müssen Sie die Option `format` mit '`image/png`' belegen. Leaflet lädt ansonsten automatisch das Format '`image/jpeg`' und dieses Format unterstützt keine Transparenz.



## L.tileLayer.wms und L.tileLayer zusammen auf einer Karte

Das nachfolgende Beispiel zeigt Ihnen, wie Sie einen `L.tileLayer` mit einem `L.tileLayer.wms` kombinieren können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
```

```

<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
format: 'image/png',
transparent: true,
layers:'dwd:Warngebiete_Kreise'}).addTo(mymap);

L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
transparent: true,
format: 'image/png',
layers:'dwd:Warnungen_Gemeinden_vereinigt'
}).addTo(mymap);
</script>
</body>
</html>
<!--index_990.html-->
```

Für dieses Beispiel gilt das, was ich im vorherigen Beispiel bezüglich Transparenz und Format geschrieben habe. Zusätzlich müssen Sie darauf achten, dass Sie den `L.tileLayer` nicht über die `L.tileLayer.wms.Layer` Schicht legen. Der `L.tileLayer` ist nicht transparent. Er würde die `L.tileLayer.wms.Layer` Schicht vollständig abdecken.

Die nachfolgende Abbildung zeigt Ihnen die zwei `L.tileLayer.wms` Layer über dem `L.tileLayer` Layer.



### Achtung:

Wenn auf Ihrer Karte der Layer `dwd:Warnungen_Gemeinden_vereinigt` nicht angezeigt wird, kann es daran liegen, dass es zur Zeit keine Warnungen gibt. Dieser Layer enthält nur Daten, wenn aktuell Wetterwarnungen vorliegen. Die grünen Polygone – im Layer `dwd:Warngebiete_Kreise` – die die Landkreise darstellen werden dahingegen immer eingeblendet.

## **In diesem Kapitel haben wir ...**

Sie können nun eine digitale Karte mit Leaflet in ein HTML Dokument einbinden, Sie wissen was geografische Koordinaten sind und wie die Imagedateien für digitale Karten erstellt werden. Darauf aufbauend können wir nun im nächsten Kapitel Ihre digitale Karte mit Geodaten füllen.

# Die Karte mit Daten bestücken

## In diesem Kapitel werden wir ...

Bisher haben wir ausschließlich Layer zur Karte hinzugefügt. Wie das Beispiel mit den Warnhinweisen vom Deutschen Wetterdienst im vorherigen Kapitel zeigt, können diese Layer dynamisch mit Daten bestückt werden.

In diesem Kapitel zeige ich Ihnen nun, wie Sie selbst Layer mit Daten erstellen und als Schicht auf Ihrer Karte anzeigen können. Dabei verwenden wir einfache geometrische Objekte. Konkret sind dies Punkte, Linien und Polygone.

- Punkte/Marker:

Das einfachste Objekt ist ein Punkt. Auf einer Landkarte wird ein Punkt mit Zahlen – den Koordinaten – beschrieben. In Leaflet gibt es neben dem Punkt noch ein komfortableres Element – nämlich den Marker.

Auf einer realen Karte könnte ein Marker zum Beispiel einen interessanten Ort, also einen Point of Interest (POI) darstellen.

- Linien:

Eine Linie besteht aus mindestens zwei Punkten die miteinander verbunden sind. Auf einer Landkarte könnte eine Linie zum Beispiel eine Straße darstellen.

- Polygone:

Ähnlich wie eine Linie besteht ein Polygon aus mehreren miteinander verbundenen Punkten. Zusätzlich ist beim Polygon auch der erste Punkt mit dem letzten Punkt verbunden. Somit stellt ein Polygon eine geschlossene Form dar. Auf einer realen Karte könnte ein Polygon zum Beispiel die Fläche eines Landes oder einer Stadt darstellen.

Wie können Sie nun Punkte, Marker, Linien und Polygone auf Ihre Karte zeichnen? Das ist Thema dieses Kapitels. Außerdem erkläre ich Ihnen wie Sie die Objekte in Layer-Gruppen und Feature-Gruppen gruppieren können und welche Vorteile dies bringt. Nebenbei sehen wir uns an, wie Sie die Karte auch auf kleinen Displays passend anzeigen und wie Sie auf Ereignisse reagieren können.

## Punkte, Marker, Linien, Polygone, Rechtecke und Kreise als Leaflet Objekte

Leaflet bietet Ihnen jede Menge verschiedene Objekte. Beginnen wir mit den einfachsten – den Punkten und Markern.

# Punkte und Marker

Ich gehen noch einmal von unserer Basiskarte aus. Zur besseren Übersicht habe ich Ihnen das HTML-Grundgerüst dazu nachfolgend noch einmal abgedruckt.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
</script>
</body>
</html>
<!--index_989.html-->
```

In der nächsten Abbildung sehen Sie die Karte, die angezeigt werden sollte, wenn Sie die vorhergehende HTML-Datei in einem Browser öffnen.



Diese Karte ist bisher nicht sehr spannend. Landkarten, die Straßen und Orte anzeigen, findet man wie Sand am Meer. Auf Ihrer Website geht es sicher um etwas Besonderes und so möchten Sie sicher auch einen besonderen Ort hervorheben oder ein besonderes Thema beschreiben. Beginnen wir mit einem besonderen Ort – beziehungsweise einem besonderen Punkt.

## Punkte

Die Leaflet Klasse Point stellt einen Punkt mit X- und Y-Koordinaten in *Pixeln* dar.

Sie haben richtig gelesen. Bei einem Punkt arbeitet Leaflet nicht mit *geografischen* Koordinaten. Hier müssen Sie tatsächlich die *Pixel* der Grafik, mit die Karte angezeigt wird, angeben.

Objekte vom Typ `Point` sind in Leaflet nicht zur Anzeige gedacht. Vielmehr wird mit Ihnen gearbeitet. Zum Beispiel gibt es die Methode `panBy()` mit der die Karte um eine gegebene Anzahl von Pixeln verschoben werden kann.

Sie merken schon. Ein Objekt vom Typ `Point` ist nicht das, was wir möchten, wenn wir eine geografische Stelle auf der Karte hervorheben möchten. Um einen Punkt als besonders zu markieren, möchten Sie sicher eher einen Marker mit Informationen an dieser Stelle – also an den *geografischen* Koordinaten – anzeigen. Und für diesen Zweck bietet Leaflet ein spezielles Objekt – das `Marker` Objekt.

Im nachfolgenden Programmcode-Beispiel sehen wir uns zunächst kurz an, was Sie mit einem Punkt anstellen können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var point = L.point(400, 600);
mymap.panBy(point);
</script>
</body>
</html>
<!--index_988.html-->
```

Wenn Sie das vorhergehende HTML-Dokument in Ihrem Browser öffnen, müssen Sie ganz schnell gucken. Nur dann können Sie erkennen, dass die Karte unmittelbar nach dem Öffnen um 400 Pixel nach rechts und 600 Pixel nach unten geschoben wird.

Anstelle von

```
var point = L.point(400, 600);
mymap.panBy(point);
```

hätten Sie auch ganz einfach

```
mymap.panBy([400, 600]);
```

oder

```
mymap.panBy(L.point(400, 600));
```

schreiben können.

An dieser Stelle füge ich kein Bild ein, auf dem das Ergebnis dargestellt ist. Das Verschieben der Karte wäre auf diesem nicht zu erkennen.

## Marker

Die Leaflet Klasse `Marker` wird verwendet, um anklickbare und/oder verschiebbare *Symbole* auf einer Karte anzuzeigen. Die Klasse erweitert die Klasse `Layer`. In der Regel wird dieses *Symbol* mit einem Pop-up Fenster erweitert, auf dem weitere Informationen zu der entsprechenden Stelle auf der Erde enthalten sind.

Machen wir mit einem einfachen Beispiel weiter und fügen einen Marker zu unserer bisher noch langweiligen Karte hinzu.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css"/>
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var myMarker = L.marker([50.27264, 7.26469],
{
    title:"Gering",
    alt:"Ein schönes kleines Dorf auf dem Maifeld in der Eifel",
    draggable:true
}).addTo(mymap);
</script>
</body>
</html>
<!--index_987.html-->
```

Was haben wir genau gemacht? Mit dem Text `var myMarker = L.marker([50.27264, 7.26469])` haben wir ein `Marker` Objekt, das an den Koordinaten [50.27264, 7.26469] angezeigt wird, instanziert. Im Weiteren haben wir die Optionen `title`, `alt` und `draggable` mit Werten belegt. Dabei setzen Sie mit der Option `title` den Tooltip der erscheint, wenn Sie die Maus über den Marker bewegen, mit der Option `alt` setzen Sie den Text für das `alt` Attribut des Marker Bildes und mit der Option `draggable` legen Sie fest, ob der Marker mithilfe der Maus auf der Karte bewegt werden kann – oder nicht.

Das `<img>`-Tag des Markers fügt Leaflet zur Anzeige wie folgt ins HTML-Dokument ein:

```

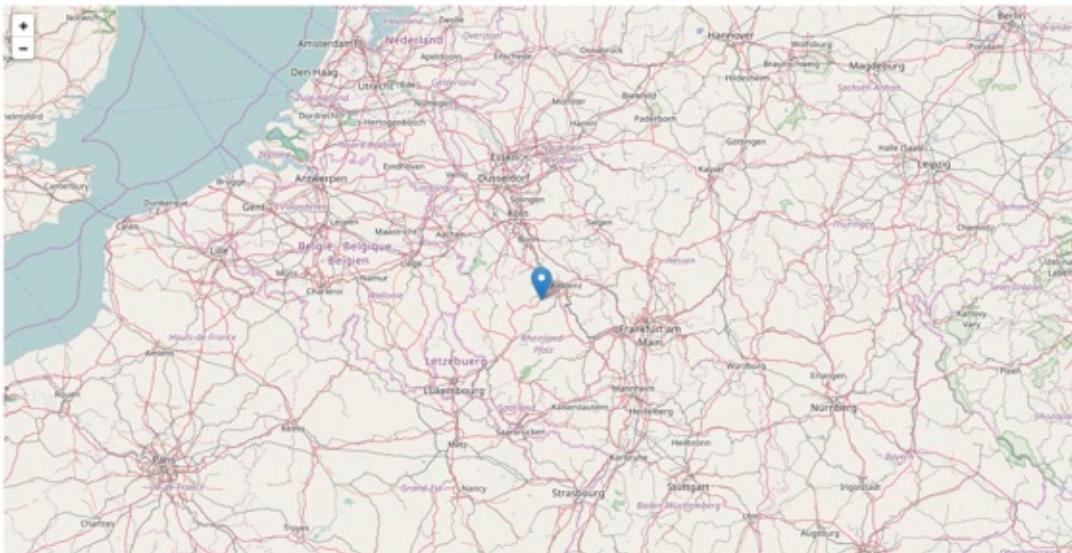
![Ein schönes kleines Dorf auf dem Maifeld in der Eifel](file:///images/marker-icon.png "Gering")

```

Alle Optionen und Methoden, die Ihnen die Leaflet Klasse Marker bietet, können Sie in der Leaflet Dokumentation nachlesen.

Im nächsten Bild sehen Sie den Marker in die Karte integriert. Als Bild wurde ein Standardbild verwendet, da die Option icon nicht gesetzt wurde. Wie Sie ein benutzerdefiniertes Icon verwenden können, sehen wir uns in Kapitel *Custom Markers* genauer an.

In der Karte des vorhergehenden Beispiels können Sie den Marker im Browser mit der Maus bewegen, da die Option draggable auf true gesetzt wurde. Probieren Sie es aus!



Sie müssen für einen Marker keine Variable instanziieren. Sie können den Marker auch ganz einfach so zur Karte hinzufügen:

```

L.marker(
[50.27264, 7.26469],
{ title:"Gering", alt:"Ein schönes kleines Dorf auf dem Maifeld in der Eifel",
draggable:true }
).addTo(mymap);

```

Beachten Sie dabei aber folgendes: Falls Sie den Marker später einmal modifizieren möchten, benötigen Sie einen Namen. Andernfalls können Sie den Marker nicht identifizieren und somit auch nicht zum Ändern auswählen.

# Objekte, die aus mehr als einem Punkte bestehen

## Linien

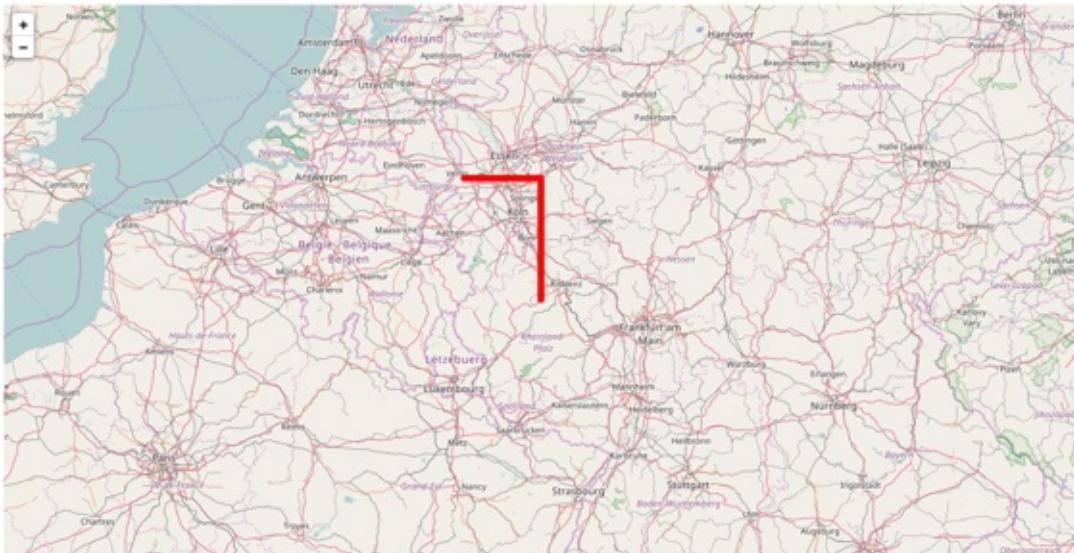
Linien können Sie in Leaflet mit der Klasse Polyline erstellen. Die Klasse Polyline erweitert die abstrakte Klasse Path. Diese Klasse ermöglicht es Ihnen eine einfache Linie oder mehrere aneinandergereihte Liniensegmente zu erstellen.

Im nachfolgenden Programmcodebeispiel habe ich den Text, der für die Erstellung des Polyline Objektes verantwortlich ist, fett formatiert.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var polyline = L.polyline([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]
]),
{color: 'red', weight:8})
.addTo(mymap);
</script>
</body>
</html>
<!--index_986.html-->
```

Im Beispiel wird ein Polyline Objekt instanziert, das aus drei Punkten besteht. Außerdem werden die Optionen color und weight verwendet. Diese Optionen erbt die Klasse Polyline von der Klasse Path. Konkret legen wir im Beispiel mit color = 'red' die Farbe der Linie fest – diese soll rot sein. Mit weight:8 bestimmen wir, dass die Linie 8 Pixel dick sein soll.

In der nächsten Abbildung sehen Sie genau diese Linie.



## Polygone

Ein Polygone ist eine geschlossene Linie – also eine Linie die einen *Innenbereich* von einem *Außenbereich* abtrennt. Mit der Klasse Polygone können Sie, wie der Name schon sagt, Polygone Objekte auf Ihre Karte zeichnen. Die Klasse Polygone erweiterte die Klasse Polyline und somit auch die Klasse Path. Alle Methoden und Optionen dieser Klassen können Sie also auch mit einem Polygone Objekt nutzen.

Die Punkte, die Sie beim Erstellen eines Polygons übergeben, sollten als letzten Punkt keinen Punkt enthalten, der dem ersten Punkt entspricht. Es ist besser, solche Punkte herauszufiltern. Warum sollen der erste und der letzte Punkt nicht identisch sein? Leaflet schließt Ihr Polygon automatisch. Wenn der erste Punkt mit dem letzten Punkt übereinstimmt, kann dies zu Problemen führen.

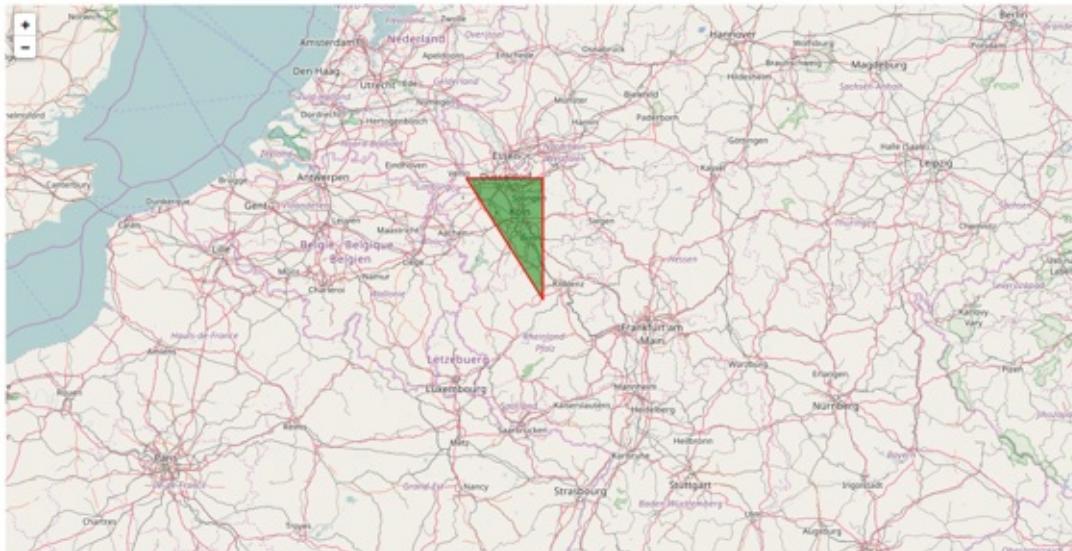
Wie Sie ein Polygon in Leaflet erstellen, zeigt Ihnen der nachfolgende Programmcode und das konkrete Polygon sehen im darauf folgenden Bild. Alle Methoden und Optionen, die Sie auf ein Polygon Objekt anwenden könne, finden Sie in der Leaflet Dokumentation.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var polygon = L.polygon(
[
[50.27264, 7.26469],
[51.27264, 7.26469],
```

```

[51.27264, 6.26469]
],
{color: 'red', weight:2, fillColor:'green', fillOpacity:0.5}
).addTo(mymap);
</script>
</body>
</html>
<!--index_985.html-->

```



## Rechtecke und Kreise

Um einen Kreis oder ein Rechteck zu erstellen benötigt man eigentlich keine eigene Klasse. Man könnten diese Formen mit der Klasse `Polygon` erzeugen. Beim Kreis wäre dies allerdings sehr mühselig und auch für ein Rechteck gibt es einfacherer Verfahrensweisen. Leaflet bietet deshalb für diese beiden Formen spezielle Klassen an.

Das Zeichnen von Rechtecken und Kreisen ist kein Hexenwerk. Der Vollständigkeit halber finden Sie in den nächsten beiden Kapiteln jeweils ein Beispiel für beide Formen.

### Rechtecke

Der nachfolgende Programmcode zeigt Ihnen an einem Beispiel, wie Sie ein Rechteck – also ein Objekt vom Typ `Rectangle` – in eine Leaflet Karte einfügen.

```

<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

```

```

var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var myRectangle = L.rectangle(
[
[50.27264, 7.26469],
[51.27264, 6.26469]],
{color: "yellow", weight: 8, fillColor:"purple"}
).addTo(mymap);
</script>
</body>
</html>
<!--index_984.html-->
```

Als Parameter geben Sie für das Rechteck immer die diagonal versetzten beiden Ecken an. Dabei ist es egal, in welcher Reihenfolge Sie diese angeben.

## Das Rechteck

```

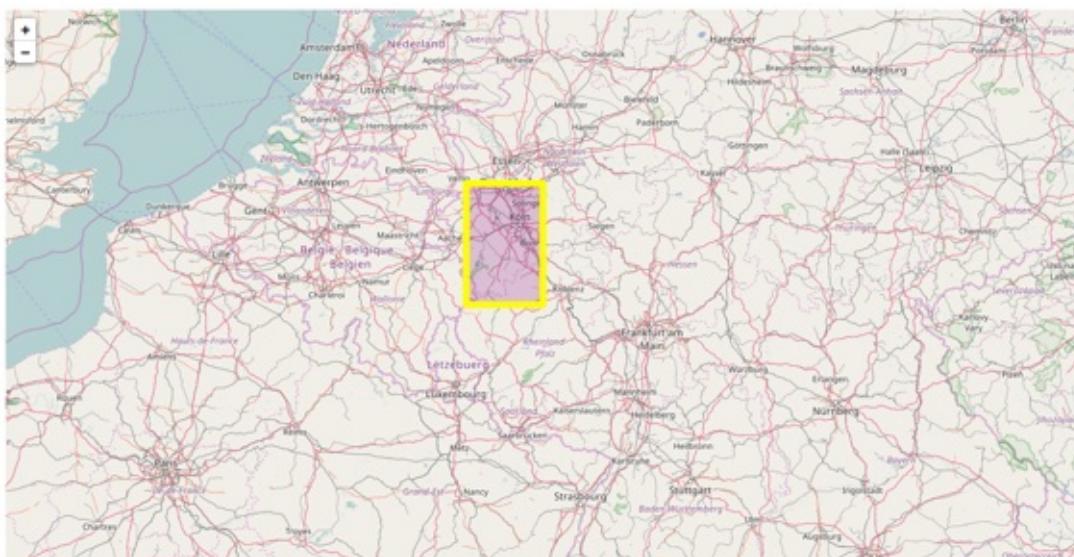
var myRectangle = L.rectangle([
[50.27264, 7.26469],
[51.27264, 6.26469]
])
```

sieht auf der Karte genauso aus, wie das Rechteck

```

var myRectangle = L.rectangle([
[51.27264, 6.26469],
[50.27264, 7.26469]
])
```

Und wie das Rechteck des Beispiels aussieht, zeigt Ihnen die nachfolgende Abbildung.

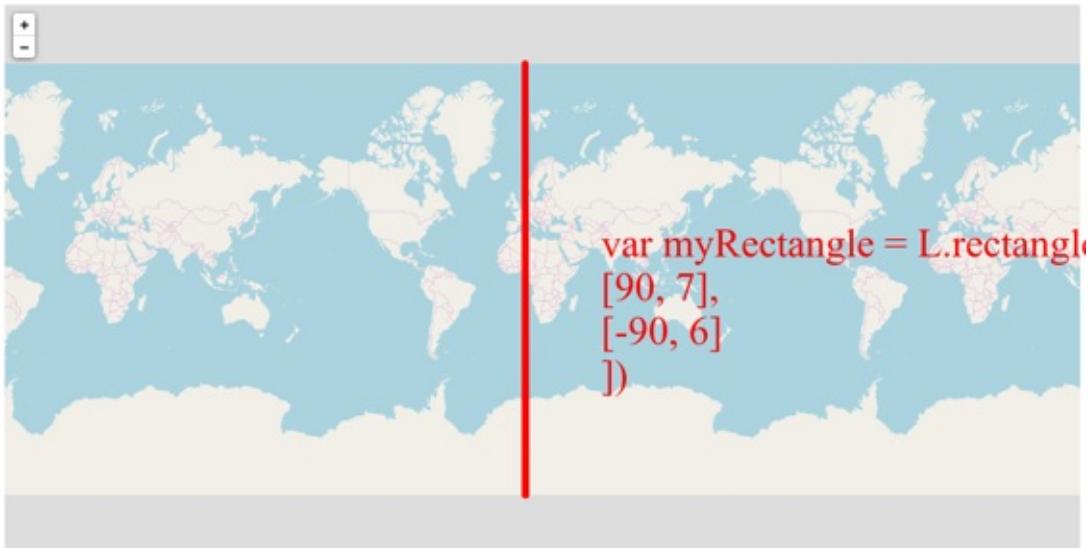


In diesem Zusammenhang ist vielleicht ganz interessant wie Leaflet das Koordinatensystem der Erde verarbeitet. Dieses Koordinatensystem ist ja kein gewöhnliches Koordinatensystem, sondern ein sphärisches. Unter einer Sphäre

versteht man in der Mathematik ganz vereinfacht ausgedrückt die Oberfläche einer Kugel. Und auf einer Kugel führen, im Gegensatz zu einer Ebene, immer zwei Linien auf direktem Wege zu einem anderen Punkt. Die nachfolgenden Abbildungen zeigen Ihnen, dass Leaflet dieses Problem vereinfacht. Es behandelt das sphärische Koordinatensystem als normales Koordinatensystem. Dies hatten wir uns im Exkurs *Das Koordinatensystem der Erde* schon einmal angesehen.

Um von der Koordinate [50, -180] zur Koordinate [51, 180] zu gelangen, muss man mit Leaflet einmal um die ganze Erde laufen. Und analog wird auch ein Rechteck mit diesen Eckpunkten um die ganze Erde gezeichnet. Und das, obwohl die Ecken hier ganz nah beieinander liegen.





## Kreise

Anhand des nachfolgenden Programmcode können Sie erkennen, wie ein Kreis – also ein Objekt vom Typ `Circle` – in eine Leaflet Karte eingefügt wird.

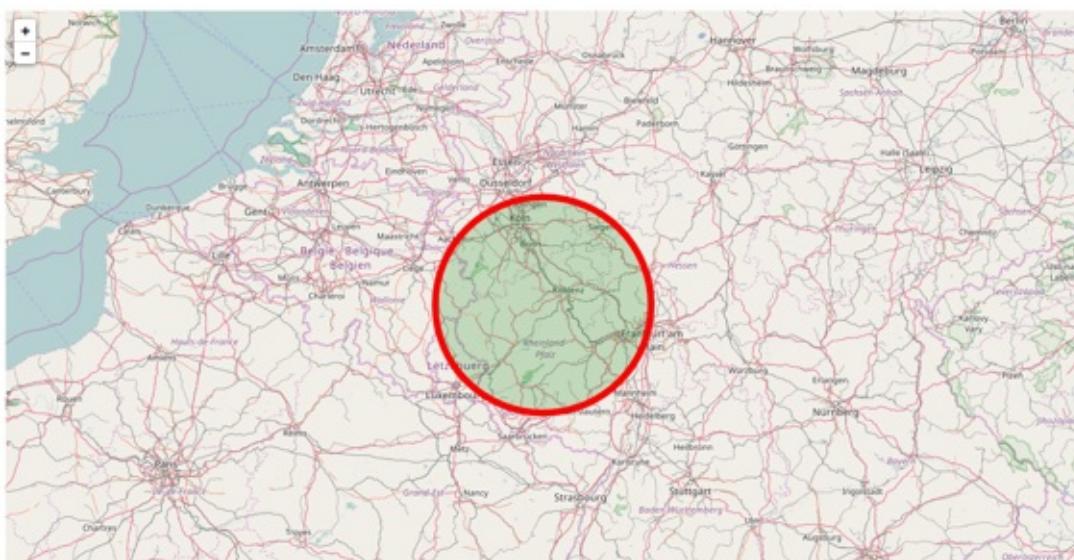
```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
L.circle(
[50.27264, 7.26469], 100000,
{color: "red", weight: 8, fillColor:"green"})
```

```

).addTo(mymap);
</script>
</body>
</html>
<!--index_983.html-->

```

Ein Kreis wird anhand seines Mittelpunktes und seines Radius definiert. Der Radius muss dabei in Metern angegeben werden. In den Beispielen habe ich den Rand mit 8 Pixel festgelegt. Diese Breite passt für die Zoom-Stufe 7. Probieren Sie doch einmal aus, wie der Rand aussieht, wenn Sie die Karte vergrößern. Sie werden feststellen, dass man irgendwann nur noch den Rand sieht und es wichtig sein könnte, dessen Größe immer relativ zur Zoom-Stufe zu setzen.



## Mehrere Poly- Objekte auf einer Ebene

In den vorhergehenden Beispielen haben wir jedes Element auf einem separaten Layer – also einer separaten Ebenen – abgebildet. Spätestens, wenn Sie selbst unterschiedliche Geodaten auf einer Karte darstellen möchten werden Sie sich wünschen, Objekte mit gleichen Eigenschaften auf einem Layer gruppieren zu können. Denn nur so können sie alle Elemente zusammen ansprechen.

Stellen Sie sich vor, Sie möchten auf Ihrer Website Touren für Aktivurlauber beschreiben. Diese Touren sind unterteilt in Wanderer, Bergsteiger, Gipfelstürmer und Freikletterer. Auf der Karte kann ein Kunde Angebote für eine bestimmte Region heraus filtern. Hierzu muss er nur die Karte passend positionieren. Wenn Sie pro Touren-Typen einen Layer anlegen und die Touren entsprechend ihres Typs auf diesen Layern ablegen, können Sie es dem Kunden zusätzlich ermöglichen, nur die für ihn relevanten Touren einzublenden. Warum erkläre ich Ihnen dies an dieser Stelle? Alle Polylines Objekte und alle Polygone Objekte die zusammen definiert wurden, liegen automatisch zusammen auf einem eigenen separaten Layer.

Das Setzen von Klammern beim Arbeiten mit Polygon Objekten und Polyline

Objekten, die mehrere Objekte gleichzeitig auf einem Layer erstellen, kann sehr herausfordernd sein.

- Sie müssen zum einen alle Objekte – Polygon Objekte oder Polyline Objekte – zusammen mit einer äußereren Klammer versehen.

```
[  
  [[50.17264, -7.26469], [49.27264, -6.26469]],  
  [[50.37264, -7.26469], [51.27264, -8.26469]]  
]
```

- Außerdem müssen Sie alle Punkte jedes einzelnen Objekts – Polyline oder Polygon – mit einer Klammer umgeben.

```
[  
  [[50.17264, -7.26469], [49.27264, -6.26469]],  
  [[50.37264, -7.26469], [51.27264, -8.26469]]  
]
```

- Und zuletzt werden auch die Koordinaten selbst noch eingeklammert.

```
[  
  [[50.17264, -7.26469], [49.27264, -6.26469]],  
  [[50.37264, -7.26469], [51.27264, -8.26469]]  
]
```

## Mehrere Polyline Objekte auf einer Ebene

In diesem Kapitel gibt es nun ein praktisches Beispiel. Der nachfolgende Programmcodeausschnitt zeigt Ihnen, wie Sie mit dem instanziieren eines Objekts mehrerer Polyline Objekte auf eine Ebene zeichnen können. Und das die Polyline Objekte alle auf einer Ebene liegen, beweise ich Ihnen gleichzeitig.

```
<!DOCTYPE HTML>  
<html>  
<head>  
<title>Eine OpenStreetMap Karte mit Leaflet</title>  
<link rel="stylesheet" href="../leaflet/leaflet.css" />  
<script src="../leaflet/leaflet.js"></script>  
</head>  
<body>  
<div style="height: 700px;" id="mapid"></div>  
<script>  
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);  
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);  
var multipolyline = L.polyline(  
[  
  [  
    [50.17264, -7.26469],  
    [49.27264, -7.26469],  
    [49.27264, -6.26469]  
  ],  
  [  
    [50.37264, -7.26469],  
    [51.27264, -7.26469],  
    [51.27264, -8.26469]  
  ],  
  {color: 'black'}).addTo(mymap);
```

```

    mymap.fitBounds(multipolyline.getBounds());
</script>
</body>
</html>
<!--index_981.html-->
```

Was zeigt Ihnen dieses Beispiel? Zunächst einmal können Sie erkennen, dass das Erstellen mehrerer Polyline Objekte zusammen möglich ist. Und das es auch einfacher ist erkennen Sie daran, dass Sie Optionen nur einmal zuweisen müssen. Beide Linien werden gleichzeitig schwarz gefärbt.

Am meisten Vorteile bringt aber die Tatsache, dass beide Linien eine Ebene darstellen. Dies bedeutet, dass die Methode `fitBounds()` die beiden Linien als ein Objekt sieht und die Abgrenzungen dieses einen Objektes berechnet. Die Methode `fitBounds()` verschiebt eine Karte an die Stelle, an der die übergebenen Koordinaten sich befinden, und setzt die Zoom-Stufe so, dass die Karte für die übergebenen Abgrenzungen ideal angezeigt wird. Würden die beiden Linien nicht auf einer Ebenen liegen, wäre es komplizierter beide Linien gleichzeitig vollständig mit optimaler Zoom-Stufe anzuzeigen.

■ Noch einmal zur Erinnerung: Ein Objekt vom Typ `Polygon` und auch ein `Polyline` Objekt ist eine Ebene – die beiden Klassen erweitern die Leaflet Klasse Layer.



## Mehrere Polygon Objekte auf einer Ebene

In diesem Kapitel zeige ich Ihnen ein Beispiel, bei dem mehrere Polygone zusammen erstellt werden.

```

<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
```

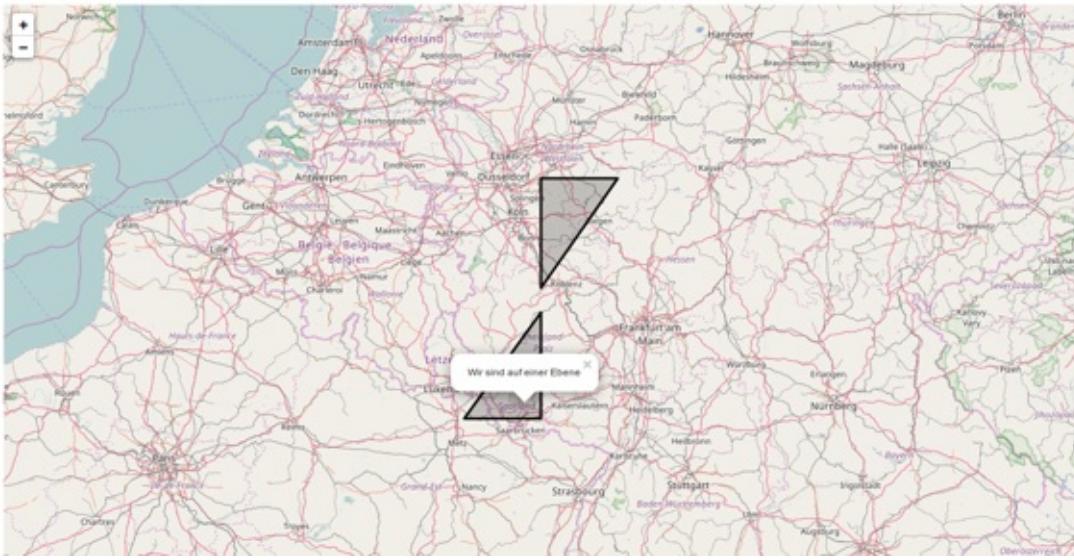
```

<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var multipolygon = L.polygon(
[
[
[50.17264, 7.26469],
[49.27264, 7.26469],
[49.27264, 6.26469]],
[
[50.37264, 7.26469],
[51.27264, 7.26469],
[51.27264, 8.26469]
]
],
{color: 'black'}).addTo(mymap).bindPopup("Wir sind auf einer Ebene");
</script>
</body>
</html>
<!--index_982.html-->
```

In diesem Beispiel sehen Sie, wie Sie mit der Methode `bindPopup()` ein Pop-up Fenster zu einem Element hinzufügen können. Später im Buch werden Pop-up Fenster noch einmal Thema sein. An dieser Stelle habe ich die Methode gewählt, weil ich der Meinung bin, das diese hier gut passt.

Es gibt viele unzusammenhängende Flächen auf der Erde, für die die gleichen Informationen zutreffend sind. Zum Beispiel haben viele Länder Kolonien. Wenn nun alle zu einem Land gehörenden Gebiete als Polygon auf einem Layer zusammengefasst sind, dann ist es ein Leichtes, allen Gebieten eines Landes das gleiche Pop-up Fenster zuzuweisen. Und wenn Sie nun die ganze Welt auf Ihrer Karte abdecken möchten, dann können Sie sich den Vorteil dieser Layer-Technik sicherlich gut vorstellen.

Auf der nächsten Abbildung sehen sie keine Ländergrenzen. Der Einfachheit halber habe ich mich auf zwei Dreiecke beschränkt.

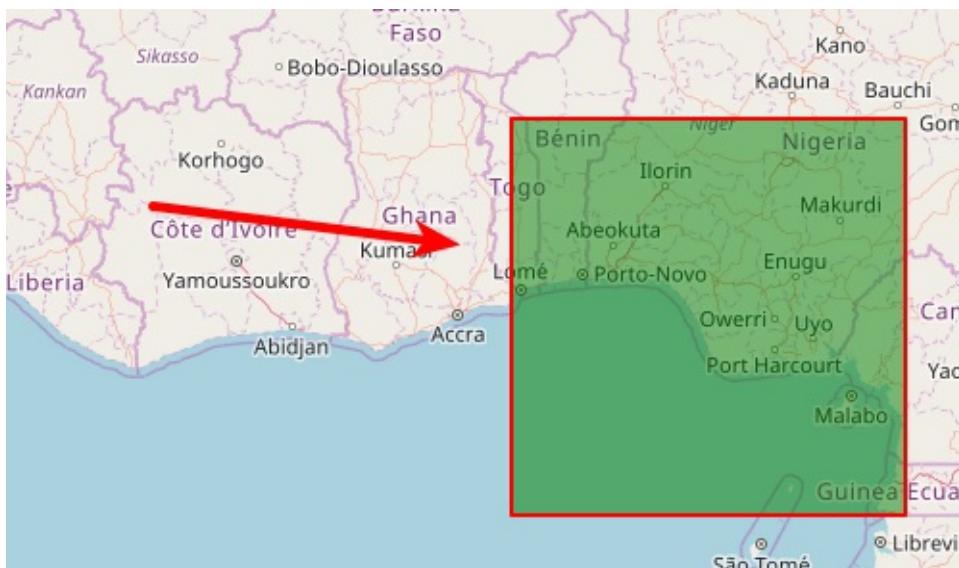


## Mehrere sich überschneidende Polygon Objekte auf einer Ebene

Die Besonderheit bei einem Polygon ist, dass es einen Außen- und einen Innenbereich gibt. Das lesen Sie hier im Buch des Öfteren. Wenn Sie mehrere Polygone auf einem Layer zusammen erstellen, dann beeinflussen diese sich gegenseitig, wenn Flächen übereinander liegen. In der nachfolgenden Tabelle habe ich verschiedene Konstellationen mit Bild und Text aufgenommen. Anhand von Beispielen ist das Zusammenspiel der verschiedenen Bereiche meiner Meinung nach am leichtesten nachzuvollziehen.

Beginnen wir mit einem einfachen Polygone:

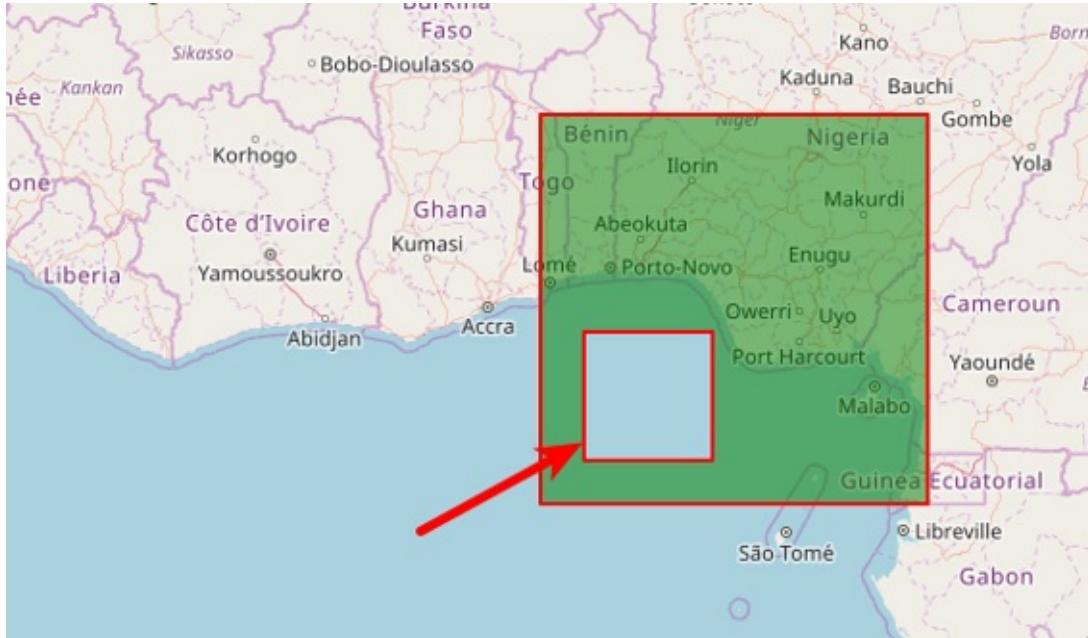
```
L.polygon([
  [ [1, 1], [1, 10], [10, 10], [10, 1] ]
])
```



Wenn Sie innerhalb dieses Polygons ein weiteres Polygon zeichnen, dann wird der

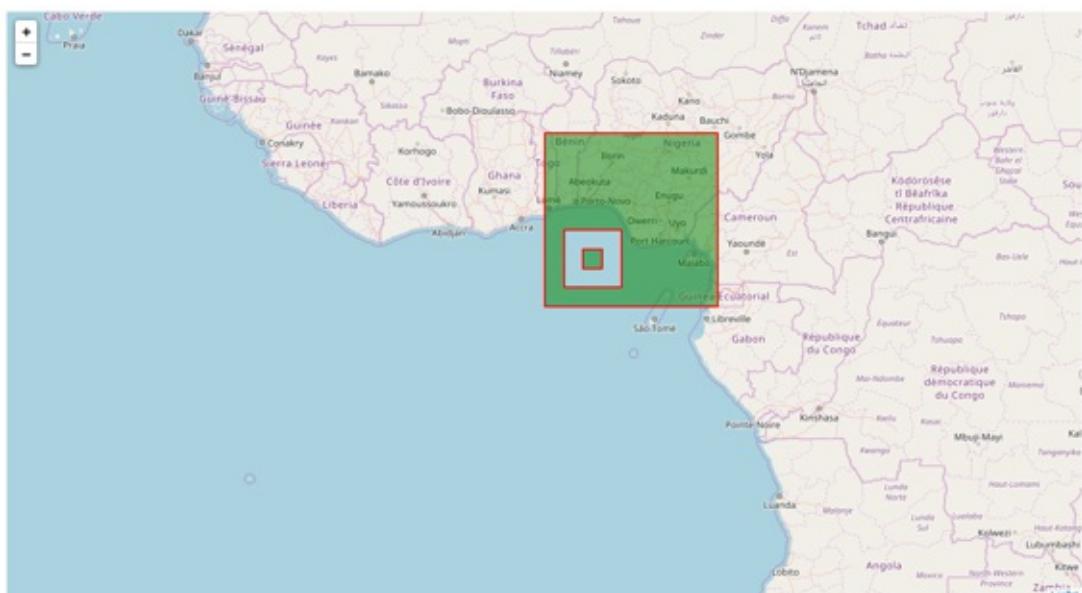
Innenbereich des ersten Polygon um die Fläche des zweiten Polygon verringert. Das zweite Polygon wird praktisch aus dem ersten Polygone ausgeschnitten.

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10], [10, 1] ],
  [ [2, 2], [2, 5], [5, 5], [5, 2] ]
])
```



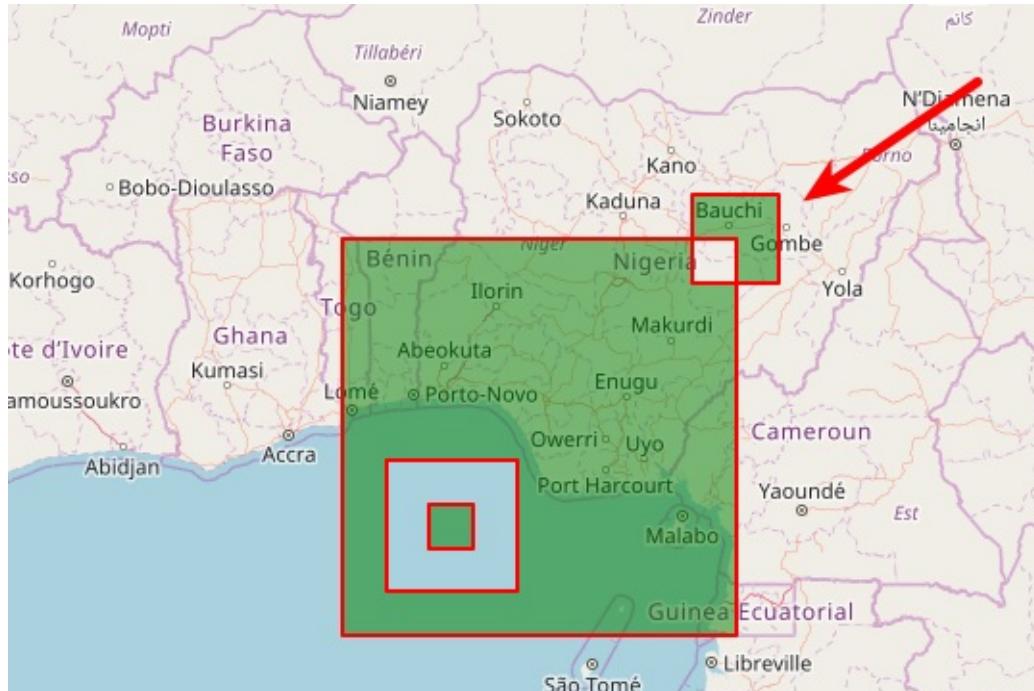
In dieses zweite Polygon können Sie nun wiederum ein Polygon zeichnen. Nun wird diese Fläche wieder voll als Innenbereich gezeichnet.

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10], [10, 1] ],
  [ [2, 2], [2, 5], [5, 5], [5, 2] ],
  [ [3, 3], [3, 4], [4, 4], [4, 3] ]
])
```



Zeichnen Sie nun ein weiteres Polygon, dass Innenbereich und Außenbereich gleichzeitig abdeckt, dann wird der abgedeckte Innenbereich zum Außenbereich und umgekehrt.

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10], [10, 1] ],
  [ [2, 2], [2, 5], [5, 5], [5, 2] ],
  [ [3, 3], [3, 4], [4, 4], [4, 3] ],
  [ [9, 9], [11, 9], [11, 11], [9, 11] ]
])
```



## Daten mit Layern gruppieren

Im vorherigen Kapitel haben Sie gesehen, dass Sie mehrere Polygone Objekte oder mehrere Polyline Objekte zusammen auf einem Layer positionieren können. Es wird sicher einmal vorkommen, dass Sie Elemente unterschiedlicher Typen auf einem Layer zusammen gruppieren möchten.

## Layergruppen

Die Leaflet Klasse `LayerGroup` wird verwendet, um mehrere Layer oder Ebenen zu gruppieren. So können Sie diese Ebenen wie eine Ebene behandeln. Wenn Sie ein Objekt vom Typ `LayerGroup` zur Karte hinzufügen, werden alle zur Gruppe gehörenden Layer zum Karten Objekt hinzugefügt.

Der nachfolgende Programmcodeausschnitt zeigt Ihnen, wie Sie ein Objekt vom Typ `LayerGroup` erstellen und zu Ihrem Karten Objekt hinzufügen können.

```

<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

var marker1 = L.marker([50.27264, 7.26469],
{
  title:"Marker1",
  alt:"Ich bin ein Marker"
}
).bindPopup('Marker1');

var polyline = L.polyline([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]],
{color: 'red', weight:8});

var myLayerGroup = L.layerGroup([marker1, polyline]).addTo(mymap);

</script>
</body>
</html>
<!--index_980.html-->

```

Im vorherigen Beispiel habe ich ein Marker Objekt und ein Objekt vom Typ Polyline erstellt und beide zusammen auf dem Layer myLayerGroup gruppiert.

Sie können auch später noch Objekt zum LayerGroup Objekt hinzufügen. Zum Beispiel so:

```

var marker2 = L.marker([51.27264, 6.26469],
{
  title:"Marker2",
  alt:"Ich bin ein anderer Marker"
}
).bindPopup('Marker2');
myLayerGroup.addLayer(marker2);

```

Entfernen können sie das LayerGroup Objekt – und somit alle zu ihr gehörenden Objekt auf einen Schlag – mit der Methode removeLayer(). Im vorhergehenden vollständigen Beispiel konkret mit der Zeile  
`myLayerGroup.removeLayer(polyline);`

## Featuregruppen

Die Klasse FeatureGroup erweitert die Klasse LayerGroup. Während die Klasse LayerGroup eher Methoden zum Gruppieren von Layern bereit stellt, geht es in der FeatureGroup hauptsächlich um das gemeinsame Verarbeiten von Ereignissen

und das Hinzufügen von Stylen.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

var marker1 = L.marker([50.27264, 7.26469],
{
title:"Marker1",
alt:"Ich bin ein Marker"
});
;

var polyline = L.polyline([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]],
);

var myfeatureGroup=L.featureGroup(
[marker1, polyline]
).addTo(mymap);

myfeatureGroup.bindPopup('Wir haben alle das gleiche Popup!');

</script>
</body>
</html>
<!--index_979.html-->
```

Analog zum Beispiel im Kapitel Layergruppen sehen Sie im vorhergehenden Programmcodeausschnitt, wie zwei Elemente erstellt und einem Objekt vom Typ FeatureGroup zugeordnet werden. Die FeatureGroup, wird dann zum Karten Objekt hinzugefügt und ein Aufruf der Methode bindPopup() beim FeatureGroup Objekt fügt das Pop-up zu allen Objekten des FeatureGroup Objektes hinzu.

Wenn Sie alle Elemente des FeatureGroup Objektes rot färben möchten, reicht die Zeile

```
myfeatureGroup.setStyle({color:'red'})
```

aus.

Der Eintrag

```
myfeatureGroup.on('click', function()
{
alert('Ein Gruppenmitglied wurde angeklickt!');
```

```
}
```

würde bewirken, dass sich immer, wenn ein Element des FeatureGroup Objektes angeklickt wird, ein Hinweisfenster öffnet.

Wenn Sie das FeatureGroup Objekt mit Inhalt mit einem Mausklick entfernen möchten, können Sie folgenden Text zu Ihrem Skript hinzufügen:

```
myfeatureGroup.on('click', function()
{
myfeatureGroup.removeLayer(polyline);
})
```

## Popups

In diesem Kapitel geht es um Leaflet Objekte vom Typ Popup. Mithilfe dieses Objekts können Sie Pop-up Fenster an bestimmten Stellen auf Ihrem Karten Objekt öffnen. Wenn Sie die Methode openPopup() einsetzen, um das Pop-up Fenster zu öffnen, stellt Leaflet sicher, dass gleichzeitig nur ein Pop-up geöffnet ist. Ich empfehle Ihnen diese Methode einzusetzen, weil ich der Meinung bin, dass das gleichzeitige Öffnen von mehreren Pop-up Fenstern nicht benutzerfreundlich ist. Nichtsdestotrotz kann es aber sein, dass Sie gerne mehrere Pop-up Fenster gleichzeitig anzeigen möchten. In diesem Fall müssen Sie das Pop-up Fenster mit der Methode addLayer() auf einem eigenen Layer einbinden.

So nun aber genug der Theorie. Sehen Sie sich das alles anhand des nachfolgenden Programmcodes – am besten auf praktische Weise – an.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {closePopupOnClick: false})
.setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var marker1 = L.marker([50.27264, 7.26469],
{
title:"Marker1",
alt:"Ich bin Marker 1"
}).addTo(mymap);
```

```

var marker2 = L.marker([51.27264, 6.26469],
{
title:"Marker2",
alt:"Ich bin Marker 2"
}).addTo(mymap);

marker1.bindPopup("<h1>Gering</h1>
<p>Ein kleines <a href='https://de.wikipedia.org/wiki/Dorf'>Dorf</a></p>
<ul><li>auf dem Maifeld</li><li>in der Eifel</li><li>an der Elz</li></ul>");
marker2.bindPopup("<h1>Boisheim</h1>
<p>Ein kleines Dorf</p>
<ul><li>irgendwo</li><li>nordwestlich</li><li>von Gering</li></ul>");

```

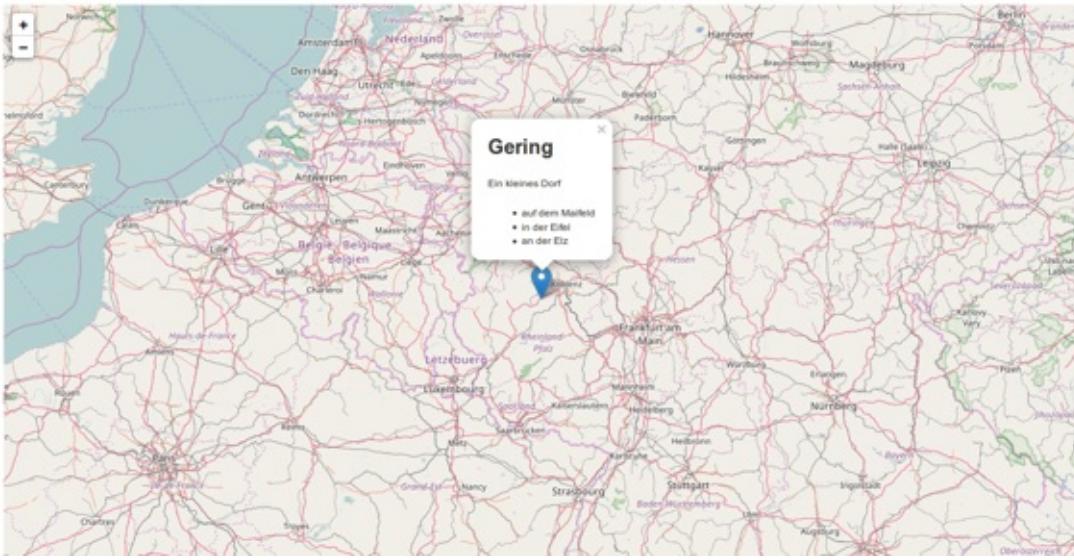
</script>  
</body>  
</html>  
<!--index\_978.html-->

Was zeigt Ihnen dieses Beispiel genau? Sie sehen, wie Sie zwei Marker erstellen und diesen mit der Methode `bindPopup()` ein Pop-up Fenster hinzufügen könne. Wenn Sie den ersten Marker anklicken, dann öffnet sich das Pop-up Fenster dieses Markers. Wenn Sie danach den zweiten Marker anklicken, schließt sich das Pop-up Fenster des ersten Markers und das Pop-up Fenster des zweiten Markers wird aktiv.

Vielleicht ist Ihnen aufgefallen, dass wir beim Karten Objekt die Option `closePopupOnClick` auf `false` gesetzt haben. Diese Option ist standardmäßig mit `true` belegt, was bedeutet, dass auch beim Klick auf eine beliebige Position auf der Karte ein eventuell offenes Pop-up Fenster von Leaflet automatisch geschlossen wird.

Möchten Sie, dass das Pop-up Fenster zum Marker mit dem Namen `marker1` schon beim Öffnen der Karte angezeigt wird? Dann schreiben Sie einfach den Text `marker1.openPopup();` in Ihr Skript.

Wie die Karte zum vorhergehenden Programmcodeausschnitt im Browser aussieht, sehen Sie in der nachfolgenden Abbildung.



Ich hatte Ihnen zu Beginn dieses Kapitels schon erklärt, dass Leaflet standardmäßig nur ein Pop-up Fenster gleichzeitig offen anzeigt. Dies ist so, wenn Sie das Pop-up Fenster mithilfe der Methode `openOn()` zu Karte hinzufügen:

```
var popup1 = L.popup({keepInView:true})
.setLatLng([49.27264, 6.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein Popup.</p>')
.openOn(mymap);
```

Falls Sie mehrere Pop-up Fenster gleichzeitig anbieten möchte, dann können Sie dies mithilfe der Methode `addTo()` bewerkstelligen.

```
var popup1 = L.popup()
.setLatLng([49.27264, 6.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein Popup.</p>')
.addTo(mymap);
```

Die Eigenschaft `keepInView: true` bewirkt hier übrigens, dass die Karte nur so weit verschoben werden kann, wie der Marker sichtbar ist.

## Mobil

Ein großer Vorteil von JavaScript ist es, dass Landkarten in jedem aktuellen Standardbrowser ohne notwendige externe Applikationen oder Plugins angezeigt werden können.

Genaue Informationen zu den unterstützten Browsern finden Sie auf der Website von Leaflet im Bereich *Features*.

Jede Website, die eine Leaflet Karte anzeigt, kann vom Entwickler auf die gleiche Art und Weise programmiert werden. Der Entwickler muss im Normalfall nichts Spezielles für ein Gerät beachtet. Ein paar erwähnenswerte Punkte gibt es aber

trotzdem und das, was meiner Meinung nach erwähnenswert ist, finden Sie in diesem Kapitel.

Das Spannendste bei der Arbeit mit mobilen Anwendungen ist sicherlich die Funktion `locate()`. Diese Funktion versucht, den Benutzer mit der W3C Geolocation API zu lokalisieren. Die W3C Geolocation API ist eine einheitliche Webbrowserschnittstelle zum Ermitteln des geografischen Standorts des zugehörigen Endgeräts. Aber beginnen wir von vorne mit dem Bereich HTML und CSS.

## HTML und CSS

Sie möchten Ihre Leaflet Karte auch für mobile Geräte optimal konfigurieren? Dann sollten Sie als Erstes sicherstellen, dass die Karte passend angezeigt wird. Sie möchten sicher nicht, dass jemand der die Karte über ein Gerät mit einem kleinen Display aufruft, nur einen Teil der Karte sieht und er diese erst verschieben muss, um auch die Randbereiche zu erkennen. Oder, dass die Karte so klein dargestellt wird, dass der Inhalt auf einem kleinen Display nicht lesbar ist. Ideal ist es, wenn die vollständige Karte im Display lesbar angezeigt wird. In diesem Punkt stimmen Sie mir sicher zu. Das nächste Beispiel zeigt Ihnen, wie Sie die vollständige Karte im Display lesbar anzeigen können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0
<style>body {padding: 0; margin: 0;}html, body, #mapid {height: 100vh; width: 100vw;}</style>
</head>
<body>
<div id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick: false}).setView([50.27264, 7.26469], 18)
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

mymap.locate({setView: true, maxZoom: 18});

mymap.on('locationfound', onLocationFound);
mymap.on('locationerror', onLocationError);

function onLocationFound(e) {
var radius = e.accuracy / 2;
L.marker(e.latlng).addTo(mymap).bindPopup("Sie sind etwa" + radius + " Meter von diesem
L.circle(e.latlng, radius).addTo(mymap);
}

function onLocationError(e) {
alert(e.message);
}

</script>
</body>
</html>
<!--index_977.html-->
```

Wissen Sie, dass Sie zum Debuggen die Methode `console.log()` verwenden können?

Der Eintrag

```
<script>
...
console.log(e);
...
</script>
```

gibt an einer Stelle, an der das Objekt `e` instanziert ist, den Inhalt dieses Objektes in der Konsole aus.

Die Konsole können Sie im Browser Mozilla Firefox über das Menü Extras | Web Entwickler | Web Konsole öffnen.

Im Browser Google Chrome finden Sie die Konsole über das Menü Tools | Javascript-Konsole.

Dieses Beispiel enthält zusätzlich zu den vorhergehenden Beispielen ein `<style>`-Element. In diesem `<style>`-Element sind die Abstände, also `padding` und `margin`, auf `0` gesetzt. So passt die Karte sich genau an das Display an und es wird kein Platz mit Rändern verschwendet. Außerdem haben wir die Höhe mit `height: 100vh` auf 100 % gesetzt, damit diese voll ausgenutzt wird.

Die Einheiten `vw` und `vh` definieren eine Breite - beziehungsweise Höhe - in Relation zur Größe des Browser-Fensters. Dabei steht `vw` für die Breite und `vh` für die Höhe des Viewport. Diese Viewport Units ermöglichen es, Größen in Relation zur jeweils aktuellen Größe des Browser-Fensters zu definieren.

Den fixen Wert für die Höhe, der im `<div>`-Element des Karten-Objektes enthalten war, habe ich entfernt. So verschwindet auch die Bildlaufleiste am rechten Rand.

Außerdem ist in diesem Beispiel das Meta-Element `viewport` hinzu gekommen. Somit kann die Seite für mobile Geräte nicht künstlich verkleinert oder vergrößert werden – zoomen ist aber immer noch möglich!

Bei der Anzeige auf mobilen Geräten spielt das Meta-Element `viewport` eine große Rolle. In unserem Beispiel haben wir die Attribute `initial-scale=1.0`, `maximum-scale=1.0`, und `user-scalable=no` verwendet.

Das Attribut `initial-scale` legt die anfängliche Zoom-Stufe fest. `1.0` führt dazu, dass die Inhalte in ihrer Originalgröße dargestellt werden. Das bedeutet, dass auf einem Display mit 320 Pixel Breite, eine 320 Pixel breite Grafik die volle Breite ausfüllt. Im Gegensatz dazu würde der Wert `2.0` zu einer 2-fachen Vergrößerung führen – das Bild wäre dann nur noch halb zu sehen.

Das Attribut `user-scalable` bestimmt, ob der Nutzer die Anzeige der Website vergrößern oder verkleinern kann.

Die Attribute `minimum-scale` und `maximum-scale` ermöglichen es die Zoom-Stufe einzuschränken. Die Vorgabe `maximum-scale=1.0`, bewirkt, dass der Inhalt nicht vergrößert werden kann.

## JavaScript

Die Karte wird nun passend auf der Website angezeigt. Soweit so gut! Wir können die Karte aber noch benutzerfreundlicher machen. Jemand der mit einem mobilen Gerät im Internet unterwegs ist, nutzt sein Gerät in der Regel an verschiedenen Standorten. Und meistens ist es so, dass das, was in der Nähe ist, am interessantesten ist. Ideal wäre es also, wenn die Karte sich sofort so öffnet, dass der aktuelle Standort in der Mitte der Karte dargestellt wird. Und das ist möglich. Leaflet bietet Ihnen Funktionen, die Sie nutzen können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0">
<style>
body {
padding: 0;
margin: 0;
}
html, body, #mapid {
height: 100vh;
width: 100vw;
}
</style>
</head>
<body>
<div id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick: false}).setView([50.27264, 7.26469], 18)
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

mymap.locate({setView: true, maxZoom: 18});
mymap.on('locationfound', onLocationFound);
mymap.on('locationerror', onLocationError);

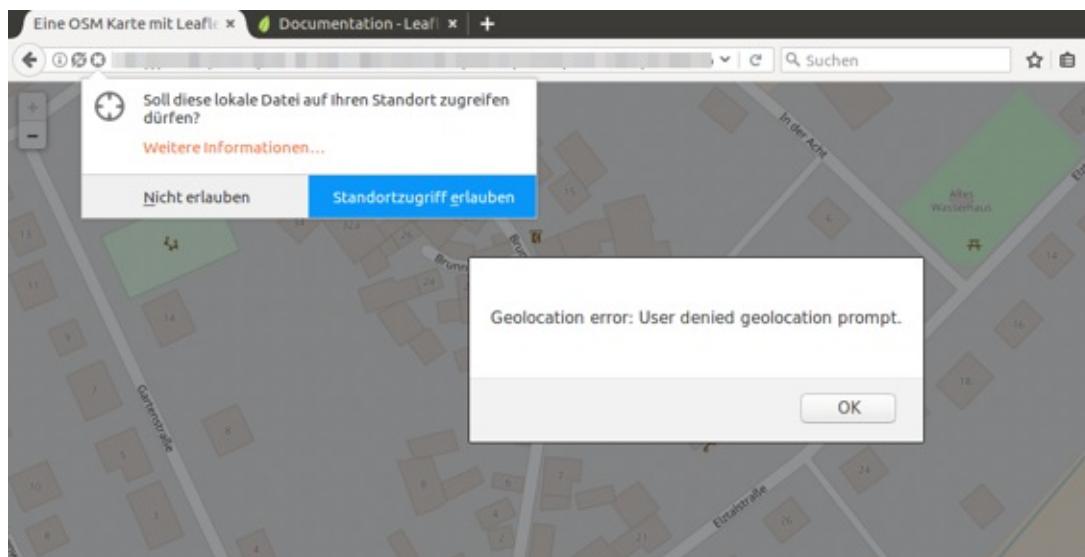
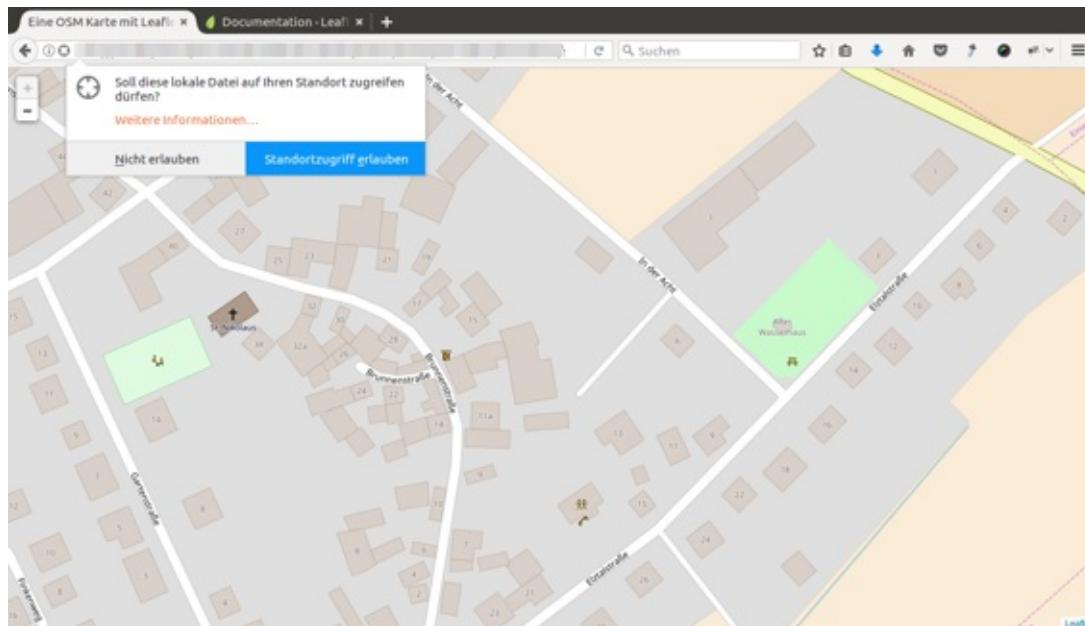
function onLocationFound(e) {
var radius = e.accuracy / 2;
L.marker(e.latlng).addTo(mymap).bindPopup("Sie sind etwa" + radius + " Meter von diesem");
L.circle(e.latlng, radius).addTo(mymap);
}

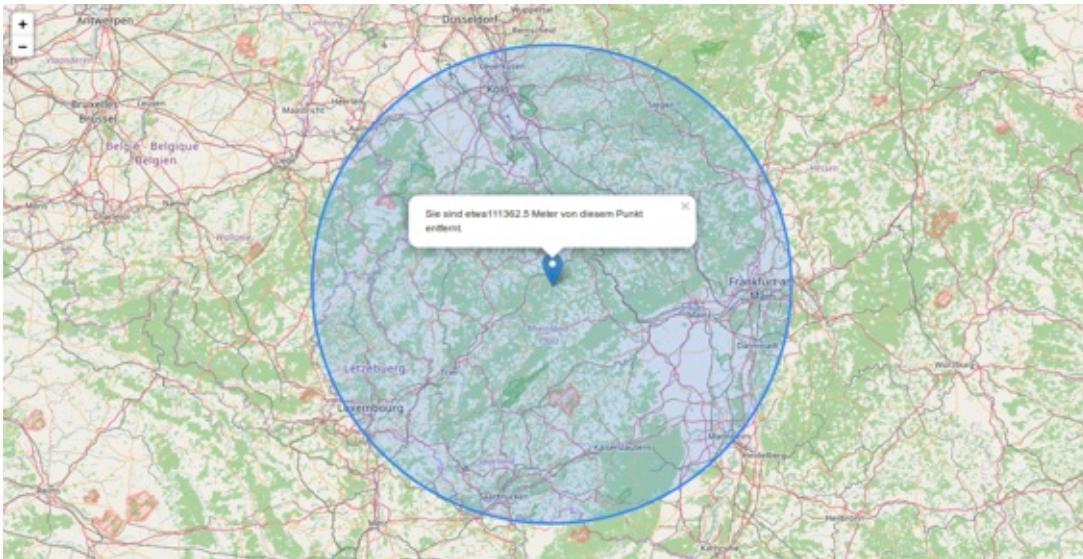
function onLocationError(e) {
alert(e.message);
}
</script>
</body>
</html>
<!--index_977.html-->
```

Wenn Sie eine Website besuchen, die standortbezogenes Surfen unterstützt – zum Beispiel die Leaflet Karte im Beispielcode – prüft der Browser, ob Sie Ihren Aufenthaltsort bekannt geben möchten. Wenn Sie zustimmen, berechnet der Browser Ihren Standort über nahe gelegene Funkzugangsknoten oder die IP-Adresse Ihres Computers. Diese ungefähre Ortsangabe wird dann an die anfragende Website weitergegeben.

Wenn Sie Ihren Standort freigegeben haben, dann sehen beim Aufruf der Karte einen Kreis, der Ihren ungefähren Standort wiedergibt und ein Pop-up Fenster, dass Ihnen erklärt wo Sie sich gerade befinden. Möchten Sie Ihren Standort nicht mit anderen teilen, dann sehen Sie eine Fehlermeldung beim Öffnen der Karte. Diese Fehlermeldung erklärt Ihnen, warum Sie nicht geortet werden können.

Damit Sie sich auch Bild davon machen könne, wenn Sie das Beispiel gerade nicht praktisch nachvollziehen können, habe ich Ihnen Bilder eingefügt.





Leaflet bietet Ihnen noch weitere Geolokalisierungsmethoden und Geolokalisierungseignisse an.

## Ereignisse in Leaflet

Bisher haben wir ausschließlich mit statischen Daten gearbeitet. In der Welt passiert aber fortwährend ganz schön viel und Sie möchten sicherlich mit Ihrer Karte auf das ein oder andere Ereignis reagieren. Vielleicht möchten Sie je nach Standort einen Pop-up Text ändern. Oder Sie möchten auf ein anderes Ereignis reagieren. Vielleicht wissen Sie auch noch gar nicht so genau auf welche Ereignisse Sie reagieren können. Dann schen Sie sich zur Inspiration doch einfach einmal an, was Leaflet Ihnen bietet. Insgesamt bietet Leaflet Ihnen 34 verschiedene Ereignisse, die Sie nutzen können. Alle Ereignisse sind in der Dokumentation gut erklärt.

Exemplarisch sehen wir uns mit dem folgenden Programmcode – wieder anhand eines ganz einfache Beispiels – die Vorgehensweise genauer an. Konkret reagieren wir auf einen Mausklick.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick: false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

mymap.on('click',
function(){
alert('Sie haben auf die Karte geklickt.')
}
);
```

```
</script>
</body>
</html>
<!--index_976.html-->
```

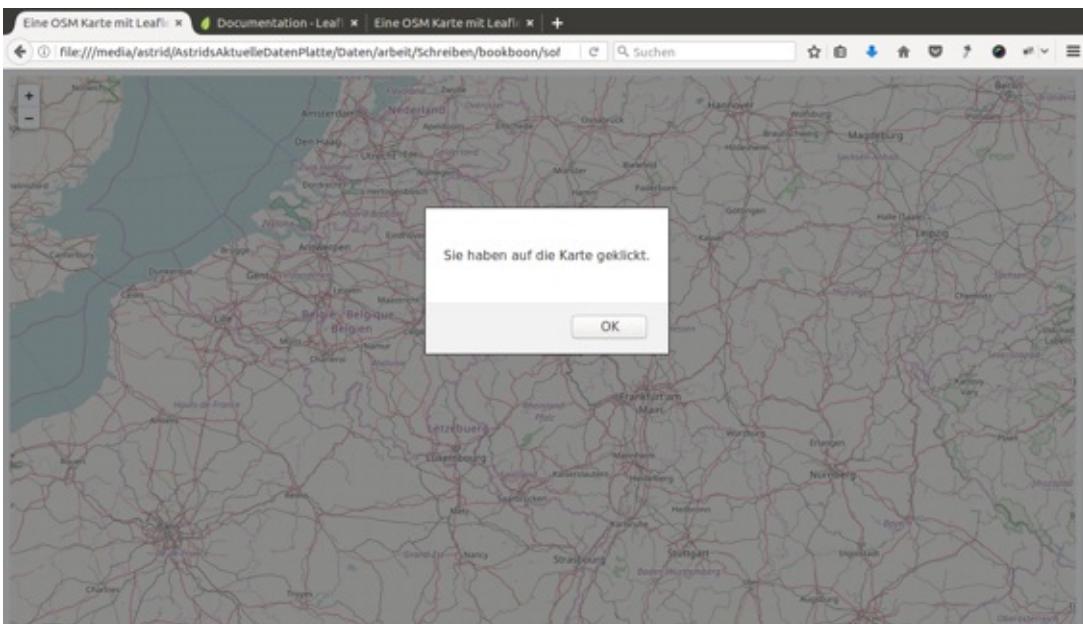
Leaflet bietet die Methode `on()` um auf ein Ereignis zu reagieren. Der erste Parameter, der mit dieser Methode übergeben wird, steht für den Ereignistyp. In unserem Falle ist dies `click`. Der zweite Wert erwartet eine Funktion, die beim Eintreten des Ereignisses ausgeführt werden soll.

In unserem Beispiel haben wir die Funktion sofort in den Methodenaufruf eingefügt:

```
mymap.on('click',
function(){
  alert('Sie haben auf die Karte geklickt.')
});
```

Diese Schreibweise wird auch als anonymer Funktionsausdruck bezeichnet, weil die Funktion keinen Namen hat.

Wenn Sie die Karte des vorhergehenden Beispiels im Browser aufrufen und anklicken, sehen Sie die in der nachfolgenden Abbildung dargestellte Meldung.



Vielleicht möchten sie selbst eigene Beispiele ausprobieren und diese anderen zeigen. Insbesondere bei der Fehlersuche ist dies oft hilfreich. Dies können Sie mithilfe des Leaflet Playgrounds tun.

## Eine benutzerdefinierte Funktionen

Die Methoden, die Leaflet Ihnen bietet, kennen Sie nun. Bisher haben Sie immer die Variable e in den Funktionen übergeben. Mit JavaScript können Sie aber auch eigene Variablen übergeben. Außerdem können Sie eigene Funktionen an beliebigen Stellen im Programmcode aufrufen.

Probieren wir das doch sofort einmal aus. Im nächsten Beispiel erstellen wir einen Marker und verbinden diesen mit einem Pop-up Fenster – und das mithilfe einer selbst erstellten Funktion.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick: false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

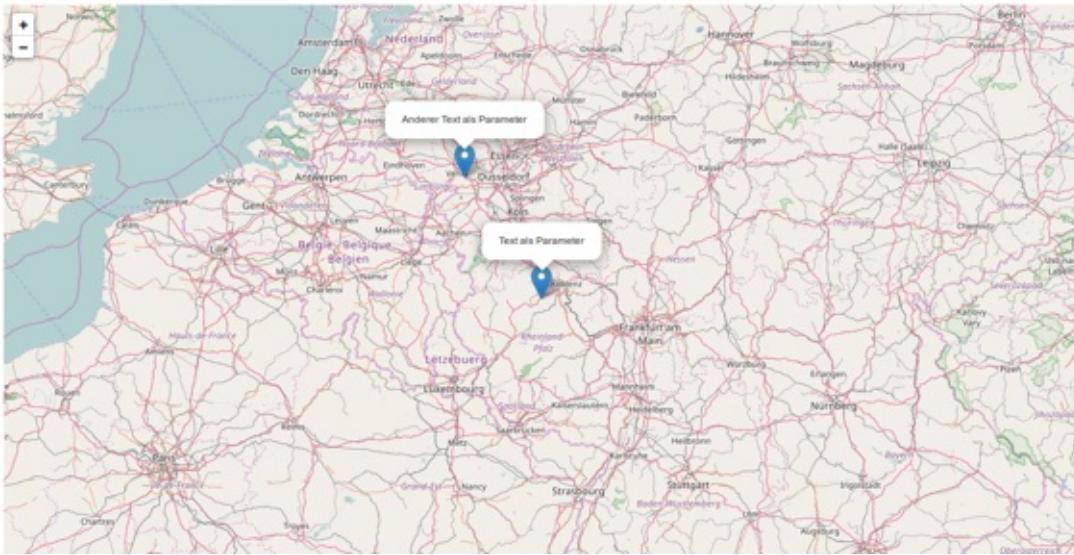
var marker1 = L.marker([50.27264, 7.26469]).addTo(mymap).bindPopup(createPopup("Text al:
var marker2 = L.marker([51.27264, 6.26469]).addTo(mymap).bindPopup(createPopup("Anderer

function createPopup(popuptext){
return L.popup({autoClose:false, keepInView:true, closeButton:false}).setContent(popupt
}

</script>
</body>
</html>
<!--index_975.html-->
```

In diesem Beispiel ging es nur um das Grundsätzliche und die Mächtigkeit von eigenen Funktionen wird mit nur zwei Markern noch nicht klar. Vielleicht können Sie sich aber vorstellen, wie viel Text mithilfe einer benutzerdefinierten Funktion eingespart werden kann.

Achtung: Wenn Sie bei einem Pop-up Fenster die Eigenschaften `autoClose:false` und `closeButton:false` gleichzeitig setzen, kann dieses Pop-up Fenster tatsächlich nicht mehr geschlossen werden.



## Ein interaktives Beispiel

Und zu guter Letzt möchte ich Ihnen noch ein interaktives Beispiel zeigen.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<p>Bitte geben Sie eine Koordinate an:</p><br>
Geographische Breite:<input type="text" id="lat"><br>
Geographische Länge:<input type="text" id="long"><br>
Name:<input type="text" id="name"><br>
<button onclick="save()">Speichern</button>
<div style="height: 700px;" id="mapid"></div>

<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

function save(){
var p1 = parseFloat(document.getElementById("lat").value);
var p2 = parseFloat(document.getElementById("long").value);
L.marker([p1,p2]).bindPopup(document.getElementById("name").value).addTo(mymap);
}
</script>
</body>
</html>
<!--index_974.html-->
```

Im vorhergehenden Beispiel ist eine Schaltfläche integriert, die – wenn sie angeklickt wird – eine benutzerdefinierte Funktion ausführt. Mit der Funktion wird ein Marker mit einem Pop-up Fenster in die Karte eingesetzt. Dies ist nur als Beispiel gedacht. Die können in der benutzerdefinierten Funktion alles Mögliche ausführen – lassen Sie Ihrer Fantasie freien Lauf. Ein weiterer Anwendungsfall könnte eine Art Heiß-Kalt Spiel sein. Sie kennen sicher das einfache Suchspiel.

Möchten die interaktiv eingefügten Daten in einem Objekt speichern, um einen Überblick über diese zu haben und eventuell weitere Aktionen von dem Datenbestand abhängig zu machen?

Das vorhergehende Beispiel könnten Sie in diesem Falle wie folgt erweitern. Sie könnten mithilfe der Zeile

```
var myStorage = localStorage;
```

ein localStorage Objekt instanziieren. Mit

```
myStorage.setItem(  
document.getElementById("name").value,  
document.getElementById("lat").value + "," + document.getElementById("long").value  
)
```

könnten Sie daraufhin alle eingefügten Objekte speichern und mit diesen arbeiten.

Die sessionStorage-Eigenschaft erlaubt den Zugriff auf ein, nur während der aktuellen Sitzung, verfügbares Storage-Objekt. sessionStorage ist mit einer Ausnahme identisch zu window.localStorage: In localStorage gespeicherte Daten besitzen kein Verfallsdatum, während Sie im sessionStorage mit Ablauf der Sitzung gelöscht werden.

Eine Sitzung endet mit dem Schließen des Browsers, sie übersteht das Neuladen und Wiederherstellen einer Webseite. Das Öffnen einer Webseite in einem neuen Tabulator oder Browser-Fenster erzeugt jedoch eine neue Sitzung. Dies unterscheidet ein sessionStorage Objekt von einem Session-Cookie: <https://developer.mozilla.org/de/docs/Web/API/Window/sessionStorage>

## In diesem Kapitel haben wir ...

In diesem Kapitel haben Sie Punkte, Marker, Linien und Polygone kennen gelernt. Sie können diese nun auf eine Leaflet-Karte zeichnen und wissen, wann welches Objekt das Richtige ist. Sie können Layer-Gruppen und Featur-Gruppen voneinander unterscheiden und wissen die Grundlagen zur Anzeige von mobilen Leaflet Karten. Und auf einen Mausklick oder ein anderes Ereignis können Sie entsprechend reagieren.

Im nächsten Kapitel geht es um das Format GeoJson und darum, wie Sie Daten auch in großen Mengen gut Handhaben können.

# GeoJSON

GeoJSON ist ein offenes Format, welches es einfach macht, geografische Daten zu beschreiben. Dabei richtet es sich nach einer Spezifikation – nämlich nach der Simple-Feature-Access-Spezifikation. Für die Beschreibung der Geodaten verwendet GeoJSON die JavaScript Objekt Notation (JSON).

Hinter dem Begriff Simple Feature Access-Spezifikation versteckt sich eine Spezifikation des Open Geospatial Consortium (OGC). Diese Spezifikation beinhaltet eine allgemein gültige Beschreibung für Geodaten und deren Geometrien. Dadurch, dass die Spezifikation allgemeingültig ist, können diese Daten ausgetauscht werden. Das OGC ist eine gemeinnützige Organisation, die die Entwicklung von allgemeingültigen Standards für Geodaten zum Ziel hat.

## In diesem Kapitel werden wir ...

Als Erstes sehen wir uns an, warum GeoJSON entwickelt wurde. Als Nächstes vergleichen wir die einzelnen GeoJSON Elemente mit den Objekten, die uns Leaflet zur Verfügung stellt. Und zu guter Letzt probieren wir die Methoden aus, die Leaflet uns – spezielle für das Verarbeiten von GeoJSON-Daten – anbietet.

## Die Entwicklungsgeschichte von GeoJSON

GeoJSON baut auf JSON auf und bevor JSON als Datenformat spezifiziert wurde, gab es die erweiterbare Auszeichnungssprache XML (englisch Extensible Markup Language). Immer wenn etwas Neues entsteht, gibt es einen Grund dafür. XML wurde 1998 veröffentlicht, um Daten zwischen Maschinen austauschen zu können, ohne das Menschen nacharbeiten müssen. Dies wurde in Zeiten des Internets immer wichtiger. Nun muss es auch einen Grund geben, warum neben XML erst JSON – und später GeoJSON entstanden ist.

## Warum ist das Format GeoJSON entstanden?

XML hat für den Austausch von Daten nicht ausgereicht. Warum war dies so und welche Vorteile bietet JSON, beziehungsweise GeoJSON? Zunächst einmal bieten alle drei Formate folgendes:

- Alle drei Formate können von einem Menschen gelesen und verstanden werden.
- Alle drei Formate sind hierarchisch gegliedert. Das bedeutet, dass Werte innerhalb von anderen Werten dargestellt werden können.

- Alle drei Formate sind relativ leicht zu erlernen.
- Alle drei Formate können von vielen Programmiersprachen analysiert und genutzt werden.
- Alle drei Formate sind mit mithilfe des Hypertext Transfer Protokolls (HTTP) – also über das Internet – austauschbar.

Sehen wir uns in diesem Kapitel die einzelnen Formate einmal genauer an um zu erkennen, welche Vorteile das Format JSON – beim Arbeiten mit Geodaten das Format GeoJSON – gegenüber XML bringt.

## XML

XML beschreibt die Struktur von Daten. Anhand der *Tags* wird den Daten eine Bedeutung – eine Semantik – gegeben. Durch das Tag-System von XML werden oft kleine Datenbestände sehr aufgebläht und unübersichtlich. Zusätzlich ist das Ansprechen einzelner Tags in einer XML-Datei nicht immer leicht.

## JSON

JSON ist im Grunde genommen nichts anderes als die Festlegung auf eine bestimmte Syntax – also eine Syntax-Konvention. Den Daten wird keine bestimmte Bedeutung geben, vielmehr geht es um die syntaktische Anordnung. Da JSON Daten strukturiert, können leicht Objekte aus diesen Daten definiert werden. Im Dezember 1999 wurde die erste JSON Format-Spezifikation verabschiedet. Im Juni 2017 wurde die *8. Edition des Standards ECMA-262* veröffentlicht.

Der große Vorteil von JSON im Vergleich zu XML liegt in der einfachen Handhabung. Da JSON selbst gültiges Javascript darstellt, kann es direkt ausgeführt und somit in ein Javascript-Objekt überführt werden. Auf die einzelnen Eigenschaften dieser Objekte kann dann über die Attribute zugegriffen werden. Im Kapitel Heatmaps in Leaflet – Dichte werden wir eine Datei, die GeoJSON Objekte enthält, als Skript einbinden. Allein durch das Einbinden der Datei können wir innerhalb anderer Skripte auf die GeoJSON Objekte der eingebundenen Datei zugreifen. Im Gegensatz dazu muss eine XML-Datei erst mit einem XML-Parser analysiert werden.

Ein weiterer Vorteil von JSON: In JSON gibt es kein Ende-Tag. Hauptsächlich deshalb ist JSON kompakter. Dies hat zur Folge, dass JSON schneller gelesen und verarbeitet werden kann.

Die Daten einer jeden GeoJson-Datei, die sich in einem GitHub Repository befindet, werden – wenn man die Datei im Repository anklickt – automatisch auf einer interaktiven Karte angezeigt.

Github erstellt diese Karte schon seit 2013 mit Leaflet. Dies können Sie sich beispielsweise im Repository world.geo.json ansehen.

Schon ein kleines Beispiel veranschaulicht, dass XML für das Beschreiben des gleichen Objektes mehr Zeichen benötigt als JSON. Ein in XML mit 95 Zeichen kodiertes Objekt benötigt in JSON gerade einmal 73 Zeichen. Bei einem Objekt ist dieser Unterschied vernachlässigbar. In der Regel werden aber eine Vielzahl von Objekten digital beschrieben. Bei einer Vielzahl von Objekten kann dieser Unterschied stark ins Gewicht fallen.

Hier sehen Sie zunächst den aus 95 Zeichen bestehende XML Ausschnitt.

```
<joomlers>
<number>1721</number>
<vorname>Astrid</vorname>
<nachname>Günther</nachname>
</joomlers>
```

Das gleiche Objekt kann mit 73 Zeichen im JSON Format beschrieben werden.

```
„joomlers“: {
„number“: „1721“,
„vorname“: „Astrid“,
„nachname“: „Günther“
},
```

## Und warum nun auch noch *GeoJSON*?

Geodaten könnten in JSON beschrieben und verarbeitet werden. Welchen Vorteil bringt das spezielle GeoJSON-Format? GeoJSON ist JSON – allerdings auf Geodaten spezialisiert. GeoJSON gibt den Geodaten wieder eine Semantik – also eine Bedeutung.

GeoJSON beschreibt Punkte, Linien und Polygone und kann gut mit diesen Formen in einem Koordinatensystem umgehen. Im vorausgehenden Kapitel haben wir gesehen, dass das Arbeiten mit Geodaten im Grunde genommen nichts anderes ist.

GeoJSON hat sich zu einem sehr beliebten Datenformat vieler Geoinformationssysteme entwickelt. In diesem Buch erwähne ich GeoJSON speziell, weil auch Leaflet sehr gut im Umgang mit Daten im GeoJSON Format ist. Hier sehen wir uns zunächst die GeoJSON Objekte einmal genauer an. Wenn Sie lieber sofort praktisch arbeiten möchten, dann Blättern Sie am besten ein Kapitel weiter. Im Kapitel GeoJson in Leaflet erfahren Sie, wie Sie GeoJSON-Elemente auf Ihrer Karte anzeigen und weiter bearbeiten können.

Im Juni 2008 wurde die erste GeoJSON Format-Spezifikation verabschiedet. Im August 2016 wurde die RFC (Requests for Comments) 7946 veröffentlicht.

Die formale Spezifikation des GeoJSON Formates finden Sie unter der Adresse <https://tools.ietf.org/html/rfc7946> im Internet.

## GeoJSON erkunden

Sie wissen nun, dass Sie mit GeoJSON viele geografische Datenstrukturen in einer maschinenlesbaren Sprache kodieren können. Ein GeoJSON-Objekt kann dabei eine einfache Geometrie, zum Beispiel einen Punkt eine Linie oder ein Polygon, darstellen. Zusätzlich können Sie einer Geometrie aber auch Eigenschaften zuordnen. In diesem Fall erstellen Sie ein Objekt vom Typ Feature. Wenn Sie mehrere Feature-Objekte zu einer Gruppe zusammen fassen möchten, können Sie diese zu einer Sammlung von Features zusammen fassen. Hierfür gibt es den GEOJson Typ FeatureCollection. Das Verständnis dieser Konzepte bringt viele Vorteile. Es hilft Ihnen auch, die Arbeit mit Geodaten im Allgemeinen zu verstehen: Die Grundkonzepte, die in GeoJSON angewandt werden, sind schon seit vielen Jahren ein Teil von Geoinformationssystemen. Beginnen wir aber von vorne.

### Eine Geometrie

Eine Geometrie ist eine Form. Alle Formen in GeoJSON werden mit einer oder mehrerer Koordinaten beschrieben. Eine Koordinate heißt in GeoJSON Position. GeoJSON unterstützt die Geometriearten

1. Point,
2. LineString,
3. Polygon,
4. MultiPoint,
5. MultiLineString, und
6. MultiPolygon

– und jede dieser Geometriearten beinhaltet Positionen.

### Position

Das wichtigste Element beim Arbeiten mit Geodaten ist die Beschreibung des Punktes auf der Erde. Der Punkt auf der Erde ist der, dem die Geodaten zugeordnet werden. Diesen Wert kennen wir auch unter dem Namen Koordinate. Im Kapitel Das Koordinatensystem der Erde habe ich schon eine ganze Menge zum Thema Koordinaten auf der Erde geschrieben. Hier noch einmal kurz: Eine Koordinate ist eine Zahlenkombination. Jede Zahl einer Koordinate steht für eine Dimension. Wir beschränken uns in diesem Buch auf zwei Dimensionen, nämlich die geografische Längen und die geografische Breite. GeoJSON unterstützt drei Dimensionen – neben der geografischen Länge und der geografischen Breite können Sie zusätzlich die

Höhe auf der Erde angeben.

Beim globalem Navigationssatellitensystem (GPS) ist noch eine vierte Größe relevant. Neben der horizontalen Position und der Höhe spielt auch die aktuelle Zeit eine Rolle.

Die Koordinaten werden in GeoJSON im Dezimalformat formatiert. Die erste Zahl steht für die Longitude – also die geografische Länge – und die zweite Zahl für die Latitude – also die geografische Breite. Konkret sieht eine Position in GeoJSON so aus:

[Länge, Breite, Höhe] oder [50.254, 7.5847, 324.1]

Velleicht haben Sie in der Vergangenheit schon öfter mit Geodaten gearbeitet und wundern sich nun über die Reihenfolge, in der die Dimensionen im Format GeoJSON stehen? Viele Systeme geben zuerst die geografische Länge und erst dann die geografische Breite an. Auch in Leaflet wird bei der Koordinate zuerst die Latitude - also die geografische Breite - und erst dann die Longitude - also die geografische Länge - angegeben: Breitengrad | Längengrad. Um dieses Durcheinander zu verstehen, müssen Sie folgendes bedenken: Früher war es üblich, dass die erste Stelle einer Koordinate den Breitengrad und die zweite Stelle den Längengrad beschrieb. In der Mathematik ist die übliche Reihenfolge beim Arbeiten mit Koordinatensystemen: X-Wert | Y-Wert. Wenn man eine Landkarte mit einem Koordinatensystem vergleicht, erkennt man schnell, dass der Breitengrad dem X-Wert und der Längengrad dem Y-Wert entspricht. Dies hat zur Folge, dass es beim Rechnen mit einem Computer viele Vorteile bringt, wenn man die Reihenfolge Längengrad | Breitengrad einhält. In der digitalen Welt gibt es momentan noch keine Einigkeit über die Reihenfolge. Es sieht so aus, als ob wir mitten in einem Umbruch stecken. Eine Übersicht, die zeigt, welche Anwendung welche Dimension als erstes verwendet, finden Sie unter anderem unter der Adresse <https://macwright.org/lonlat/>.

Früher erlaubte GeoJSON die Speicherung von mehr als drei Zahlen pro Position. Diese Möglichkeit wurde auch genutzt. Es wurden beispielsweise Sportdaten wie die Herzfrequenz zusammen mit der Position gespeichert. Da dies nicht der Sinn einer Position ist, führte dieses Vorgehen teilweise zu Problemen in anderen GeoJSON Anwendungen. In der neuen Spezifikation ist das Speichern von mehr als drei Werten pro Position nun nicht mehr zulässig.

## Point

Der Typ Point – also ein Punkt – ist die einfachste Geometrie in GeoJSON. Er gibt die Koordinaten einer bestimmten Position an. Die genaue Schreibweise sehen Sie nachfolgend.

```
{ "type": "Point",
  "coordinates": [30, 10]
}
```

In Leaflet wird der Typ Point als Marker eingefügt.

## Multipoint

Der Typ MultiPoint wird mit einem Array von Positionen beschrieben. Mit ihm können mehrere Punkte auf der Erde angegeben werden.

```
{ "type": "MultiPoint",
  "coordinates": [
    [10, 40], [40, 30], [20, 20], [30, 10]
  ]
}
```

Mit dem Typ Multipoint können Sie mehrere Marker auf einen Schlag zu Ihrer Leaflet Karte hinzufügen.

## LineString

Um eine Linie darzustellen, benötigen Sie mindestens zwei Punkte. Die Linie ist die Verbindung zwischen diesen Punkten. Eine Linie wird mit einem Array von zwei oder mehr Positionen gebildet. In GeoJSON wird eine Linie mit dem Typ LineString dargestellt.

```
{ "type": "LineString",
  "coordinates": [
    [30, 10], [10, 30], [40, 40]
  ]
}
```

Ein GeoJSON Objekt vom Typ LineString entspricht einem Polyline Objekt in Leaflet.

## MultiLineString

Beim Typ MultiLineString werden die Koordinaten mit einem Array von LineString-Koordinaten-Arrays angegeben.

```
{ "type": "MultiLineString",
  "coordinates": [
    [
      [10, 10], [20, 20], [10, 40]
    ],
    [
      [40, 40], [30, 30], [40, 20], [30, 10]
    ]
  ]
}
```

```
]  
}
```

Ein GeoJSON Objekt vom Typ MultiLineString entspricht einem Leaflet Polyline Objekt, welches mehr als eine abgeschlossene Linie definiert. Das bedeutet, dass alle Linien zusammen auf einem Layer gezeichnet werden.

Wie bei den Objekten in Leaflet gilt auch hier: Linien und Punkte sind die einfachsten Geometriegerüste. Bei beiden müssen Sie nicht viele geometrische Regeln beachten. Ein Punkt kann irgendwo an einem Ort liegen und eine Linie kann eine beliebige Anzahl an Punkten enthalten. Eine Linie kann sich selbst überqueren. Punkte und Linien haben keine Fläche – somit gibt es auch kein *Außen* und kein *Innen*.

## Polygone

Im Vergleich zu Linien sind Polygone komplexe Geometrien. Polygone verfügen über eine Fläche. Es gibt also einen Innenbereich, der sich von einem Außenbereich unterscheidet. Und hinzu kommt: Der Innenbereich kann Löcher haben! Wie die Löcher in einem Polygon entstehen, habe ich Ihnen im Kapitel Die Karte mit Daten bestücken im Unterkapitel *Polygone* bereits erklärt. Ein GeoJSON Objekt vom Typ Polygon entspricht einem Polygon Objekt in Leaflet. Die Koordinatenliste enthält bei einem Polygon – analog zu Leaflet – eine Ebene mehr als die Koordinatenliste des Typs LineString. Ich habe die relevante Klammerung im nachfolgenden Beispiel fett formatiert.

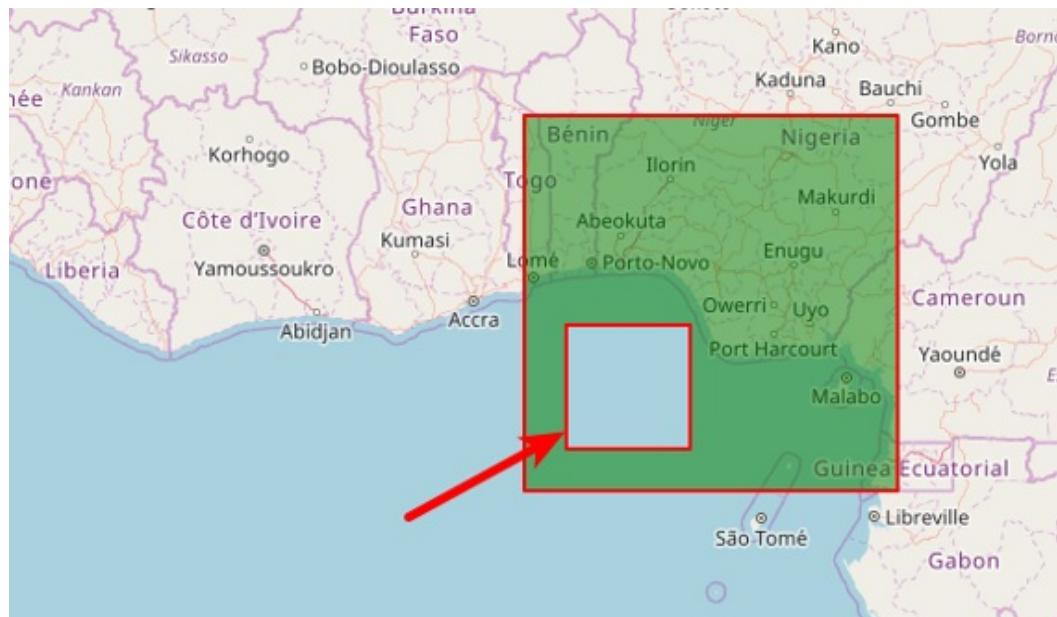
```
{ "type": "Polygon",  
"coordinates": [  
  [  
    [30, 10], [40, 40], [20, 40], [10, 20], [30, 10]  
  ]  
]
```

Bei einem einfachen Polygon ist der Sinn dieser Ebene nicht offensichtlich. Auf den ersten Blick könnten Sie der Meinung sein, dass es einfacher wäre, das Polygon genau wie die Linie zu erstellen. Dass es sich um ein Polygon handelt, ist über den Eintrag bei der Eigenschaft Typ klar – und wenn es sich um den Typ Polygon handelt, wird die Linie einfach geschlossen! Der erste Blick ist oft trügerisch. Wir benötigen diese zusätzliche Möglichkeit der Klammerung oder Verschachtelung um Löcher in die Fläche zeichnen zu können. Polygone sind in GeoJSON mehr als nur geschlossene Linien. Ich wiederhole mich. Polygone haben einen Innenbereich, und dieser Innenbereich kann Löcher haben. Aus diesem Grund ist beim Typ Polygone in der GeoJSON Spezifikation ein neuer Begriff zu lesen, nämlich der Begriff LinearRing. Ein LinearRing ist ein geschlossener LineString mit vier oder

mehr Positionen. Obwohl ein *LinearRing* nicht explizit als GeoJSON-Geometrie-Typ eingeführt ist, wird der Begriff in der Polygon-Geometrie-Typ-Definition der Spezifikation erwähnt.

Ein *LinearRing* ist entweder die äußere Ringposition, die die äußere Kante des Polygons bildet und definiert, welche Flächen gefüllt sind. Ein *LinearRing* kann aber auch ein Innenring sein, der die Flächen des Polygons definieren, die leer sind. Es kann eine beliebige Anzahl von Innenringen geben, einschließlich null Innenringen. Wenn das Polygon über keinen Innenring verfügt bedeutet dies, dass das Polygon nur einen Innenbereich und einen Außenbereich hat – also keine Löcher hat.

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [
        [1, 1], [1, 10], [10, 10], [10, 1], [1, 1]
      ],
      [
        [2, 2], [2, 5], [5, 5], [5, 2], [2, 2]
      ]
    ]
  ]
}
```



*Ein Polygon mit einem Innenring. Der Innenring definiert einen Außenbereich im Polygon – er schneidet quasi ein Loch in das Polygon.*

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [
        [1, 1], [1, 10], [10, 10], [10, 1], [1, 1]
      ],
      [
        [2, 2], [2, 5], [5, 5], [5, 2], [2, 2]
      ],
      [
        [3, 3], [3, 4], [4, 4], [4, 3], [3, 3]
      ]
    ]
  ]
}
```



*Ein Polygon mit zwei Innenringen – der zweite Innenring wird innerhalb des ersten Innenringes gezeichnet. Dieser zweite*

]} *Innenring zeichnet einen neuen Innenbereich in den Außenbereich der durch den ersten Innenring entstanden ist.*

Vielleicht ist Ihnen aufgefallen, dass die erste und die letzte Koordinate jedes LinearRings gleich ist. Die Wiederholung der Koordinate ist bei einem Leaflet Objekt nicht erwünscht. Hier werden die Ringe eines Polygone automatisch geschlossen. Die erste und letzte Position eines GeoJSON LinearRing muss im Gegensatz dazu identisch sein.

## **MultiPolygon**

Beim Typ MultiPolygon werden die Koordinaten mit einem Array von Polygon-Koordinaten-Arrays angegeben. Hier sehen Sie zunächst ein Beispiel, dass zwei einfache Polygone darstellt.

```
{ "type": "MultiPolygon",
  "coordinates": [
    [
      [
        [30, 20], [45, 40], [10, 40], [30, 20], [30, 20]
      ],
      [
        [
          [15, 5], [40, 10], [10, 20], [5, 10], [15, 5]
        ]
      ]
    ]
  }
}
```

Es geht aber auch komplizierter. Sie können auch mehr als ein Polygon mit Löchern – also mehr als einem LinearString – zusammen als MultiPolygon gruppieren.

```
{ "type": "MultiPolygon",
  "coordinates": [
    [
      [[40, 40], [20, 45], [45, 30], [40, 40]]
    ],
    [
      [
        [[20, 35], [10, 30], [10, 10], [30, 5], [45, 20], [20, 35]],
        [[30, 20], [20, 15], [20, 25], [30, 20]]
      ]
    ]
  }
}
```

Ein GeoJSON Objekt vom Typ MultiPolygon entspricht einem Leaflet Polygon

Objekt, welches mehr als ein Polygon definiert. Das bedeutet, dass alle Formen zusammen auf einen Layer gezeichnet werden.

### Mehrere Geometrien zusammenfassen - GeometryCollection

Geodaten wollen die Welt beschreiben. Jeder der grundlegenden GeoJSON Typen ist ideal für die Darstellung eines Objektes auf der Erde. Unsere Welt enthält eine Menge Objekte, die gemeinsame Eigenschaften haben. Wenn wir diesen Objekten diese gemeinsamen Eigenschaften auf einen Schlag zuordnen möchten, können wir diese Objekte mit dem Typ GeometryCollection zusammenfassen.

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "GeometryCollection",  
    "geometries": [{  
      "type": "Point",  
      "coordinates": [0, 0]  
    }, {  
      "type": "LineString",  
      "coordinates": [[0, 0], [1, 0]]  
    }]  
  },  
  "properties": {  
    "name": "Der Name dieser GeometryCollection"  
  }  
}
```

Einen Anwendungsfall für eine Geometriekollektionen gibt es in der Praxis allerdings nur sehr selten: Meist ist es so, dass jede Geometrie auch eigene Eigenschaften besitzt. Im nächsten Kapitel werden Sie lesen, dass Sie eine Geometrie mit eigenen Eigenschaften im Typ Feature beschreiben können und Feature Objekte werden mit dem Typ FeatureCollection zusammengefasst.

### Einer GeoJSON Geometrien Eigenschaften zuordnen

Geometrien sind Formen – nicht mehr und nicht weniger. Sie sind ein zentraler Teil von GeoJSON, aber die meisten Daten, die etwas mit der Welt zu tun haben, sind nicht einfach nur eine Form. Die Formen haben auch eine Identität und Attribute. Ein Polygon stellt beispielsweise den Reichstag dar. Ein anderes Polypen ist die Grenze von Deutschland. Und bei der Arbeit mit den Geometrien ist es wichtig zu wissen, welche Geometrie was ist und, welche Eigenschaften die Geometrie hat. In GeoJSON kann genau dies mit einem Objekt des Typs Feature erreicht werden.

Das nachfolgende Programmcodebeispiel enthält ein ganz einfaches Feature Element.

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Point",  
    "coordinates": [0, 0]  
  },  
  "properties": {  
    "name": "Reichstag"  
  }  
}
```

```

"geometry": {
  "type": "Point",
  "coordinates": [0, 0]
},
"properties": {
  "name": "Der Name des Punktes"
}
}

```

Das nachfolgende Programmcodebeispiel enthält ein etwas komplexeres Feature. Sie sehen hier, dass eine Eigenschaft eines Feature Elements mit jedem JSON-Objekt Datentyp beschrieben werden kann.

```

{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [[30, 20], [45, 40], [10, 40], [30, 20]]
  },
  "properties": {
    "prop0": "value0",
    "prop1": {"this": "that"},
    "prop2": true,
    "prop3": null,
    "prop4": ["wert1", "wert2", "wert3"],
    "prop5": 0.0
  }
}

```

## FeatureCollection

So nun haben wir jede Menge Typen kennen gelernt. Den Typ, den Sie sicherlich am meisten nutzen werden, habe ich für den Schluss aufgehoben. Die Syntax einer FeatureCollection können Sie im nachfolgenden Beispiel ablesen.

```

{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [0, 0]
      },
      "properties": {
        "name": "Der Name des Features1"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [0, 0]
      },
      "properties": {
        "name": "Der Name des Features2"
      }
    }
  ]
}

```

```
}
```

```
]
```

Ein Objekt vom Typ FeatureCollection enthält ein Schlüssel-Wert-Paar. Der Wert ist ein Array das aus Feature Objekten besteht und der Schlüssel lautet Features. Wie der Name schon sagt, darf das Array ausschließlich Objekte vom Typ Feature enthalten.

Welche Vorteile bringt ein Objekt vom Typ FeatureCollection zusätzlich zu den einzelnen Feature Objekten? Ein Objekt vom Typ FeatureCollections ist sehr sinnvoll, wenn Sie mit GeoJSON-Typen arbeiten, die gemeinsame Eigenschaften haben. Im nächsten Kapitel wird Ihnen dies anhand von praktischen Beispielen klar werden.

#### Hinweis:

Sie möchten gerne mit GeoJSON nutzen und haben auch schon die ersten Dateien erstellt. Vielleicht sind Sie auch schon auf das ein oder andere Problem gestoßen oder möchten einfach nur mit der Syntax vertraut werden. Auf der Website <http://geojsonlint.com/> können Sie Ihre GeoJSON Daten testen.

## Die Grenzen von GeoJSON

Die Vorteile von GeoJSON hatte ich Ihnen weiter vorne in diesem Kapitel näher gebracht. Wie jedes andere Format hat GeoJSON aber auch seine Grenzen. Diese sind nun Thema dieses Kapitels.

- GeoJSON hat kein Konstrukt das eine Komprimierung unterstützt wie beispielsweise TopoJSON oder OSM XML.
- GeoJSON unterstützt die gleichen Datentypen wie JSON. JSON unterstützt nicht jeden Datentyp. Zum Beispiel gibt es keinen Typ für Datumswerte in JSON.
- GeoJSON hat kein Konstrukt für die Anzeige von Pop-up Fenstern wie Titel oder Beschreibung.
- GeoJSON hat keine Geometrie vom Typ Kreis – oder irgendeine andere Art von Kurve.
- In GeoJSON können Sie den einzelnen Koordinaten – also den Positionen – keine eigene Eigenschaft zuweisen. Wenn Sie die *LineString* Darstellung eines Trainingslaufs haben und Ihr GPS Gerät mehr als 1000 verschiedene Punkte während dieses Laufs zusammen mit Ihrer Herzfrequenz protokolliert hat, bietet GeoJSON keine klare Antwort auf die Frage, wie Sie diese Daten am besten darstellen. Sie könnten eine zusätzliche Eigenschaft als Array mit der gleichen Länge wie das Koordinaten Array speichern – eine klare Regelung gibt es aber nicht.

# GeoJson in Leaflet

Leaflet unterstützt alle GeoJSON-Typen. In der Regel werden Sie überwiegend den Typ Feature und FeatureCollection nutzen. Sie möchten ja sicher nicht nur Geometrie Objekte auf Ihrer Karte anzeigen, sondern auch die Eigenschaften – also weitere Informationen – zu diesen Objekten. Die Typen Feature und FeatureCollection unterstützt Leaflet besonders gut.

## Ein GeoJSON Feature in Leaflet einbinden

Beginnen wir mit einem übersichtlichen Beispiel: Die einfachste Art GeoJSON in Ihrer Karte zu nutzen, ist die Verwendung als Variable direkt – fest programmiert. So etwas sollte man eigentlich nicht machen. Übungsbeispiele lassen sich so aber möglichst einfach darstellen. Deshalb tue ich es in den Programmcodebeispielen in diesem Buch.

Eine andere alternative Art GeoJSON Daten in ein HTML-Dokument einzubinden finden Sie im Kapitel Choroplethenkarte im Unterkapitel Open Data.

Das nachfolgende Programmcodebeispiel enthält einen Punkt. Sobald Sie diesen – in GeoJSON formatierten – Punkt in einer JSON-Variablen gespeichert haben, können Sie diesen ganz einfach zur Karte hinzufügen. Leaflet zeigt auf der Karte als Ergebnis einen Marker an.

Wir haben festgestellt, dass Leaflet Koordinaten in einer anderen Reihenfolge als GeoJSON schreibt. Wenn Sie die Standardfunktionen in Leaflet verwenden, müssen Sie sich hierüber aber keine Sorgen machen. Leaflet sortiert die Koordinaten selbstständig in die passende Form.

Hier also nun das erste praktische Beispiel zum Thema GeoJSON und Leaflet.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var geojsonFeature1 = {
" type": "Feature",
```

```

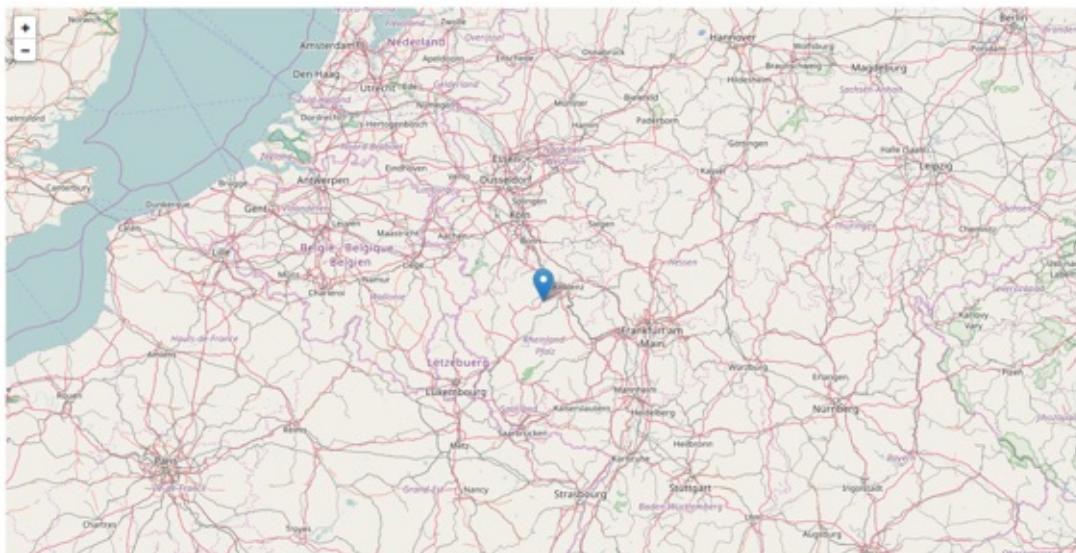
"geometry": {"type": "Point", "coordinates": [7.26469, 50.27264]},
"properties": {"name": "Gering"}
};L.geoJSON(geojsonFeature1).addTo(mymap);
</script>
</body>
</html>
<!--index_973.html-->
```

Was passiert hier genau? Als erste haben wir den GeoJSON Code fest in der Variablen geojsonFeature1 gespeichert. Als Nächstes haben wir ein Leaflet Objekt vom Typ GeoJSON erstellt und diesem unseren GeoJSON Code, also die Variable geojsonFeature1, übergeben. Das GeoJSON Objekt haben wir gleichzeitig mithilfe der Methode addTo() zum Leaflet Kartenobjekt hinzugefügt.

Mehr mussten wir nicht tun! Das Ergebnis ist ein Standard Marker an der Stelle auf der Erde, die das GeoJSON Point Element beschreibt.

Das Objekt GeoJSON ist ein Leaflet Layer. Ganz konkret erweitert die Klasse GeoJSON die Klasse FeatureGroup.

Auf der nachfolgenden Abbildung können Sie sich das Ergebnis ansehen.



Wenn Sie an die Zeile

```
L.geoJSON(geojsonFeature1).addTo(mymap);
```

noch den Text

```
.bindPopup('Pop-up Text');
```

anhängen, also

```
L.geoJSON(geojsonFeature1).addTo(mymap).bindPopup('Pop-up Text');
```

schreiben, öffnet sich ein Pop-up Fenster, wenn Sie den Marker anklicken.

# Eine GeoJSON FeatureCollection in Leaflet einbinden

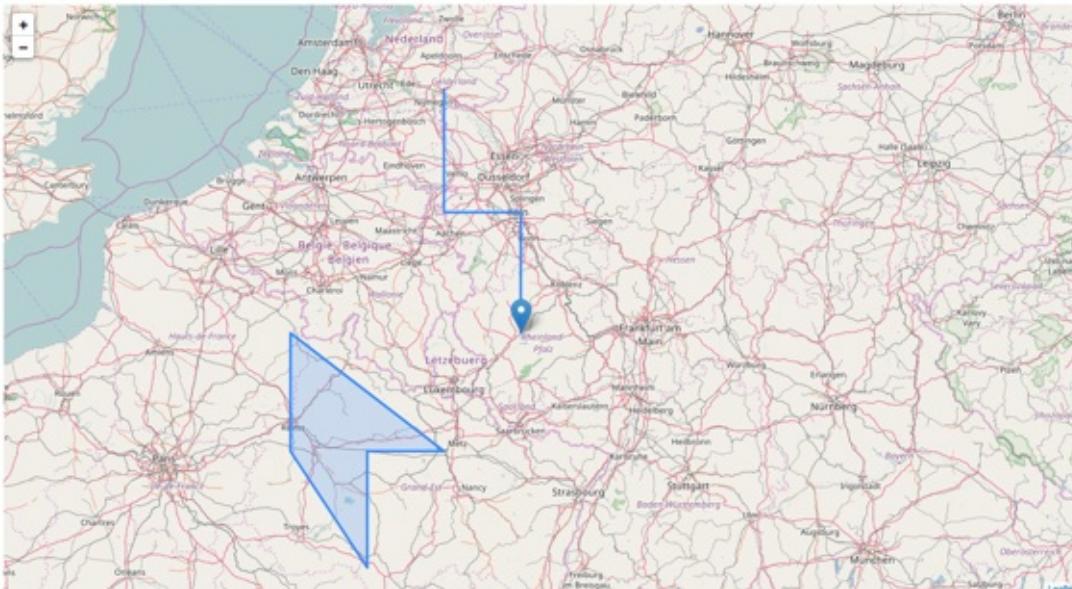
Das Beispiel im letzten Kapitel enthielt ausschließlich einen Punkt – also ein Feature. Geodaten bestehen in der Regel aus mehreren Geometrien mit dazugehörigen Eigenschaften – also FeatureCollections. Leaflet liest eine FeatureCollection genauso ein, wie Sie es im letzten Beispiel anhand des einen Features gesehen haben. Das nächste Beispiel zeigt Ihnen, wie Sie einen Punkt, ein Polygon und eine Linie in einem Schritt auf Ihrer Karte anzeigen könnten.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": { "type": "Point", "coordinates": [7, 50]},
      "properties": { "prop0": "value0" }
    },
    {
      "type": "Feature",
      "geometry": { "type": "LineString", "coordinates": [
        [7, 50], [7, 51], [6, 51], [6, 52]
      ]},
      "properties": { "prop0": "value0", "prop1": 0.0 }
    },
    {
      "type": "Feature",
      "geometry": { "type": "Polygon", "coordinates": [
        [ [6, 49], [5, 49], [5, 48], [4, 49], [4, 50] ]
      ]},
      "properties": { "prop0": "value0", "prop1": {"this": "that"} }
    }
  ]
}
L.geoJSON(geojsonFeatureCollection).addTo(mymap);
</script>
</body>
</html>
<!--index_972.html--&gt;</pre>
```

Voila! Drei Elemente mit Standardeigenschaften auf der Leaflet Karte.



## GeoJSON aus Leaflet exportieren

So, und nun machen wir genau das Gegenteil. Ein jedes Leaflet Objekt, das wir uns im Kapitel Die Karte mit Daten bestücken angesehen haben, hat eine Leaflet Methode mit dem Namen `toGeoJSON()`. Und diese Methode tut genau das, was der Name schon vermuten lässt: Das übergebene Leaflet Objekt wird, in ein GeoJSON Objekt umgewandelt und ausgegeben. Sehen Sie sich im nächsten Beispiel die Anwendung der Methode `toGeoJSON()` an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var myMarker=L.marker([50.27264, 7.26469]);

var markerAsGeoJSON = myMarker.toGeoJSON();
var geoJsonLayer = L.geoJson().addTo(mymap);

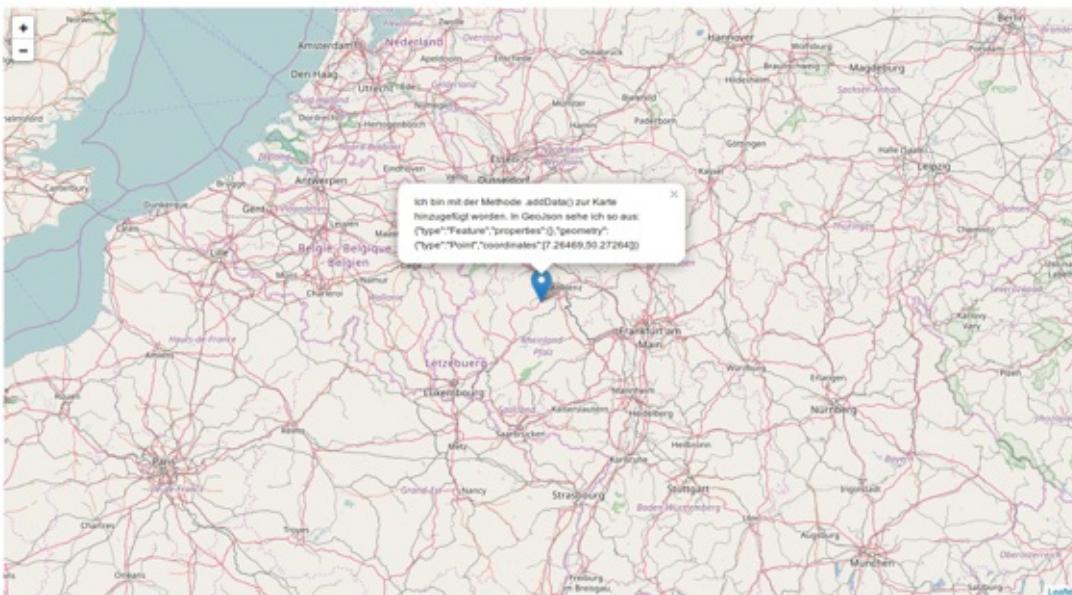
geoJsonLayer.addData(markerAsGeoJSON)
.bindPopup("Ich bin mit der Methode .addData()
zur Karte hinzugefügt worden. In GeoJson sehe ich so aus:<br> "
+ JSON.stringify(markerAsGeoJSON));

</script>
</body>
</html>
<!--index_971.html-->
```

Was zeigt Ihnen dieses Beispiel genau? Das folgende Beispiel zeigt Ihnen, wie Sie einen Marker ins GeoJSON Format konvertieren können. Dazu erstellen Sie zunächst mit `var myMarker=L.marker([50.27264, 7.26469])` einen Leaflet Marker. Danach rufen Sie die Methode `toGeoJSON()` des Markers auf und speichern das zurückgegebene GeoJSON Objekt in der Variablen `markerAsGeoJSON`. Als Nächstes erstellen Sie einen leeren GeoJSON Layer und fügen diesen zum Kartenobjekt hinzu: `var geoJsonLayer = L.geoJson().addTo(mymap)`. Sie hätten den GeoJSON Code in der Variablen `markerAsGeoJSON` wie in vorherigen Beispielen sofort beim Erstellen des Layers als Parameter mitgeben können. Hier wollte ich ihnen aber die Methode `addData()` zeigen, mit der Sie auch nachträglich noch GeoJSON Objekte zur GeoJSON Ebene hinzufügen können.

Im vorausgehenden Beispiel habe ich die Methode `JSON.stringify()` beim Erstellen des Textes im Pop-up Fenster angewandt. Mit der Methode `JSON.stringify()` können Sie eine JavaScript Variable in einen Text im JSON-Format konvertieren.

Die nachfolgende Abbildung zeigt das Bild, welches Sie im Browser sehen, wenn Sie die HTML-Datei des vorausgehenden Beispiels im Browser öffnen.



Sie werden nun sicher mein Beispiel als umständlich ansehen. Ich habe einen Marker, denn ich ohne zusätzliche Schritte auf der Karten hätte anzeigen könnte, vorher umgewandelt dann in umgewandelte Form zur Karte hinzugefügt. Diese Vorgehensweise ist nicht Sinn und Zweck der Leaflet Methode `toGeoJSON()`. Sinn und Zweck ist es eher, Daten der Karte zum Export anzubieten.

Wenn Sie möchten, dass die Methode `toGeoJSON()` Eigenschaften zu Ihren eigenen Leaflet Objekten exportiert, müssen Sie diese in einer bestimmten Form mit Ihrem Leaflet Objekt speichern. Möchten Sie beispielsweise ein Polyline Objekt exportieren, dann müssen Sie mit diesem Polyline Objekt eine Variable `feature` speichern. Die Variablen `feature` enthält den Text Feature in der Variablen `type` und die zu exportierenden Eigenschaften in der

Variablen properties.

```
var polyline = L.polyline([
[50.27264, 7.26469], [51.27264, 7.26469], [51.27264, 6.26469]
]);
polyline.feature = {};
polyline.feature.type = "Feature";
polyline.feature.properties = {};
polyline.feature.properties["Foo"] = "Bar";
```

Der Export des Polyline Objektes würde wie folgt aussehen:

```
{"type": "FeatureCollection",
"features": [
{
  "type": "Feature",
  "properties": {"Foo": "Bar"},
  "geometry": {"type": "LineString",
  "coordinates": [[7.26469, 50.27264], [7.26469, 51.27264], [6.26469, 51.27264]]}
}
]
```

## Stylen

Ein GeoJSON Layer bietet Ihnen mit der Methode `setStyle()` die Möglichkeit das Aussehen der Kartenschicht zu gestalten.

Sie können neben den hier beschriebenen Optionen auf eine große Auswahl weitere Stil Optionen zugreifen. Die vollständige Liste finden Sie in der Dokumentation von Leaflet. Sehen Sie sich dazu die Optionen zur Klasse Path an:  
<http://leafletjs.com/reference-1.1.0.html#path>.

### Beim Erstellen eines GeoJSON Layer einen Stil mitgeben

Der nachfolgende Programmcode zeigt Ihnen, wie Sie Stylesheets beim Erstellen eines GeoJSON Layer als Parameter mitgeben können. Im nachfolgenden Programmcode finden Sie eine Funktion, die je nach GeoJSON Objekt eine andere Farbe zurück gibt. Wenn es sich um den Typ LineString handelt, gibt die Funktion die Farbe Rot zurück, falls ein Polygon vorliegt, antwortet die Funktion mit Violett.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
```

```

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

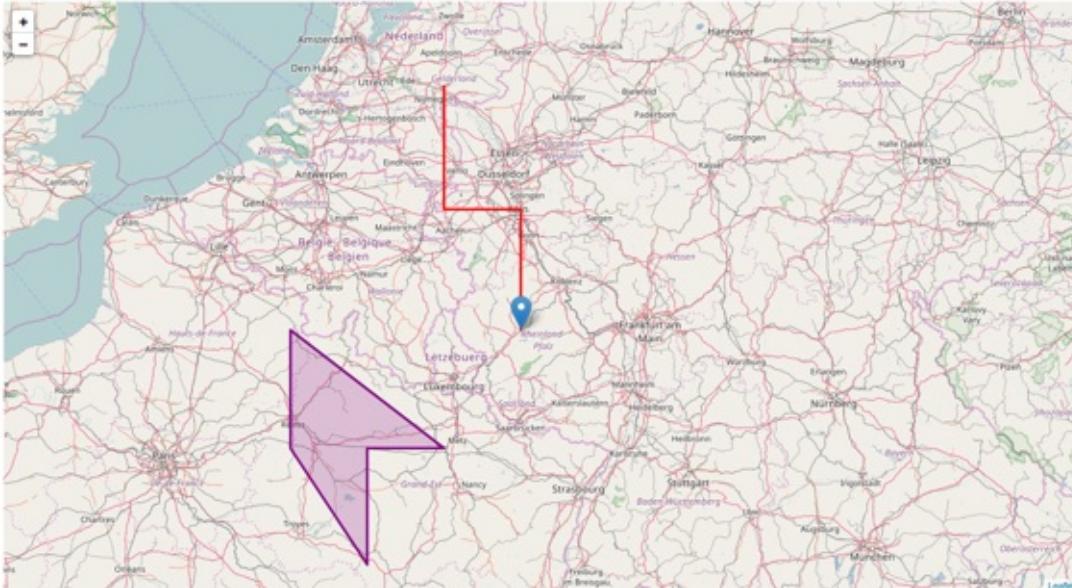
function styleFunction(feature)
{
switch (feature.geometry.type)
{
case 'LineString': return {color: "red"};
case 'Polygon': return {color: "purple"};
}
}
var geojsonFeatureCollection =
{
"type": "FeatureCollection",
"features":
[
{
"type": "Feature",
"geometry": {"type": "Point", "coordinates": [7, 50]},
"properties": {"prop0": "value0"}
},
{
"type": "Feature",
"geometry": { "type": "LineString", "coordinates": [
[7, 50], [7, 51], [6, 51], [6, 52]
]},
"properties": { "prop0": "value0", "prop1": 0.0 }
},
{
"type": "Feature",
"geometry": { "type": "Polygon", "coordinates": [
[ [6, 49], [5, 49], [5, 48], [4, 49], [4, 50] ]
]},
"properties": { "prop0": "value0", "prop1": {"this": "that"} }
}
]
}

var geoJsonLayer = L.geoJson(geojsonFeatureCollection,
{style:styleFunction}).addTo(mymap);
</script>
</body>
</html>
<!--index_970.html-->

```

Wenn Sie einen Stil auf alle Objekte anwenden möchten, dann ist es nicht notwendig eine Funktion zu erstellen. Die Zeile  
`var geoJsonLayer = L.geoJson(geojsonFeatureCollection,{color: "purple"}).addTo(mymap);`  
reicht völlig aus.

Auf der Karte sehen Sie nun die Objekte in der für sie bestimmten Farbe.



## Ein komplexeres Beispiel: Eine andere Farbe beim Überrollen mit der Maus

Im vorherigen Kapitel haben Sie gelernt, wie Sie mit der Option `style` des GeoJSON Layers ein Aussehen festlegen können. Sicherlich ändert sich das Aussehen Ihrer Objekte im Laufe der Zeit. Ganz häufig kommt es vor, dass man Objekte, die anklickbar sind und angeklickt wurden, als schon besucht kennzeichnen möchte. Oder Sie möchten ein Objekt, über dem sich die Maus gerade befindet, hervorheben. Genau diese beiden Anwendungsfälle sind Thema im nachfolgenden Beispielcode.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
function styleFunction(feature)
{
switch (feature.geometry.type)
{
case 'LineString': return {color: "red"};
case 'Polygon': return {color: "purple"};
}
}
var geojsonFeatureCollection =
{
"type": "FeatureCollection",
"features":
[
{
"type": "Feature",
"geometry": {"type": "Point", "coordinates": [7, 50]},
"properties": {"prop0": "value0"}
},
{

```

```

"type": "Feature",
"geometry": { "type": "LineString", "coordinates": [
[7, 50], [7, 51], [6, 51], [6, 52]
]},
"properties": { "prop0": "value0", "prop1": 0.0 }
},
{ "type": "Feature",
"geometry": { "type": "Polygon", "coordinates": [
[ [6, 49], [5, 49], [5, 48], [4, 49], [4, 50] ]
]
},
"properties": { "prop0": "value0", "prop1": {"this": "that"} }
}
]
}
}

var geoJsonLayer = L.geoJson(
geojsonFeatureCollection,{style:styleFunction})
.addTo(mymap);

geoJsonLayer.on('mouseover', styleWhenMouseOver);
geoJsonLayer.on('mouseout', styleWhenMouseOut);

function styleWhenMouseOver(e){
geoJsonLayer.setStyle({color:"green"});
}

function styleWhenMouseOut(e){
geoJsonLayer.eachLayer(function (layer) {
geoJsonLayer.resetStyle(layer);
});
}

</script>
</body>
</html>
<!--index_969.html-->

```

Auf den ersten Blick hat sich im Vergleich zum vorherigen Beispiel nichts geändert. Wenn Sie allerdings die Maus über ein Objekt bewegen, sehen Sie eine Änderung. Das Polygon und die Linie verfärben sich nun grün. Der Marker kann seine Farbe nicht ändern. Im Grunde genommen handelt sich bei ihm um ein Image. Das HTML-Element Image verfügt nicht über die CSS Eigenschaft color. Wenn Sie das Aussehen des Markers ändern möchten, dann müssten Sie diesem eine andere Bilddatei zuordnen.

Wenn Sie ein schon angeklicktes Element mit der Farbe Grau einfärben möchten, dann könnten Sie dies über die Funktion

```

function styleWhenMouseOut(e){
geoJsonLayer.setStyle({color:"gray"});
}
}
```

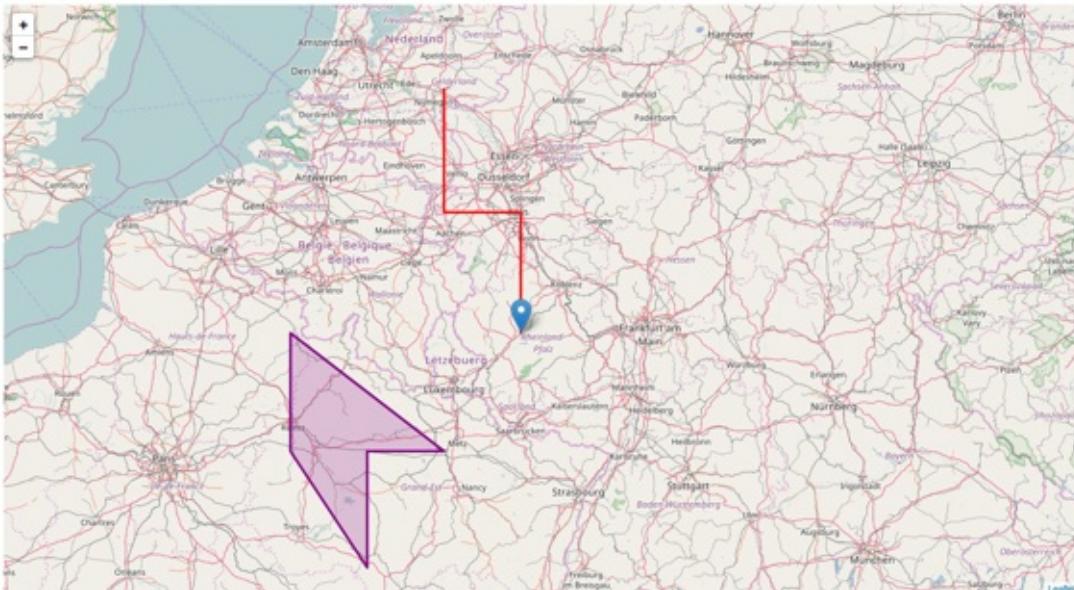
anstelle von

```

function styleWhenMouseOut(e){
geoJsonLayer.eachLayer(function (layer) {
geoJsonLayer.resetStyle(layer);
});
```

}

erreichen.



Wenn Sie

```
function styleWhenMouseOut(e){  
geoJsonLayer.eachLayer(function (layer) {  
geoJsonLayer.resetStyle(layer);  
});  
}
```

anstelle von

```
geoJsonLayer.on('mouseout',function(e){  
geoJsonLayer.resetStyle(e.layer);  
});
```

schreiben würden, würde nur ein Layer – nämlich der, der gerade überrollt wird – geändert.

## Iterieren

Interessant werden Karten, wenn Sie viele Informationen bieten. Eine Karte mit vielen Informationen setzt die Arbeit mit vielen Daten für den Kartenersteller voraus. Und, beim Arbeiten mit vielen Daten werden Sie die Möglichkeit, alle Features mit einem Schlag zu bearbeiten, zu schätzen lernen. Iterieren können sie in Leaflet durch GeoJSON Objekte mithilfe der Methode `onEachFeature()`.

### OnEachFeature() – Bearbeite jedes Feature

`OnEachFeature()` ist eine Methode, die einmal für jedes im Layer vorhandene GeoJSON Objekt vom Typ `Feature` aufgerufen wird. Nützlich ist diese Option zum

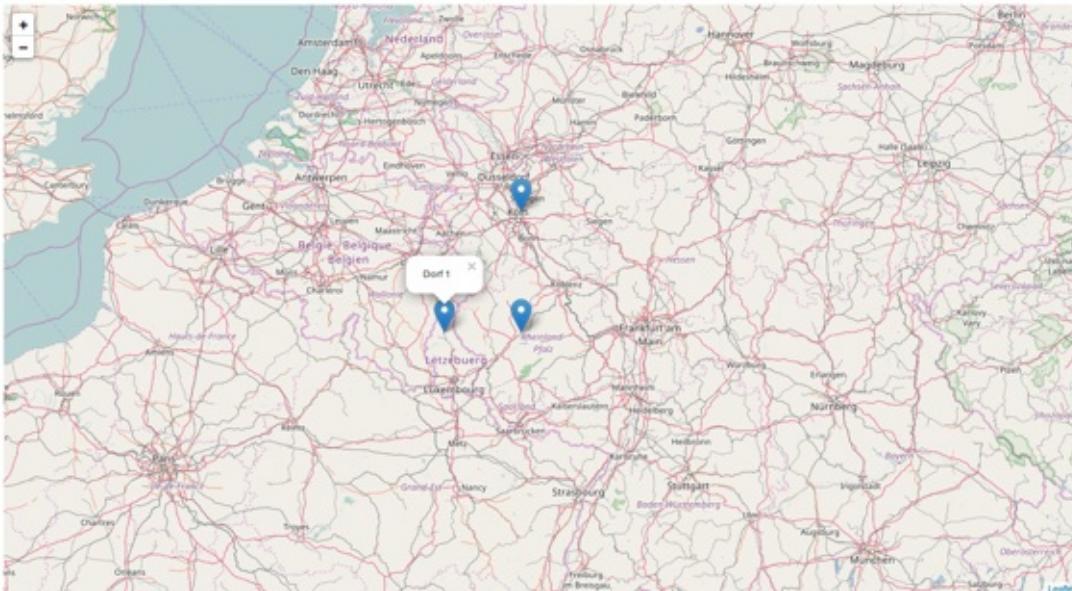
## Anhängen von Ereignissen oder Pop-up Fenstern an jedes Feature Objekt.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features": [
    [
      {
        "type": "Feature",
        "geometry": {"type": "Point", "coordinates": [6, 50]},
        "properties": {"name": "Dorf 1"}
      },
      {
        "type": "Feature",
        "geometry": {"type": "Point", "coordinates": [7, 50]},
        "properties": {"name": "Dorf 2"}
      },
      {
        "type": "Feature",
        "geometry": {"type": "Point", "coordinates": [7, 51]},
        "properties": {"name": "Dorf 3"}
      }
    ]
  ];
  L.geoJson(geojsonFeatureCollection,
  {
    onEachFeature: function(feature, layer)
    {layer.bindPopup(feature.properties.name);}
  }
).addTo(mymap);

</script>
</body>
</html>
<!--index_968.html-->
```

Der vorhergehende Programmcode zeigt Ihnen, wie Sie jedem Feature Objekt, das über die hart kodiert eingefügten GeoJSON Daten eingelesen wird, ein Pop-up anhängen können – und zwar jedem ein individuelles. Der Text für das Pop-up wird aus den GeoJSON Daten ausgelesen, er versteckt sich in der Eigenschaft `feature.properties.name`.

Das nachfolgende Bild stellt die drei Marker dar.



Nun würden wir dem Marker aber vielleicht auch gerne ein spezielles Aussehen geben. Vielleicht möchten Sie sogar jeden Marker unterschiedlich gestalten. `onEachFeature()` bietet Ihnen hierzu keine Möglichkeit. Der Marker wird automatisch mit Standardwerten erstellt. Leaflet bietet Ihnen eine andere Methode für diesen Zweck an. Die Methode heißt `pointToLayer()` und ein Beispiel dazu, wie Sie mit dieser Methode einen eigenen Marker erstellen und mit individuellen Optionen versehen können, finden Sie im nächsten Kapitel.

## PointToLayer – Punkt zu Ebene

Die Methode `pointToLayer`, die wir uns in diesem Kapitel ansehen, ist spezielle für die Arbeit mit einem GeoJSON Objekten vom Typ `Point` gemacht. Dieses Objekt ist das GeoJSON Pendant zum Marker. Wenn wir einen Point beim Erstellen des GeoJSON Layers als Parameter übergeben, dann wird ein Standard Marker erstellt. Wollen wir diesen Marker individuell gestalten, dann brauchen wir entweder einen Variablenamen, den wir ansprechen können, oder wir müssen den Marker selbst instanziieren. Für das Instanziieren brauchen wir eine Position oder einen Point. Und nun schließt sich der Kreis. Die Option `pointToLayer` gibt uns Zugriff auf die Koordinaten. Sehen Sie sich das nächste Beispiel an. Ein Beispiel erklärt oft mehr als viele Worte.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
```

```

var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features":
  [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [6, 50]},
      "properties": {"name": "Dorf 1"}
    },
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [7, 50]},
      "properties": {"name": "Dorf 2"}
    },
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [7, 51]},
      "properties": {"name": "Dorf 3"}
    }
  ]
};

var options_draggable = {
  draggable: true,
  title: "Ein Ort in der Nähe von Gering"
};

var options_notdraggable = {
  draggable: false,
  title: "Ein Ort in der Nähe von Gering"
};

L.geoJson(geojsonFeatureCollection, {
  pointToLayer: function(feature, latlng) {
    switch(feature.properties.name){

      case "Dorf 1":
        return L.marker(latlng, options_draggable)
        .bindPopup(feature.properties.name);

      case "Dorf 2":
        return L.marker(latlng, options_notdraggable)
        .bindPopup(feature.properties.name);

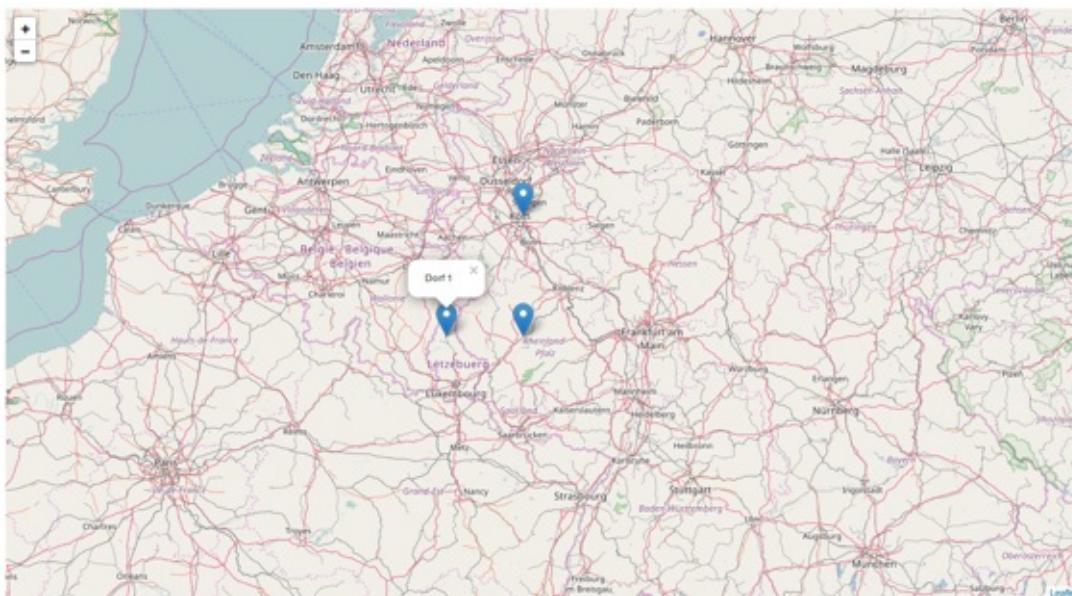
      case "Dorf 3":
        return L.marker(latlng, options_notdraggable)
        .bindPopup(feature.properties.name);

    }
  }
}).addTo(mymap);
</script>
</body>
</html>
<!--index_967.html-->

```

Im vorhergehenden Programmcodebeispiel sehen Sie, wie ein Marker erstellt und mit einem individuellen Pop-up Text versehen wird. Das Ergebnis ist auf den ersten Blick genau das Gleiche, wie im vorhergehenden Kapitel. Welcher Pop-up Text dem Marker genau zugewiesen wird, ist in dem Beispiel auch von der Eigenschaft `feature.properties.name` abhängig. Zusätzlich werden in diesem Beispiel Optionen von dem Namen abhängig gemacht. Nur der Marker von Dorf 1 kann auf der Karte verschoben werden. Der größte Unterschied ist aber, dass wir den Marker hier selbst erstellen und deshalb Einfluss auf die Optionen haben.

Sehen Sie sich die Karte, die Sie in der folgenden Abbildung sehen, in Ihrem Browser an. Auf den ersten Blick hat sich nichts zu dem Beispiel des vorherigen Kapitels geändert. Auf den zweiten Blick werden Sie feststellen, dass Sie nur den Marker von Dorf 1 auf der Karte verschieben können.



## Filtern mit der Option filter

Mithilfe der Option `filter` können Sie große Datenbestände auf das Wesentliche beschränken. Sehen Sie sich dies *im Kleinen* im nächsten Beispiel an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [6, 50]},
      "properties": {"name": "Dorf 1"}
    },
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [7, 50]},
      "properties": {"name": "Dorf 2"}
    }
  ]
}</script>
```

```

},
{
  "type": "Feature",
  "geometry": {"type": "Point", "coordinates": [7, 51]},
  "properties": {"name": "Dorf 3"}
}
]
};

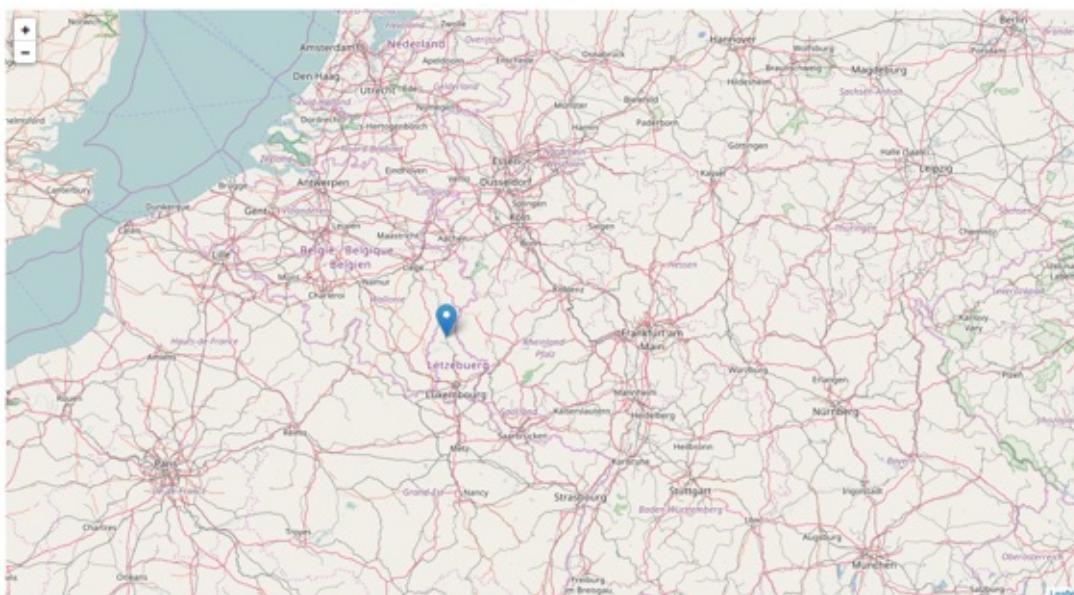
var options = {
draggable: true,
title: "Ein Ort in der Nähe von Gering"
};

L.geoJson(geojsonFeatureCollection, {
filter: function(feature, latlng) {
switch (feature.properties.name) {
case "Dorf 1": return true;
default: return false;
}
}
}).addTo(mymap);
</script>
</body>
</html>
<!--index_966.html-->

```

Im Beispiel sehen Sie, dass auf einem GeoJSON Layer nur die Elemente angezeigt werden, deren Rückgabewert beim Filtern positiv oder true ist.

Sie nachfolgende Abbildung zeigt es Ihnen: Nur Dorf 1 wird angezeigt. Die beiden anderen Orte, die sich in den GeoJSON Daten befinden, werden heraus gefiltert. Sie sehen nur einen Marker.



## In diesem Kapitel haben wir ...

In diesem Kapitel haben wir uns das Format GeoJSON genau angesehen. Wir haben die GeoJSON Typen mit den Objekten die wir mit Leaflet erstellen können, verglichen. Außerdem haben wir viele der Methoden und Optionen, die Leaflet zum

Arbeiten mit GeoJSON bietet, angewandt. Eine Karte mit Daten füllen können wir nun. Im nächsten Kapitel sehen wir uns an, wie wir mit den Daten auch Aussagen treffen oder Fragen beantworten können.

# Heatmaps und Choroplethenkarten

In den vorherigen Kapiteln haben wir uns angesehen, wie Sie Elemente direkt – oder mithilfe einer in GeoJSON formatierten Daten – zu Ihrer Karte hinzugefügt können. Jetzt geht es darum, dieses Wissen in die Tat umzusetzen und eigene Vorstellungen umzusetzen.

## In diesem Kapitel werden wir ...

In diesem Kapitel werden wir nicht nur einfach Elemente auf die Karte zeichnen. Wir werden mit der Karte Informationen weitergeben und Fragen beantworten. Insbesondere statistische Daten können auf Karten viel besser veranschaulicht werden, als in einer trockenen Tabelle – und nebenbei macht es sogar Spaß eine solche Karte zu erkunden.

Zwei Typen von Karten – nämlich Heatmaps und Choroplethenkarten – sehen wir uns nun in diesem Kapitel näher an. Beginnen wir mit der Heatmap.

## Heatmaps

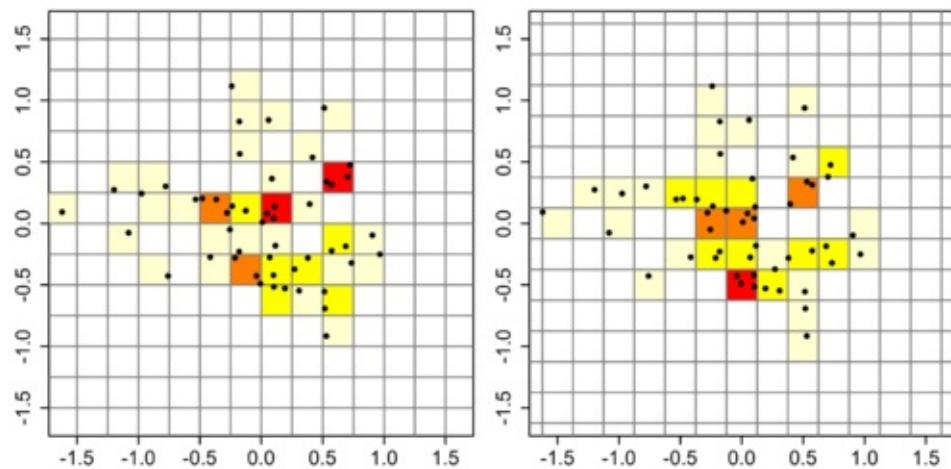
Heatmaps kennen wir im Deutschen auch unter dem Namen Wärmebild.

### Was ist eine Heatmap?

Eine Heatmap ist im Grunde genommen ein Diagramm, mit dem Daten visualisiert werden. Diese Visualisierung dient dazu, in einer großen Datenmenge intuitiv und schnell einen Überblick zu bekommen. In der grafischen Darstellung kristallisieren sich besonders markante Werte oft schnell heraus.

Heatmaps färben Bereich unterschiedlich, wenn die *Intensität* oder die *Dichte* des untersuchten Objektes unterschiedlich ist.

Möchten Sie gerne wissen, wie die Darstellung einer Heatmap technisch umgesetzt wird? Vereinfacht ausgedrückt wird zunächst ein Gitternetz über die Karte gelegt. Im nächsten Schritt wird die Anzahl der Punkte in jedem Bereich des Gitternetzes gezählt. Je nach Punktanzahl wird zum Schluss die Anzeige angepasst. Ist in einem Bereich kein Punkt vorhanden, wird dieser Bereich nicht mit Farbe gefüllt. Bei den anderen Bereichen wird je nach Anzahl der Punkte die passende Farbe im Bereich eingeblendet. Dieses Verfahren nennt man Multivariate Kerndichte Schätzung (englisch: Multivariate kernel density estimation). Detaillierter können Sie dies beispielsweise bei Wikipedia nachlesen.



(By Drleft (Own work) [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (<http://www.gnu.org/copyleft/fdl.html>)], via Wikimedia Commons)

Meist werden bei geringer Intensität oder geringer Dichte kalte Farben verwendet. Bei einem hohen Aufkommen wird der Bereich mit warmen Farben eingefärbt. Dies erklärt auch den Namen Heatmap – der englische Begriff für Hitze ist *heat*. Blau gilt als kalte Farbe, Rot, Orange und Gelb gelten als warme Farben.

## Heatmaps in Leaflet – Dichte

Unser erstes Beispiel zeigt eine Heatmap, die die unterschiedliche Dichte von Punkten sichtbar macht. Die Funktionen zum Anzeigen einer Heatmap werden nicht im Standardumfang von Leaflet mitgeliefert. Es handelt sich hierbei um eine besondere Funktion, die im Normalfall nicht zur Darstellung einer digitalen Karte benötigt wird. Leaflet selbst konzentriert sich auf die Anwendungsfälle, die üblicherweise beim Anzeigen einer Karte benötigt werden – ist aber offen für Plugins. Eines dieser Plugins ist Leaflet.heat.

Weitere Plugins, mit denen Sie Wärmeabbildungen oder ähnliche Visualisierungen aus Vektordaten erstellen können, finden Sie auf der Website von Leaflet im Bereich Plugins: <http://leafletjs.com/plugins.html#heatmaps>

Nachfolgende finden Sie das erste Beispiel für diesen Themenbereich.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
```

```

var mymap = L.map('mapid').setView([50.1555 , 7.591838], 10);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var points = [
[50.1555 , 7.591838 , 0.2],
[50.0931 , 7.664177 , 0.2],
...
[50.088041 , 7.652033 , 0.2],
[50.088041 , 7.652033 , 0.2],
[50.088041 , 7.652033 , 0.2]];

var heat = L.heatLayer(
points,
{
minOpacity: 0.95,
blur: 15,
maxZoom: 15,
max: 1,
radius: 15,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);
</script>
</body>
</html>
<!--index_965.html-->
```

Was haben wir gemacht? Als Erstes haben wir die Skript-Datei zum Plugin mit `<script src="leaflet-heat.js"></script>` eingebunden. Ich hatte die Datei von der oben verlinkten Website kopiert und unter dem Namen `leaflet-heat.js` im gleichen Verzeichnis wie die Datei `index.html` abgelegt. Danach habe ich eine Variable mit dem Namen `points` vom Typ `Array` erstellt. In dieser Variablen habe ich 700 Koordinaten gespeichert. Zu guter Letzt habe ich ein Objekt der Klasse `L.heatLayer` instanziert und diesem die Koordinaten in der Variablen `points` und einige Optionen übergeben.

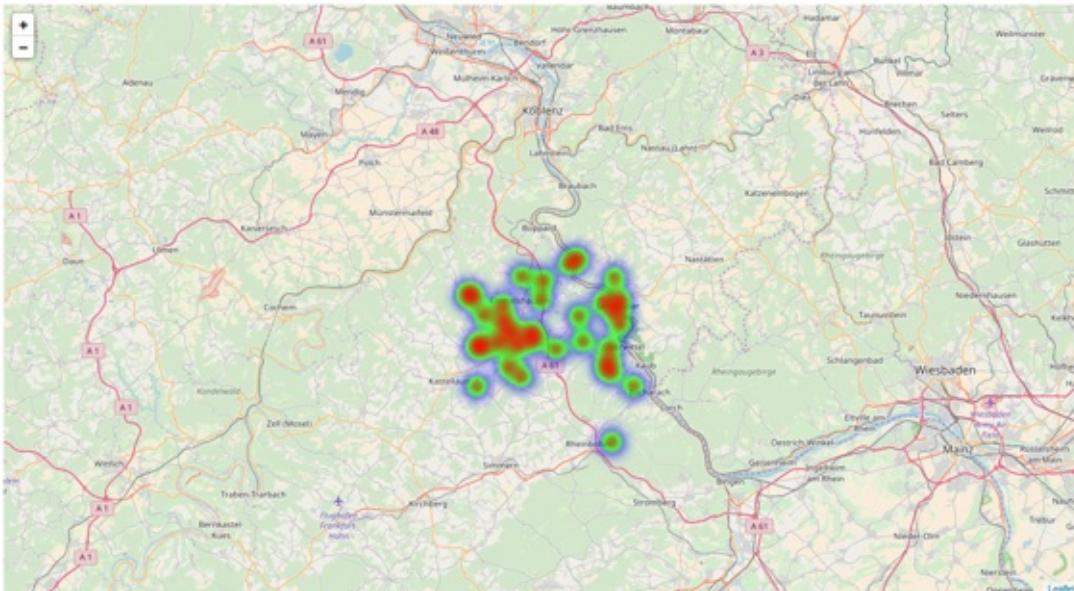
Der Dritte Wert bei der Übergabe der Koordinaten steht für die Intensität.

```

var points = [
[50.1555 , 7.591838 , 0.2],
...
...
```

Ich habe diesen Wert hier absichtlich immer mit der gleichen Zahl belegt. Ich nutze dieses Plugin nicht für die Visualisierung der Intensität. Hierfür zeige ich Ihnen im nächsten Kapitel eine andere Leaflet Erweiterung.

Das Ergebnis sehen Sie im nachfolgenden Bild. Da wir noch nichts gestaltet haben, sehen Sie die Standardansicht. Diese ist noch sehr rudimentär. Das geht besser, Sie werden sehen.



## Stile-Optionen

Das Plugin Leaflet.heat erlaubt es Ihnen Parameter zu übergeben. Dabei haben Sie folgende Möglichkeiten:

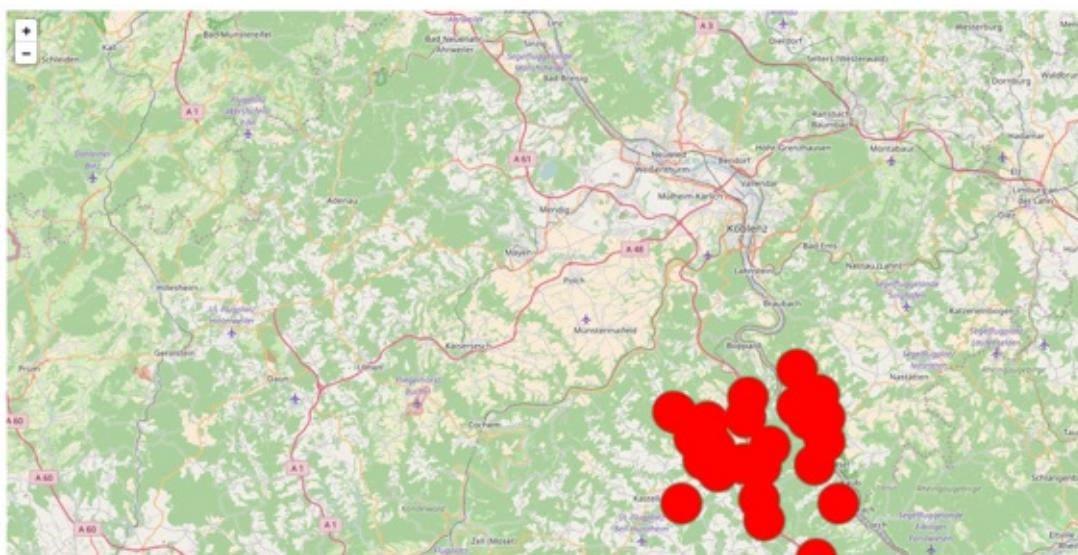
- **minOpacity:**  
Die Option `minOpacity` gibt die minimale Dichte an, ab der die Anzeige beginnt.
- **maxZoom:**  
Die Option `maxZoom` sollte auf die Zoomstufe gesetzt werden, bei der die Punkte die maximale Intensität erreichen. Als Standard wird der Wert, der für `maxZoom` in der Karte gesetzt ist, verwendet. Auf die Darstellung der Heatmap Ebene hat dies keine Auswirkungen. Sie sollten den Wert so setzen, dass Ihre Karte die beste Aussagekraft hat. Wenn der Wert zu hoch ist, kann es sein, dass die Punkte so weit auseinander liegen, dass der dichteste Bereich nicht mehr deutlich erkennbar ist. Setzen Sie den Wert zu niedrig an, kann man die Daten vielleicht nicht mehr zusammenhängend im Überblick sehen.
- **max:**  
Der Standardwert der maximalen Punktintensität ist 1.0.
- **radius:**  
Diese Option erklärt sich meiner Meinung nach von selbst. Mit dem Wert `radius` geben Sie die Größe an, in der die Punkte angezeigt werden sollen. 25 ist der Standardwert.
- **blur:**  
Mit der Option `blur` können Sie die Schärfe beeinflussen. Der Wert bestimmt, wie viele Punkte zusammen gefasst werden. Ein niedriger `blur` Wert erzeugt

einzelne Punkte, wohingegen ein hoher Wert Punkte zusammenfasst. Standardmäßig ist der Wert auf 15 festgesetzt.

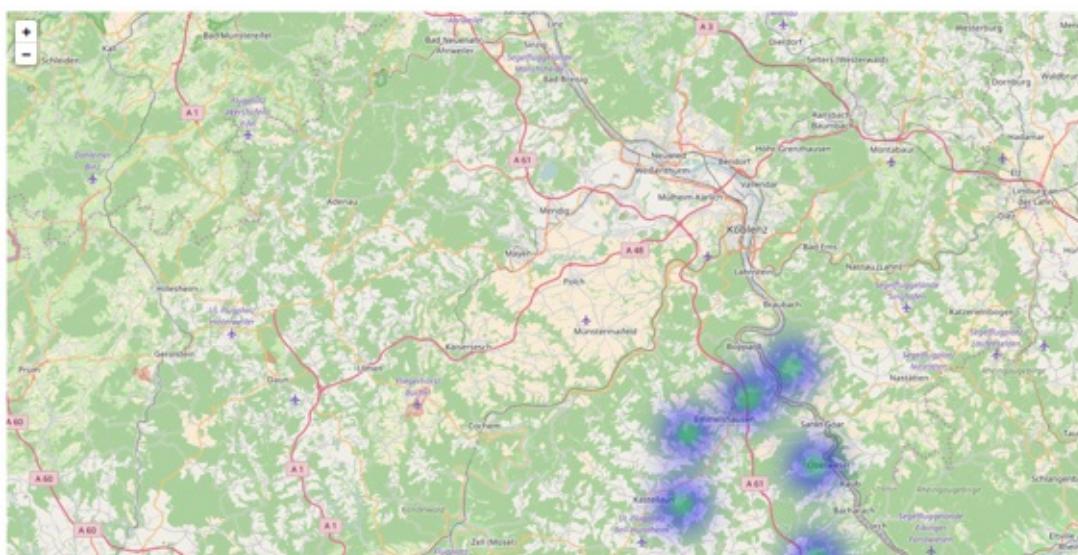
- **gradient:**

Die Option gradient steht für die Konfiguration des Farbverlaufs. Standardwert ist `{0.4: 'blue', 0.65: 'lime', 1: 'red'}`. Sie können beliebig viele Werte zwischen 0 und 1 zwischen Klammern angeben. Die Farbe, die am Rand angezeigt werden soll, sollten Sie mit dem niedrigen Wert angeben. Der Rand ist der Bereich in dem die Punkte nicht dicht beieinander sind. Die Farbe, die im Zentrum zu sehen sein soll, geben Sie idealerweise beim Wert 1 an. Das Zentrum ist der Bereich in dem die Punkte am dichtesten beieinander liegen.

**Nachfolgend sehen Sie einen Vergleich einer Ansichten mit unterschiedlichen Werten für die Optionen blur, gradient und maxZoom**



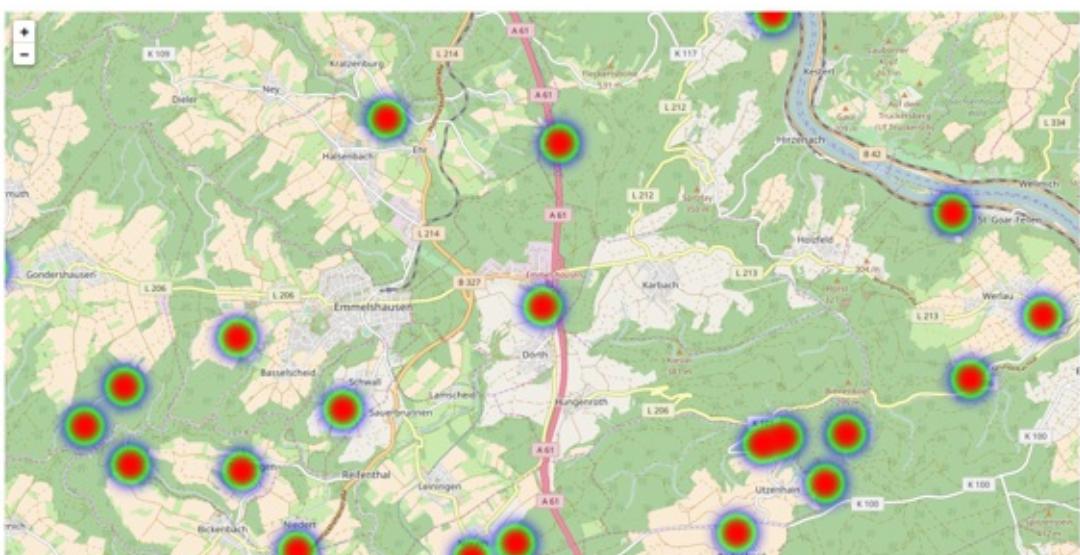
blur = 1



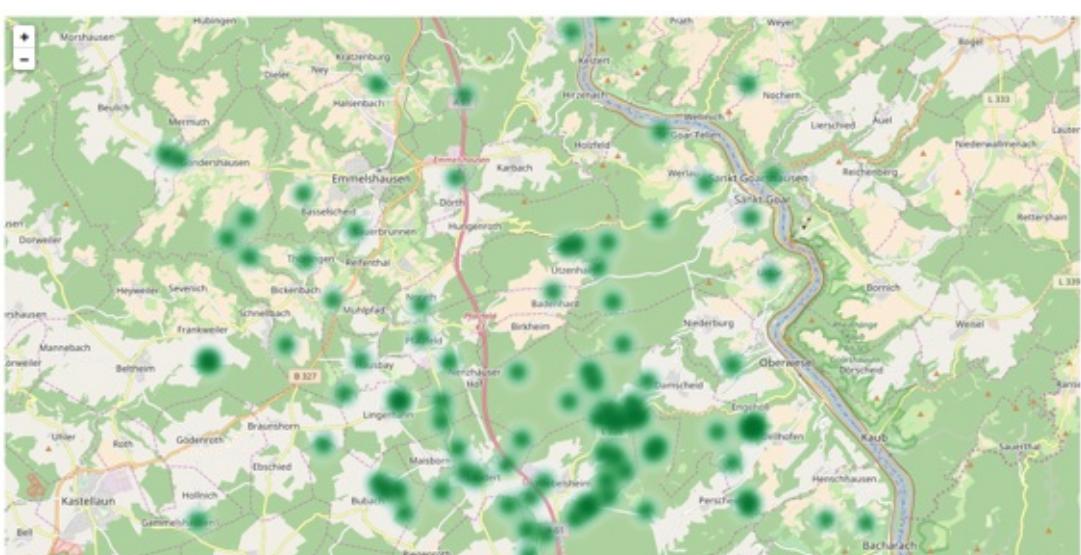
blur = 40



maxZoom = 6



maxZoom = 13



```
gradien {0.1: '#edf8fb', 0.2: '#ccece6', 0.3: '#99d8c9', 0.5: '#66c2a4', 0.7: '#2ca25f', 1: '#006d2c'}
```

Gefallen Ihnen die kalten und warmen Farben nicht? Möchten Sie lieber Ihre eigene

Farbzusammenstellung nutzen? Die Website <http://colorbrewer2.org> hilft beim Auswählen von Farben.

## Methoden

Zusätzlich zu den Stylesheet Optionen bietet Ihnen das Plugin `Leaflet.heat` vier Methoden:

- `setOptions(options)`:  
Diese Methode setzt die Optionen der Heatmap neu und zeichnet die Heatmap neu.
- `addLatLng(latlng)`:  
Diese Methode fügt einen Punkt zur Heatmap hinzu und aktualisiert die Ansicht gleichzeitig.
- `setLatLngs(latlngs)`:  
Diese Methode liest die Daten neu ein und aktualisiert danach die Ansicht.
- `redraw()`:  
Diese Methode zeichnet die Heatmap neu. Sie wird beim Ausführen der drei anderen Methoden automatisch ausgeführt, so dass Sie dies nicht selbst veranlassen müssen.

### Die Methode `addLatLng()`

Mithilfe der Methode `addLatLng()` können Sie nachträglich einen Punkt zum Heatmap Layer hinzufügen.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
var points = [
[50.1555, 7.591838, 0.2],
..
[50.188041, 7.662033, 0.2],
[50.88041, 7.52033, 0.2]
];
var heat = L.heatLayer(points,
{
minOpacity: 0.95,
```

```

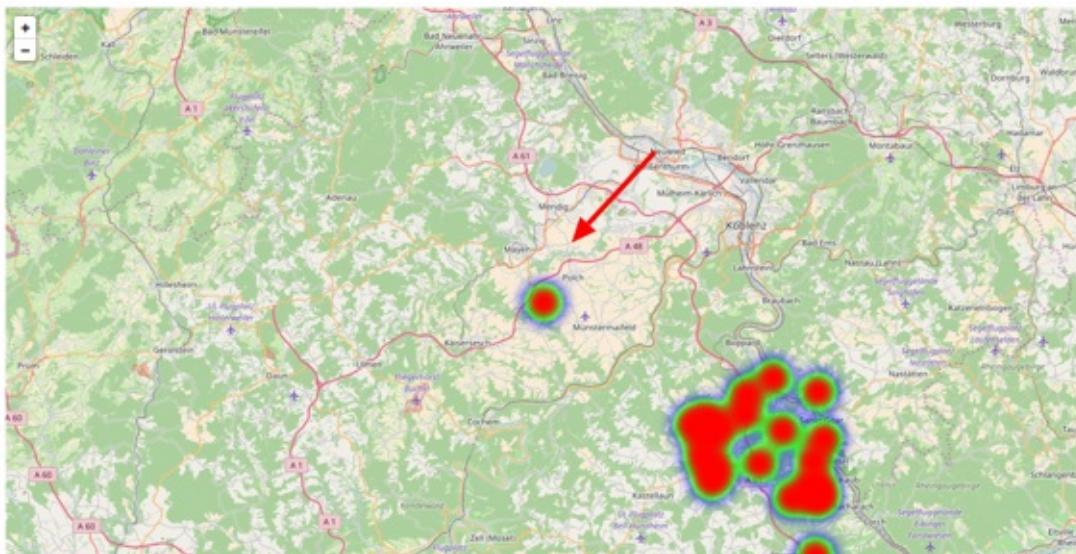
blur: 15,
maxZoom: 15,
max: 1.0,
radius: 25,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);

heat.addLatLng([50.27264, 7.26469, 0.2]);

</script>
</body>
</html>
<!--index_964.html-->

```

In der nächsten Abbildung können Sie den neu hinzugefügten Punkt erkennen.



Möchten Sie es Website Besuchern ermöglichen selbst Punkte zur Heatmap auf ihrer Karte hinzuzufügen? Eine Demo auf der Website zum Plugin zeigt genau solch ein Beispiel:  
<http://leaflet.github.io/Leaflet.heat/demo/draw.html>

### Die Methoden `addLatLng()`, `addLatLngs()` und `setOptions()` in einem Beispiel

In diesem Kapitel stelle ich Ihnen ein Beispiel vor, dass per Schaltfläche die verschiedenen Methoden anwendet.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>

<button onclick="add()">Punkt hinzufügen</button>
<br>
<button onclick="newPoint()">Daten neu setzen</button>
<br>

```

```

<button onclick="reset()">Reset</button>
<br><button onclick="setOptions()">Farben ändern</button>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 10);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var points =
[50.1555, 7.591838, 0.2],
...
[50.088041, 7.652033, 0.2],
[50.088041, 7.652033, 0.2]
];
var heat = L.heatLayer(points,
{
minOpacity: 0.95,
blur: 15,
maxZoom: 15,
max: 1.0,
radius: 25,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);
function add()
{
heat.addLatLng([50.1, 7.1, 0.2]);
}
function newPoint()
{
heat.setLatLngs([[50.5, 7, 0.2]]);
}
function reset()
{
heat.setLatLngs(points);
}
function setOptions()
{
heat.setOptions({
gradient: {0.4: 'black', 0.65: 'gray', 1: 'white'}
});
}
</script>
</body>
</html>
<!--index_963.html-->
```

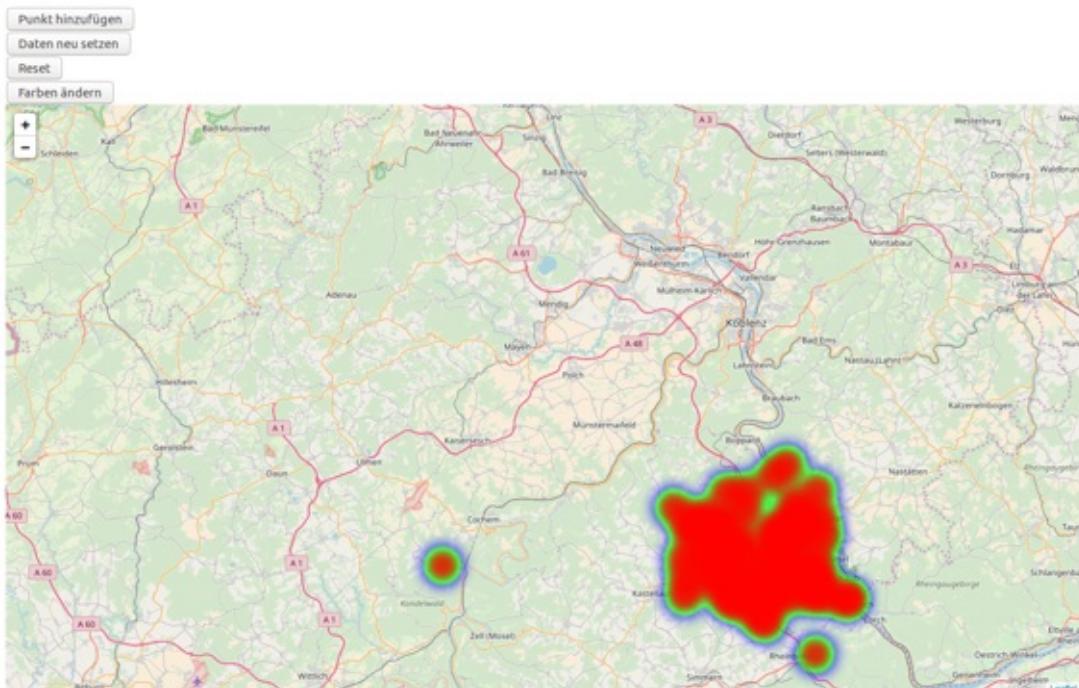
Was passiert genau in diesem Beispiel? Zunächst fügen wir vier Schaltflächen in das HTML-Dokument ein. Jede Schaltfläche führt eine der Methoden `add()`, `newPoint()`, `reset()` oder `setOptions()` aus, wenn sie angeklickt wird.

- Der Aufruf von `add()` führt den Code `heat.addLatLng([50.1, 7.1, 0.2]);` aus und fügt dabei den Punkt `[50.1, 7.1, 0.2]` zu den Daten der Heatmap Ebene hinzu.
- Die Methode `newPoint()` führt die Programmcodezeile `heat.setLatLngs([[50.5, 7, 0.2]]);` aus und setzt dabei die Daten der Heatmap Ebene neu. Die Heatmap Ebene enthält nun nur den einen Punkt `[[50.5, 7, 0.2]]` im Array.
- Die Methode `reset()` führt den Code `heat.setLatLngs(points);` aus und ersetzt damit die Daten der Heatmap Ebene wieder mit den Daten der Variablen

points. Achtung: Wenn Sie einen Punkt zum Layer hinzugefügt haben, als die Variable points Datenbestand des Layers war, dann ist dieser Punkt nun auch in der Variablen points enthalten!

- Die Methode setOptions() ist meiner Meinung nach selbsterklärend. Sie ändert einfach nur CSS Eigenschaften der Ebene.

Einen Screenshot der Karte zu diesem Beispiel sehen Sie in der nachfolgenden Abbildung.



## Marker

Im nächsten Beispiel zeige ich Ihnen, wie Sie zu jedem Punkt in einer Heatmap einen Marker hinzufügen können.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.1555 , 7.591838], 10);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var points = [
[50.1555 , 7.591838 ,
"<img src='http://lorempixel.com/200/200/'>"],
..
```

```

[50.088041 , 7.652033 ,
"<img src='http://lorempixel.com/200/200/'>"]
];

for(var i=0;i<points.length;i++) {
L.marker(
[parseFloat(points[i][0]),parseFloat(points[i][1])],
{opacity:0}).bindPopup(points[i][2],{keepInView:true})
.addTo(mymap);
}

var heat = L.heatLayer(
points,
{minOpacity: 0.95,
blur: 15,maxZoom: 15,
max: 1,radius: 15,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}
).addTo(mymap);

</script>
</body>
</html>
<!--index_962.html-->

```

Was ist diesem Beispiel anders? Wir haben der Variablen point einen dritten Wert mitgegeben. Dieser dritte Wert beinhaltet hier konkrete die Adresse zu einem Image. Dies hätte aber auch ein gewöhnlicher Text sein können. Diesen dritten Wert haben wir dann in der Methode bindPopup(points[i][2], {keepInView}) dazu genutzt, einen individuellen Marker zu kreieren.

So wie in der nächsten Abbildung könnte die fertige Karte auch bei Ihnen aussehen.



## Heatmaps mit Leaflet – heatmap.js – Intensität

Im vorhergehenden Kapitel haben wir eine Heatmap mithilfe des Plugins Leaflet.heat erstellt. Diese Heatmap hat die Dichte von Punkten auf einer Karte visualisiert. Wir haben bisher die Möglichkeit die Intensität zu visualisieren nicht genutzt. Ich stelle Ihnen lieber ein anderes Plugin für diesen Zweck vor. Als Nächstes möchte ich nun

mit Ihnen eine Heatmap, die die Intensität der Eigenschaft eines Punktes darstellt, mithilfe des Plugins heatmap.js erarbeiten. Herunterladen können Sie das Plugin unter der Adresse <https://www.patrick-wied.at/static/heatmapjs/plugin-leaflet-layer.html>. Hier finden Sie auch die Dokumentation zum Plugin.

Nachfolgende sehen Sie das erste Beispiel.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="heatmap.js"></script>
<script src="leaflet-heatmap.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var myData={
max: 100,data: [
{lat: 51.0934, lon:8.666819, value: 99},
..
{lat: 50.088041, lon:7.652033, value: 23},
{lat: 50.088041, lon:7.652033, value: 23}]
};

var baseLayer =
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png');

var cfg = {
"radius": .1,
"maxOpacity": .5,"scaleRadius": true,
"useLocalExtrema": true,
latField: 'lat',
lngField: 'lon',
valueField: 'value'
};

var heatmapLayer = new HeatmapOverlay(cfg);

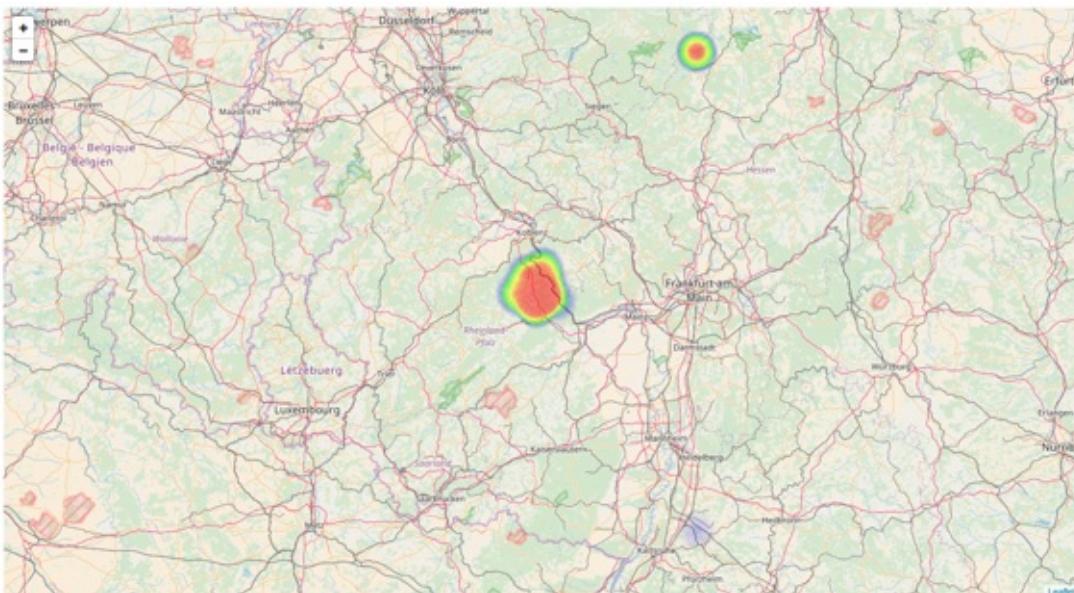
heatmapLayer.setData(myData);
var map = new L.Map('mapid', {
center: new L.LatLng(50.0586, 7.6568),
zoom: 8,layers: [baseLayer, heatmapLayer]
});

</script>
</body>
</html>
<!--index_961.html-->
```

Was zeigt Ihnen dieses Beispiel genau? Sie sehen, dass Sie neben dem Skript leaflet.heat.js auch das Skript heat.js einbinden müssen. Als Nächstes müssen Sie die Daten, die Sie visualisieren möchten, angeben – diese müssen eine Angabe zu dem maximal möglichen Wert enthalten: var myData={ max:100, data: [ {lat: 51.0934, lon:8.666819, value: 99} .... Danach können Sie die Heatmap Ebene erstellen. Der Übersicht halber habe ich die Optionen in einer

separaten Variablen, nämlich der Variablen `cfg`, definiert. Die Anweisung zum Erstellen der Heatmap Ebene mit den Optionen in der Variablen `cfg` lautet  
`var heatmapLayer = new HeatmapOverlay(cfg);`. Dieser Ebene können Sie nun die Daten mit dem Befehl `heatmapLayer.setData(myData);` hinzufügen. Zum Schluss fügen diesen Layer nun zum Kartenobjekt hinzu. Die Anweisung hierfür lautet: `L.Map('mapid', { center: new L.LatLng(50.0586, 7.6568), zoom: 8, layers: [baseLayer, heatmapLayer]});`.

Wie das aussehen sollte, können Sie sich in der nächsten Abbildung ansehen.



## Dokumentation und Methoden

Das nachfolgende Beispiel zeigt Ihnen, wie unterschiedlich Punkte mit unterschiedlicher Intensität aussehen können.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="heatmap.js"></script>
<script src="leaflet-heatmap.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var myData={
max: 100,
data: [
{lat: 51.0934, lon:8.666819, value: 99},
...
{lat: 50.088041, lon:7.652033, value: 23}]
};

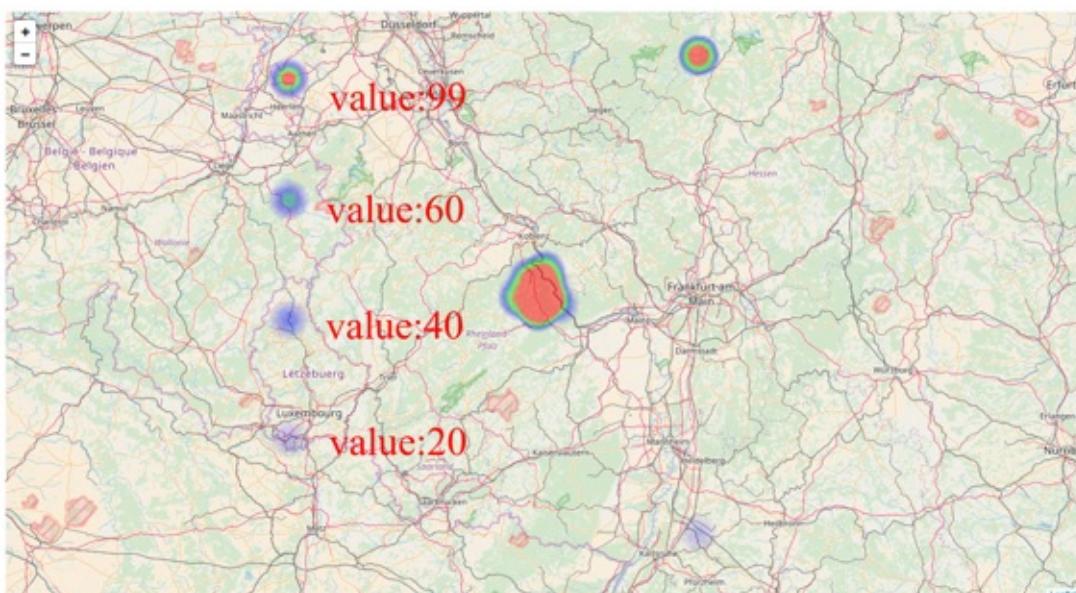
var baseLayer =
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png');
```

```

var cfg = {
  "radius": .1,
  "maxOpacity": .5,
  "scaleRadius": true,
  "useLocalExtrema": true,
  latField: 'lat',
  lngField: 'lon',
  valueField: 'value',
  gradient: {
    '.4': 'blue',
    '.8': 'lime',
    '.95': 'red'
  },
  blur: 0.75
};
var heatmapLayer = new HeatmapOverlay(cfg);
var map = new L.Map('mapid', {
  center: new L.LatLng(50.0586, 7.6568),
  zoom: 8,
  layers: [baseLayer, heatmapLayer]
});
heatmapLayer.setData(myData);
var test1 = {lat:51,lon:6,value:99};
var test2 = {lat:50.5,lon:6,value:60};
var test3 = {lat:50,lon:6,value:40};
var test4 = {lat:49.5,lon:6,value:20};
heatmapLayer.addData(test1);
heatmapLayer.addData(test2);
heatmapLayer.addData(test3);
heatmapLayer.addData(test4);
</script>
</body>
</html>
<!--index_960.html-->

```

Im vorhergehenden Codebeispiel haben wir 4 Punkte mit unterschiedlicher Intensität zur Karte hinzugefügt. Die Werte für die Intensität betragen 99, 60, 40 und 20 bei einem maximalen Wert von 100. In der nachfolgenden Abbildung sehen Sie die unterschiedliche Darstellung auf der Karte.



## Interaktive Heatmaps

Auch eine Heatmap kann interaktiv programmiert werden. Sie können es beispielsweise Benutzern ermöglichen, Daten zur Heatmap Ebene hinzuzufügen. Wie das geht sehen Sie im nachfolgenden Beispiel.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.1555, 7.591838], 15);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var points = [];
var heat = L.heatLayer(points,
{
blur: 15,
maxZoom: 15,
max: 1.0,
radius: 25,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);

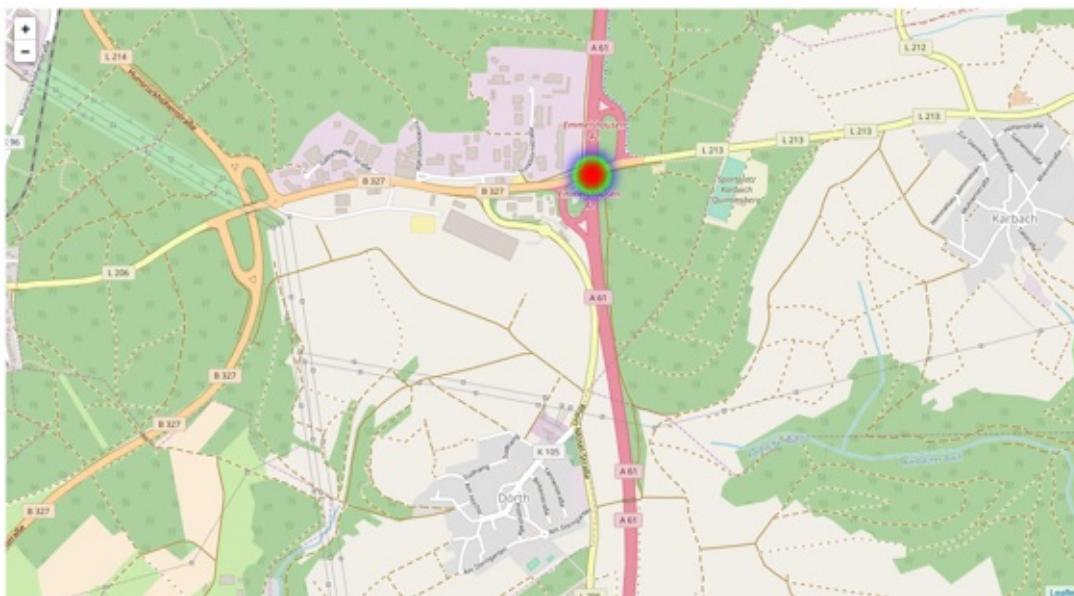
function addpoint(e){
heat.addLatLng(e.latlng);
}

mymap.on('click', addpoint);

</script>
</body>
</html>
<!--index_959.html-->
```

Was haben wir gemacht? Als Erstes haben wir mit der Zeile `<script src="leaflet-heat.js"></script>` das Plugin `leaflet.heat` eingebunden. In diesem konkreten Beispiel ist es wichtig, dass ein Doppelklick keine Änderung der Zoom-Stufen auf der Karte auslöst. So stellen wir sicher, dass jemand nicht versehentlich zu schnell klickt und ungewollt die Zoom-Stufe ändert, obwohl er eigentlich zwei Punkte hinzufügen möchte. Dies verhindern wir, indem wir die Option `doubleClickZoom` auf `false` setzen. Zu Beginn sollen noch keine Daten auf der Karte angezeigt werden. Deshalb haben wir zunächst einen leeren Datensatz mit `var points = [];` erstellt. Diesen leeren Datensatz geben wir einem `L.heatLayer` Objekt bei der Instanzierung als Parameter mit: `var heat = L.heatLayer(points, {...});`. Nun fehlt noch die Methode, mit der ein Punkt hinzugefügt wird. Diese Methode ist mit der Anweisung `function addpoint(e){ heat.addLatLng(e.latlng); }` schnell erstellt. Und `mymap.on('click', addpoint);` bewirkt, dass diese auch ausgeführt wird, wenn jemand auf die Karte klickt.

Das Ergebnis sehen Sie im Browser, wenn Sie die HTML Datei dieses Beispiels öffnen. Zunächst wird die Karte ganz normal angezeigt. Sie sehen keinen Wärmepunkt. Wenn Sie mit der Maus eine Stelle auf der Karte klicken, wird an dieser Koordinate ein Wärmepunkt hinzugefügt.



## Animierte Heatmaps

Das ist fast wie Kino – im nächsten Beispiel verändert sich die Karte automatisch.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>

<script src="leaflet-heat.js"></script>

</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.1555 , 7.591838], 10);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var points = [];
var points1 = [[50.1555 , 7.591838 ],...];
var points2 = [[50.1555 , 7.591838 ],...];
var points3 = [[50.1555 , 7.591838 ],...];
var heat = L.heatLayer(points).addTo(mymap);

x=1;
var name='';

var interval = setInterval(function(){run()},2000);

</script>
```

```

function run(){
mymap.removeLayer(heat);
name="points"+x.toString();

heat = L.heatLayer(window[name],{
blur:15,
maxZoom:10,
radius:25,
gradient:{0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);

if (x == 3) {
x=1;
} else {
x++;
}

}

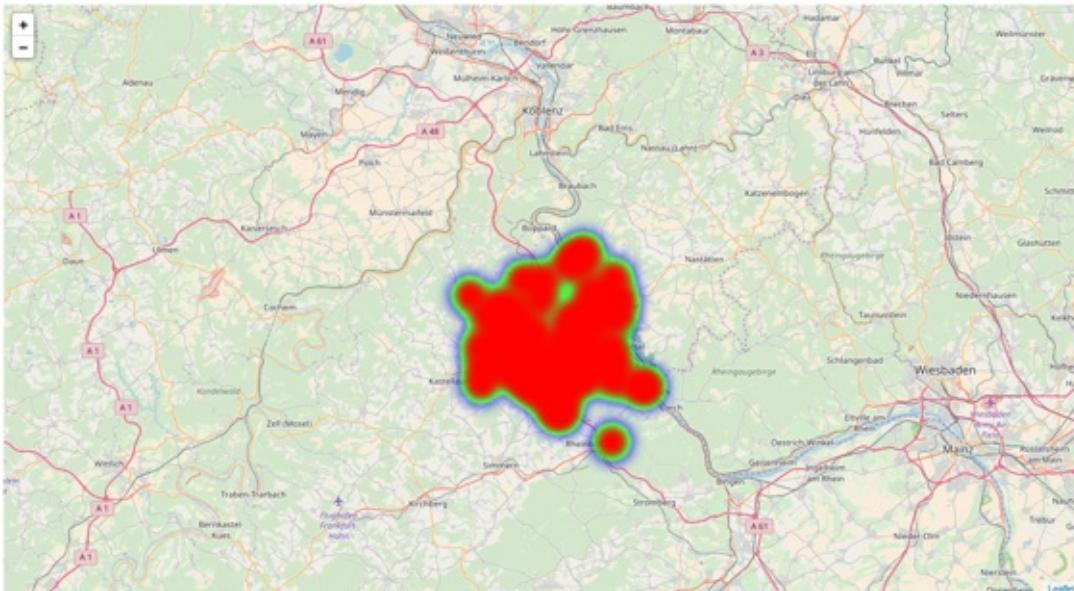
</script>
</body>
</html>
<!--index_958.html>

```

Was passiert im Beispiel genau? Als Erstes integrieren wir wieder das notwendige Skript. Danach erstellen wir die vier Arrays und nennen diese points, points1, points2 und points3. Jeder Array enthält unterschiedliche Daten. Mit dem ersten leeren Array erstellen wir die Schicht mit den Daten für die Heatmap - also das Objekt des Typs heatLayer. Die Hilfsvariable x in Verbindung mit der Methode setInterval() ermöglicht es uns dann, die Heatmap Ebenen fortlaufend mit anderen Daten anzeigen.

In diesem Beispiel haben ich zwei JavaScript Elemente verwendet, die vielleicht erkläungsbedürftig sind. Als Erstes habe ich den Variablenamen mit window[name] zusammengesetzt. Was bedeutet dies genau? Da es sich bei um Javascript Objekte handelt, wird jede Variable in einem globalen Objekt gespeichert. Wenn Sie also die Variable points1 im globalen Bereich initialisieren – also nicht in einem Funktionskontext –, schreiben Sie diese Variablen implizit in ein globales Objekt. In einem Browser ist dies das Objekt window. Der Wert dieser Variablen kann mit der Punkt Notation oder der Klammer Notation abgerufen werden. Also entweder mit var name = window.points1; oder mit var name = window['points1']; Außerdem haben wir die Methode setInterval() eingesetzt. Mit dieser Methode können Sie eine Funktion wiederholt aufrufen. Hierbei können Sie ein Intervall zwischen den einzelnen Aufrufen definieren.

So wie im nachfolgenden Bild sieht die Karte nur alle 6 Sekunden aus. In der Zwischenzeit wechselt die Ansicht zweimal.



## Choroplethenkarte

Im vorherigen Kapitel haben wir eine Heatmap zum visualisieren von Daten verwendet. Wir haben zunächst mithilfe des Skripts Leaflet.heat Bereiche, in denen Punkte dichter vorkommen, farblich hervorgehoben. Dann haben wir zusätzlich das Skript heat.js geladen und Bereiche, in denen die Punkte eine hohe Intensität haben, farblich besonders markiert. Eine Choroplethenkarte macht erst einmal nichts anderes. Sie visualisiert die Dichte oder die Intensität bestimmter Objekte.

### Was genau ist eine Choroplethenkarte?

Ich hatte Ihnen erklärt, dass eine Heatmap ein Raster über die Karte legt. Je nachdem wie die zu visualisierenden Punkte in diesem Raster verteilt sind, werden Farben sichtbar. Eine Choroplethenkarte verwendet kein separates Raster. Sie verwendet ein Polygon. Ein Polygon kann zum Beispiel ein Land oder das Einzugsgebiet eines Unternehmens sein. Im Kapitel *Die Karte mit Daten bestücken* hatte ich die Besonderheit eines Polygons beschrieben. Dieses Vieleck hat eine Grenzlinie die einen Innenbereich und einen Außenbereich voneinander abgrenzt. Eine populäre Choroplethenkarte, die Sie sicherlich schon einmal gesehen haben, ist die Darstellung der Bevölkerungsdichte eines Gebietes auf der Erde. Ich habe hier ein Beispiel erstellt, welches genau dies tut. Ich zeige Ihnen, wie Sie eine Karte, die die Bevölkerungsverteilung in Rheinland-Pfalz grafisch darstellt, selbst erstellen können.

### Choroplethenkarten in Leaflet

Das Schöne ist, dass wir mit Leaflet keine zusätzlichen Plugins für das Erstellen einer Choroplethenkarte benötigen. Leaflet ist wie dafür gemacht, GeoJSON Daten als Choroplethenkarte anzuzeigen. Beginnen tun wir ganz vorne mit dem Klären der

Frage: Wo bekommen Sie die Daten her?

## Open Data

Wenn Sie nicht selbst über Daten verfügen, können sie auf jede Menge offene Data zugreifen.

Open Data, also offene Daten, sind Daten, die von jedem ohne jegliche Einschränkungen verwendet und weitergegeben werden dürfen. Warum gibt es Open Data? Viele Menschen vertreten die Meinung, dass frei nutzbare Daten zu mehr Transparenz und Zusammenarbeit führen. Die Bereitstellung offener Daten durch öffentliche Einrichtungen wird als eine Voraussetzung für Open Government, also der Öffnung von Regierung und Verwaltung gegenüber der Bevölkerung und der Wirtschaft, angesehen. Die Befürworter von Open-Data teilen somit viele Argumente mit den Befürwortern von Open-Source. Open Data und Leaflet als Open Source Software passen gut zusammen.

GeoJSON Utilities ist ein Projekt, welches den Export von Gemeindeflächen, Landkreisflächen und Bundeslandflächen in Deutschland im GeoJSON Format ermöglicht. Jede exportierte Fläche enthält zusätzliche Eigenschaften wie die Einwohnerzahl und die Größe der Fläche in Quadratmetern. Ich habe diese Daten für Rheinland-Pfalz in eine GeoJSON-Datei exportiert. Im nächsten Beispiel zeige ich Ihnen, wie Sie mithilfe dieser GeoJSON-Datei eine Chorophletenkarte erstellen können. GeoJSON Utilities ist übrigens auch eine mit Leaflet erstellte Anwendung.

Die GeoJSON-Datei, die ich heruntergeladen haben, ist relativ groß. Deshalb habe ich die Daten in einer separaten Datei abgelegt. Ich schlage Ihnen vor, dies auch zu tun. So bleibt Ihre HTML-Datei übersichtlich und Sie können sich voll und ganz auf das Erstellen der Karte konzentrieren. Wenn Sie die Daten in einer JavaScript Datei ablegen, können Sie diese gleichzeitig als Variabel deklarieren. Wie das geht, zeigt ihnen das nachfolgende Beispiel. Zunächst sehen Sie die Datei, die die GeoJSON Daten enthält. Dieser Datei habe ich den Namen `gemeinden.js` gegeben. Die GeoJSON Daten werden in der Datei `gemeinden.js` in die Variable `ct` geladen.

```
var ct =
{
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {"name": "urn:ogc:def:crs:OGC:1.3:CRS84"}},
    "source": "© GeoBasis-DE / BKG 2013 (Daten verändert)",
    "features": [
      {"type": "Feature",
      "properties": {
        "ADE": 6,
        "GF": 4,
        "BSG": 1,
        "RS": "073345001001",
        "AGS": "07334001", "SDV_RS": "073345001001",
        "GEN": "Bellheim",
```

```

"BEZ":"Gemeinde",
"IBZ":64,
"BEM":"gemeinschaftsangehörig",
"NBD":"ja",
"SN_L":"07",
"SN_R":"3",
...
"destatis":{
  "RS":"073345001001",
  "area":20.44,
  "area_date":"31.12.2014",
  "population":8519,
  "population_m":4198,
  "population_w":4321,
  "population_density":417,
  "zip":"76756",
  "center_lon":"8,284042",
  ...
},
"geometry":{
  "type":"Polygon",
  "coordinates": [[[8.276302148832034,49.21145214735215],
  [8.295112386392997,49.21205183793759],
  ...
  ]]}},
{"type":"Feature",
"properties":{"ADE":6,
"GF":4,
"BSG":1,
"RS":"073405003205",
"AGS":"07340205",
"SDV_RS":"073405003205",
...
}

```

In den Properties der einzelnen GeoJSON Features sind unter anderem der Regionalschlüssel (RS), der geographische Name (GEN), die amtliche Bezeichnung der Verwaltungseinheit (BEZ) und die Destatis-Daten – insbesondere die Werte für die Größe der Fläche in Quadratmetern (area) und die Einwohnerzahl (population) – enthalten. Das Beispiel zeigt nur einen Ausschnitt der Datei `gemeinden.js`. Beachten Sie, dass es sich nicht um reines GeoJSON handelt. Ganz genau handelt es sich um JavaScript. Dieser JavaScript Code deklariert eine Variable, nämlich die Variable `ct`. Einbinden können Sie die JavaScript Datei genau wie jedes andere Skript in Ihre HTML-Datei mit der Zeile

```
<script src="gemeinden.js"></script>
```

Wenn die Datei `gemeinden.js` richtig eingebunden ist, können Sie auf die GeoJSON Daten in dieser Datei ab nun über die Variable `ct` zugreifen.

## Farben

Der nächste Schritt zum Fertigstellen der Choroplethenkarte ist die Farbauswahl. Beim Erstellen der Heatmaps im vorhergehenden Kapitel haben wir Farben übergeben und das jeweilige Plugin hat diese entsprechend zugewiesen. Beim Erstellen der Choroplethenkarte müssen wir uns selbst um diese Aufgabe kümmern.

Deshalb definieren wir eine Funktion, die je nach Wert eine bestimmte Farbe errechnet und zurück gibt. Idealerweise kennen Sie den höchsten Wert und den niedrigsten Wert in der Datenmenge. Dann können Sie, je nach Verteilung der Werte, bestimmten Wertbereichen eine Farbe zuordnen.

Der nachfolgende Programmcodeausschnitt nimmt den Parameter `x` entgegen und berechnet anhand des Wertes dieses Parameters eine Farbe. Die Funktion `color(x)` gibt bei einem höheren `x` Werte eine dunklere Farben aus. Bei einem niedrigeren `x` Wert gibt die Funktion eine hellere Farben zurück.

```
function color(x) {
  return
  x > 1000 ? '#990000' :
  x > 750 ? '#d7301f' :
  x > 500 ? '#ef6548' :
  x > 250 ? '#fc8d59' :
  x > 200 ? '#fdbb84' :
  x > 100 ? '#fdd49e' :
  x > 0 ? '#fee8c8' : '#ffff7ec';
}
```

## Stile

Nun benötigen wir noch eine Funktion, die das Zuweisen des passenden CSS Stylesheets zu den Daten übernimmt. Nur so können wir jedes GeoJSON Feature individuell ansehen und ihm den passenden Stil zuweisen. Im nachfolgenden Programmcodeausschnitt sehen Sie eine Funktion, die als Parameter ein GeoJSON Feature erwartet. Je nach Eigenschaft im Feature gibt die Funktion die passenden Optionen zurück. Sie erkennen sicherlich sofort, dass die wesentliche Option die mit dem Namen `fillColor` ist. Hier ist genau die Stelle, an der wir die eben erstellte Funktion `color()` aufrufen und dieser als Parameter den Wert `feature.properties.destatis.population`, also die Bevölkerungsanzahl, mitgeben.

```
function myStyle(feature) {
  return {
    fillColor: color(feature.properties.destatis.population),
    weight: 1,
    opacity: 1,
    color: 'white',
    fillOpacity: 0.85
  };
}
```

## Das vollständige Beispiel

Nachfolgenden habe ich die gerade betrachteten Codeschnipsel in einem vollständigen Beispiel zusammengesetzt.

```
<!DOCTYPE HTML>
<html lang="de">
```

```

<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="gemeinden.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([49.9555 , 7.591838], 8);

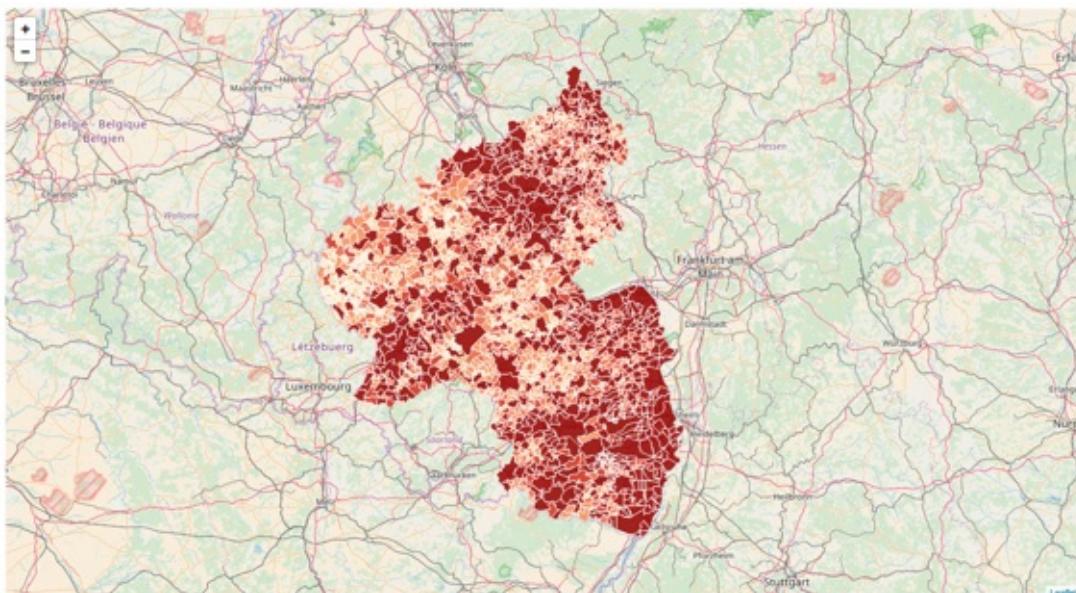
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
function color(x) {
return x > 1000 ? '#990000' :
x > 750 ? '#d7301f' :
x > 500 ? '#ef6548' :
x > 250 ? '#fc8d59' :
x > 200 ? '#fdbb84' :
x > 100 ? '#fdd49e' :
x > 0 ? '#fee8c8' : '#ffff7ec';
}
function myStyle(feature) {
return {
fillColor: color(feature.properties.destatis.population),
weight: 1,
opacity: 1,
color: 'white',
fillOpacity: 0.85
};
}

var geoJsonLayer = L.geoJson(ct, {style: myStyle})
.addTo(mymap);

</script>
</body>
</html>
<!--index_957.html-->

```

In Ihrem Browser sollte die Karte nun so wie in der nächsten Abbildung aussehen.



In diesem Beispiel haben wir einen festen Wert ganz unabhängig von anderen Werten verwendet. Wir haben die Bevölkerungszahl unabhängig von der Flächengröße der Gemeinde als Wert für die Farbe verwendet. In dieser Karte können wir zwar ablesen, wo die Gemeinden mit hoher Bevölkerungszahl sich befinden und wo die Gemeinden mit niedriger Bevölkerungsanzahl sind. Wie dicht die Besiedelung ist, sagt diese Karte aber nicht aus. Hierzu müssten wir die Bevölkerungszahl noch relativ zur Fläche der Gemeinde setzen. Dies tun wir nun im nächsten Beispiel.

## Normalisierte Choroplethenkarten

Es kommt sicherlich sehr oft vor, dass Sie mit einer Choroplethenkarten nicht nur einen absoluten Wert darstellen möchten. Oft möchten Sie die Daten zu anderen Daten ins Verhältnis stellen. Zum Beispiel könnten Sie den Anteil von Menschen, die älter als 60 Jahre sind darstellen wollen. Da Geodaten einen Bezug zu einer Fläche haben, wird es meist so sein, dass Sie die Daten in Relation zu dieser Fläche visualisieren möchten – diese also normalisieren möchten.

In unserer GeoJSON Datei ist die Fläche der Gemeinde in der Eigenschaft `area` gespeichert. Wir können es uns also einfach machen und auf die Zahl die für die Fläche einer Gemeinde gespeichert ist, zugreifen. Um die Choroplethenkarte, die wir im vorhergehenden Kapitel erstellt haben zu normalisieren, ist somit nur ein Schritt – nämlich das Teilen der Bevölkerungsanzahl durch die Fläche des betreffenden Bereichs – erforderlich. Sehen wir uns aber das ganze Beispiel noch einmal zusammenhängend an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="gemeinden.js"></script>
</head>
<body>
<button onclick="total()">Population</button>
<button onclick="density()">Population/Fläche</button>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([49.9555, 7.591838], 8);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
function color(x) {
return x > 1000 ? '#990000' :
x > 750 ? '#d7301f' :
x > 500 ? '#ef6548' :
x > 250 ? '#fc8d59' :
x > 200 ? '#fdbb84' :
x > 100 ? '#fdd49e' :
x > 0 ? '#fee8c8' : '#ffff7ec';
}
function densityColor(x) {
```

```

return x > 500 ? '#990000' :
x > 400 ? '#d7301f' :
x > 300 ? '#ef6548' :
x > 200 ? '#fc8d59' :
x > 100 ? '#fdbb84' :
x > 50 ? '#fdd49e' :
x > 0 ? '#fee8c8' : '#ffff7ec';
}
function myStyle(feature) {
return {
fillColor: color(feature.properties.destatis.population),
weight: 1,
opacity: 1,
color: 'white',
fillOpacity: 0.85
};
}
function myDensityStyle(feature) {
return {
fillColor: densityColor(
feature.properties.destatis.population
/feature.properties.destatis.area
),
weight: 1,
opacity: 1,
color: 'white',
fillOpacity: 0.85
};
}
function total(){

var geoJsonLayer = L.geoJson(ct, {style: myStyle})
.addTo(mymap);

removeLayer(densitylayer);
}
function density(){

var densitylayer=
L.geoJson(ct, {style: myDensityStyle})
.addTo(mymap);

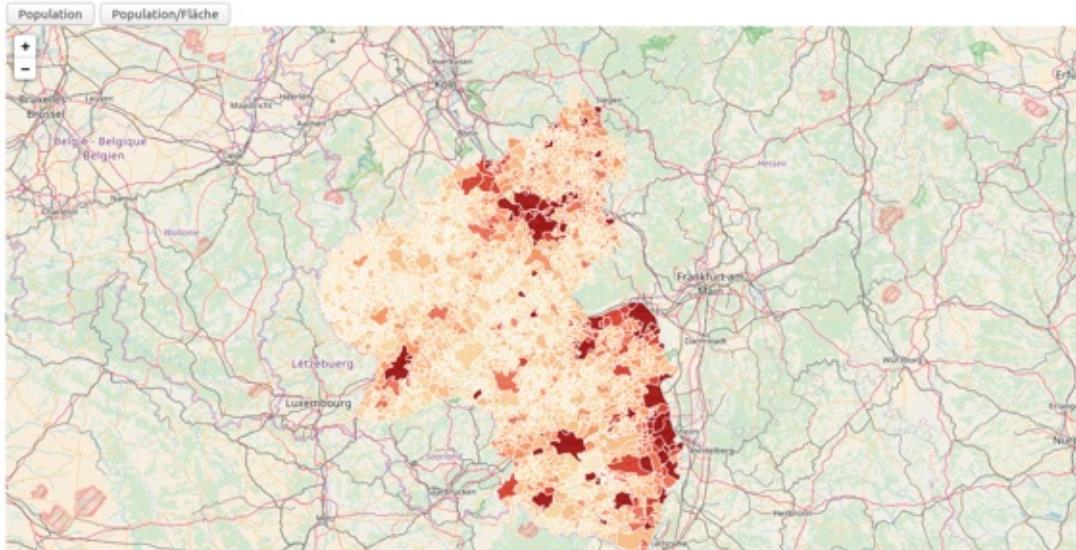
removeLayer(geoJsonLayer);
}
</script>
</body>
</html>
<!--index_956.html-->

```

Das vorhergehende Codebeispiel baut auf dem davor erstellten Codebeispiel auf. Neben der `style` Funktion fügen wir eine weitere Funktion, nämlich die Funktion `densityStyle()`, ein. Die Funktion `densityStyle()` gibt, wie der Name schon sagt, die Farbe für die *relative Angabe* zurück. Damit wir die beiden Choroplethenkarten Ebenen vergleichen können, habe ich eine Schaltfläche in das HTML Dokument integriert. Je nachdem welche Schaltfläche angeklickt wird, wird die passende Formel ausgeführt. Einmal die, die absoluten Werte anzeigt, ein anderes mal die, die die relativen Werte ausgibt.

Nun werden Sie feststellen, dass viele große Bereiche in ländlichen Gegenden, nicht mehr dunkel eingefärbt sind. Im ersten Beispiel, in dem wir mit den absoluten Zahlen gearbeitet haben, waren diese Bereiche teilweise dunkel. Die Bevölkerungszahl ist

aber in Relation zur Fläche nicht hoch.



## In diesem Kapitel haben wir ...

In diesem Kapitel haben Sie alles das, was Sie in den ersten Teilen kennengelernt haben, zum visualisieren eingesetzt. Sie haben Plugins kennen gelernt, die Sie beim Erstellen von Heatmaps unterstützen. Sie können nun eine Heatmap erstellen, mit der interagiert werden kann. Außerdem können Sie eine animierte Heatmap erstellen. Sie kennen die Besonderheiten von Choroplethenkarten und wissen, wann absolute und wann relative Werte sinnvoll sind. Sie können nun die Anzeige von absoluten und relativen Werten selbst umsetzen.

Im nächsten Kapitel werden wir noch einmal den Schwerpunkt auf die individuelle Gestaltung setzen. Hier geht es um benutzerdefinierte Marker.

# Custom Markers

Sie wissen nun wie Sie die unterschiedlichsten Geodaten auf Ihrer Karte anzeigen können und auch wie Sie mit den Geodaten ein Thema visualisieren können.

## In diesem Kapitel werden wir ...

In diesem Kapitel werden wir über die reine Anzeige hinaus noch einen Schritt weiter gehen. Nun geht es darum, der Karte eine individuelle Note zu geben. Wir sehen uns an, wie Sie Marker beliebig gestalten können. Außerdem sehen wir uns Plugins an, die Sie bei dieser Arbeit unterstützen.

## Ein individueller Marker auf Ihrer Karte

Wenn Leaflet einen Marker auf einer Karte anzeigt, werden gleich zwei Bilddateien angezeigt. Zunächst wird das eigentliche Bild an die passende Stelle auf die Karte gelegt. Danach wird ein Schatten zu diesem Bild hinzugefügt. Mit einem passenden Schatten fallen die Marker eher ins Auge. Der Schatten verleiht einem Marker eine Tiefe. So hebt dieser sich besser von der Karte ab.

Wenn Sie die Dateien zu Leaflet, wie im Kapitel *Eine lokale Leaflet-Kopie einbinden* beschrieben, auf Ihren Rechner kopiert haben, sehen Sie unter den kopierten Daten einen Unterordner mit dem Namen `images`. Dieser Ordner enthält die Imagedateien die angezeigt werden, wenn kein individuelles Image angegeben ist. Ich habe die Bilder hier nachfolgend abgedruckt. Wenn Sie dieses Buch bisher durchgearbeitet haben, kommen Ihnen die Bilder sicher teilweise bekannt vor.



Oft ist es so, dass das Bekannte vertraut ist und man sich deshalb damit sicher und wohl fühlt. Manchmal möchte man aber aus der Reihen tanzen. Wenn Sie auf Ihrer Karte außergewöhnliche Stellen mit einem Marker markieren möchten, dann sollten diese Marker vielleicht aus der Reihe tanzen. Wenn Sie an einer besonderen Stelle keine blaue Einheitsgrafik anzeigen möchten, dann zeigen Sie doch Ihre eigene Grafik an!

Haben Sie schon eine schöne Grafik für Ihren Marker und den passenden Schatten dazu? Wie Sie ein eigenes Image mit einem Grafikprogramm erstellen, gehört nicht zum Thema dieses Buches. Hier möchte ich Ihnen nur ein paar Punkte aufzählen, die

Sie beim Erstellen des Images für einen Leaflet Marker beachten sollten. Falls Sie keine Grafiken besitzen und auch nicht selbst Hand anlegen möchten, dann können Sie entweder die Übungen mit den Beispielbildern des Leaflet Tutorials durcharbeiten – oder Sie blättern direkt weiter zum nächsten Kapitel. Das Kapitel *Ein Marker Plugin* bietet Ihnen einen Kompromiss. Sie benötigen keine eigenen Grafiken, können einem Marker aber trotzdem ein anderes Aussehen verleihen.

Sie möchten gerne selbst die Grafiken erstellen, wissen aber noch nicht genau wie und womit? Dann sehen Sie sich doch das Programm GIMP an. GIMP (GNU Image Manipulation Program) ist eine gute kostenlose Alternative zum Bildbearbeitungsprogramm Photoshop von Adobe und kommt mit zahlreichen professionellen Bearbeitungsfunktionen.

Wenn Sie zwei Bilder haben – also ein Bild, das Ihren Marker selbst darstellt und eines, das den Schatten zeigt – dann können wir diese beiden Bilder als Marker in Ihre Karte einbinden. Ich habe hier zum Ausprobieren die Bild-Dateien aus dem Leaflet Tutorial Markers with Custom Icons verwenden.



Im nachfolgenden Beispiel habe ich den Text, der für die Anzeige des benutzerdefinierten Markers verantwortlich ist, fett formatiert.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 10);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var greenIcon = L.icon({
iconUrl: 'http://leafletjs.com/
examples/custom-icons/leaf-green.png',
shadowUrl: 'http://leafletjs.com/
examples/custom-icons/leaf-shadow.png',
iconSize: [38, 95],
```

```

shadowSize: [50, 64],
iconAnchor: [22, 94],
shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
});

L.marker(
[50.27264, 7.26469],
{icon: greenIcon}
).addTo(mymap)
.bindPopup(
"Ich bin ein Marker mit einem individuellen Image."
);

</script>
</body>
</html>
<!--index_955.html-->

```

Sehen wir uns diese Zeilen genau an: Zunächst einmal sticht die Instanziierung des Bildes hervor.

```

var greenIcon = L.icon({
iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-green.png',
shadowUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-shadow.png',
iconSize: [38, 95],
shadowSize: [50, 64],
iconAnchor: [22, 94],
shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
});

```

Die Bedeutung der einzelnen Optionen habe ich in der nachfolgenden Abbildung veranschaulicht. Aber zunächst einmal sehen wir uns das Problem genauer an: Die Schwierigkeit beim Positionieren des Bildes liegt darin, die Position des Icons mit der Stelle auf der Erde, die markiert werden soll, zu verbinden. Das Bild ist in der Regel größer als der zu markierende Punkt. Außerdem enthält das Bild oft eine Art Pfeil, dessen Spitze auf den zu markierenden Punkt zeigen sollte. Zudem gibt es meist noch ein Pop-up Fenster, dass relativ zum Bild geöffnet werden sollte. Wie also nutzen Sie die Optionen, um das Bild an der passende Stelle auf der Karte anzuzeigen?

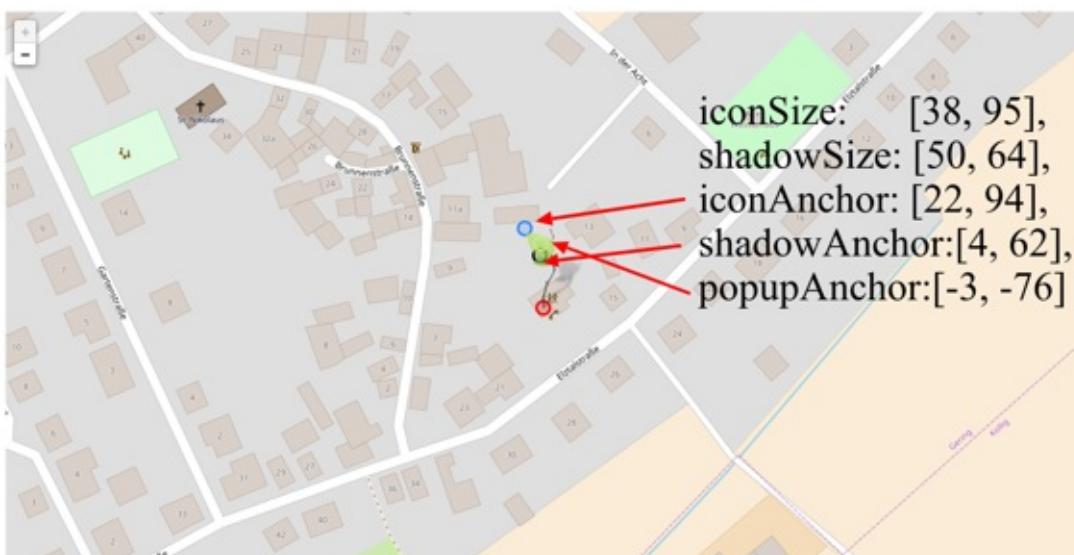
Sie können mit der optionalen Option `iconSize: [38, 95]` die Größe des Bildes mitgeben. Mit der optionalen Option `shadowSize: [50, 64]` können Sie die Größe des Schattens beeinflussen. Wenn Sie die Werte für diese Option nicht setzen, wird die Originalgröße des Images verwendet.

Die Option `iconAnchor: [22, 94]` gibt Ihnen nun die Möglichkeit, die Stelle, an der das Bild in die Karte eingefügt werden soll, zu definieren. Ohne ein Setzen dieser Option, würde die linke obere Ecke des Bildes an der Stelle, die markiert werden soll,

beginnen. In der Regel ist es aber so, dass man das Bild mittig oder vielleicht sogar über dieser Stelle einfügen möchte. `iconAnchor: [22, 94]` fügt das Bild 22 Pixel weiter links und 94 Pixel oberhalb der zu markierenden Stelle ein.

Wenn Sie sich die nachfolgende Abbildung ansehen, wird dies klar. Der rote Punkt stellt in der Abbildung die zu markierende Stelle dar. Für die Option `shadowAnchor: [4, 62]` gilt das gleiche wie für `iconAnchor`. Das Schattenbild wird 4 Pixel links und 76 Pixel oberhalb des roten Punktes, also der zu markierenden Stelle, eingefügt.

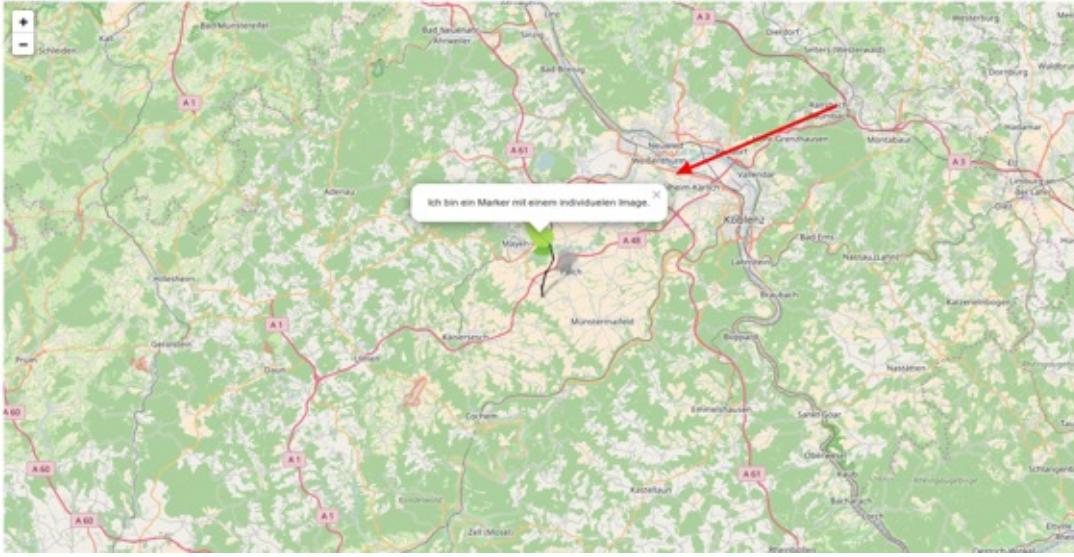
Die Belegung der Wert in der Option `popupAnchor: [-3, -76]` ist meiner Meinung nach etwas verwirrend, weil im Gegensatz zu den anderen Optionen für die gleiche Wirkung ein Minuszeichen vorangestellt werden muss. Wenn Sie das Pop-up Fenster links obenhalb der zu markierenden Position öffnen möchten, müssen Sie bei der Option `popupAnchor` also ein Minuszeichen voran stellen!



Im Kapitel *Die Karte mit Daten bestücken – genau im Unterkapitel *Punkte und Marker** – haben wir schon Marker mit Optionen angelegt. Im nächsten Codeschnipsel sehen Sie nun, dass Sie auch das Bild zum Marker – also das Icon – als Option angeben können. Im Programmcode geben Sie dazu einfach den Namen des eben erstellen `L.Icon` Objektes an.

```
L.marker(  
[50.27264, 7.26469],  
{icon: greenIcon}  
).addTo(mymap).bindPopup("Ich bin ein Marker mit einem individuellen  
Image");
```

Wenn Sie das HTML Dokument dieses Beispiels im Browser öffnen, sehen Sie eine Karte auf der das grüne Icon als Marker an der festgelegten Koordinate – passend positioniert – erscheint.



## Eigenschaften eines individuellen Marker

Einen eigenes Marker Bild erstellen Sie in Leaflet mithilfe der Klasse L.icon. Das haben Sie eben praktisch gesehen. Diese Klasse bietet Ihnen zehn Optionen.

- **iconUrl:**  
Die iconUrl müssen Sie auf alle Fälle angeben. Diese Option muss die Adresse zur Bilddatei enthalten – absolut oder relativ zu Ihrem Skript-Pfad.
- **iconRetinaUrl:**  
Die iconRetinaUrl bietet Ihnen optional die Möglichkeit, die Adresse einer für Retina Bildschirme optimierten Version des Bildes – absolut oder relativ zum Skript-Pfad – anzugeben.
- **iconSize:**  
Die optionale iconSize beeinflusst die Größe, in der das Bild angezeigt wird.
- **IconAnchor:**  
Die Option IconAnchor beschreibt die Pixel Koordinate, an der das Bild eingefügt werden soll. Die Koordinate gibt die Pixelwerte relativ zur zu markierenden Stelle an.
- **PopupAnchor:**  
Die Option PopupAnchor beschreibt den Punkte, an dem ein Pop-up Fenster geöffnet werden soll. Die Koordinate gibt die Pixelwerte relativ zur zu markierenden Stelle an. Wenn Sie das Pop-up Fenster links obenhalb der zu markierenden Position öffnen möchten, müssen Sie bei den Werten der Option popupAnchor ein Minuszeichen voran stellen!
- **ShadowUrl:**  
Die Option ShadowUrl enthält die Adresse zur Imagedatei, die den Schatten

darstellen soll. Wenn nichts angegeben ist, wird kein Schattenbild erstellt.

- **ShadowRetinaUrl:**

Mit der Option `ShadowRetinaurl` können Sie ein Bild angeben, dass speziell für Retina Displays optimiert ist. Wie Leaflet dieses Bild genau optimiert, erkläre ich Ihnen im Anschluss an die Auflistung dieser Optionen.

- **shadowSize:**

`shadowSize` beschreibt die Höhe und Breite des Bildes, dass den Schatten darstellen soll, in Pixeln.

- **shadowAnchor :**

Die Pixel Koordinaten an der das Schattenbild eingefügt werden soll, können Sie mit der Option `ShadowAnchor` übergeben. Ansonsten gilt für diese Option das Gleiche, was ich bei der Option `iconAnchor` geschrieben habe.

- **className :**

Mit der Option `className` können Sie den Namen einer CSS-Klasse, die beiden Bildern - also dem Schattenbild und dem eigentlichen Marker Bild - hinzugefügt wird, definieren.

Hochauflösende Displays haben eine höhere Pixeldichte als gewöhnliche Monitore. Auf der gleichen Fläche werden etwa viermal so viele Pixel dargestellt. Der Vorteil dieser Technologie liegt darin, dass die Pixel nun so klein sind, dass das menschliche Auge sie nicht mehr auflösen kann. Das Ergebnis sind sehr scharfe Grafiken und Texte. Damit das Bild nun auf einem HiDPI (High Dots Per Inch) Bildschirm, also einem Retina Display, scharf dargestellt wird, muss es mit mindestens zweifacher Breite und Höhe zur Verfügung gestellt werden. Denn: Eine Pixelgrafik, die bei gewöhnlicher Auflösung das ganze Display ausfüllt, würde auf einem Retina Display der gleichen Größe nur ein Viertel des Displays einnehmen. Ein Pixel der Grafik entspricht auch auf dem Retina-Display einem Pixel. Es werden aber auf der gleichen Fläche viermal so viele Pixel kleiner abgebildet. Damit die Pixelgrafiken nicht plötzlich alle zu klein dargestellt werden, rechnen die Geräte die Grafiken um. Dadurch geht Qualität verloren. Pixelgrafiken sehen auf dem Retina-Display daher unscharf aus. Um dieses Problem zu umgehen, prüft Leaflet die Auflösung des Anzeigegerätes. Anschließend werden die Marker Bilder – sofern eine Grafik mit höherer Auflösung vorhanden ist – in hoher Auflösung angezeigt.

## Die Klasse `L.Icon` erweitern

So, nun haben Sie einen benutzerdefinierten Marker erstellt und möchten weitere Marker kreiere. Schön, dass Leaflet objektorientiertes Arbeiten unterstützt. So können Sie sich das Erstellen von neuen Marker Objekten sehr einfach machen. Erweitern Sie einfach die Klasse `L.Icon` und geben alle gemeinsamen Eigenschaften hier nur einmal an. Nur die besonderen Eigenschaften des Markers müssen Sie separat für jeden Marker angeben. Das nachfolgende Codebeispiel zeigt Ihnen, wie Sie drei unterschiedliche Marker, die aber gemeinsame Eigenschaften haben, mithilfe

von einer Eltern-Klasse erstellen können. So können die drei Marker von dem Elternteil die gemeinsamen Eigenschaften erben.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 9);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var LeafIcon = L.Icon.extend({
options: {
shadowUrl: 'leaf-shadow.png',
iconSize: [38, 95],
shadowSize: [50, 64],
iconAnchor: [22, 94], shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
}
});

var greenIcon = new LeafIcon(
{iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-green.png'}
);
var redIcon = new LeafIcon(
{iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-red.png'}
);
var orangeIcon = new LeafIcon(
{iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-orange.png'}
);

L.marker(
[50.27264, 7.26469], {icon: greenIcon}
).addTo(mymap).bindPopup(
"Ich bin ein grüner Marker."
);
L.marker(
[50.27264, 6.26469], {icon: redIcon}
).addTo(mymap).bindPopup(
"Ich bin ein roter Marker."
);
L.marker(
[50.27264, 8.26469], {icon: orangeIcon}
).addTo(mymap).bindPopup(
"Ich bin ein oranger Marker."
);

</script>
</body>
</html>
<!--index_954.html-->
```

Dieser Programmcodeabschnitt hat Ähnlichkeit mit dem vorherigen Beispiel. Wenn Sie genau hinsehen, finden Sie aber Unterschiede. Wir erstellen im ersten Schritt kein neues Icon Objekt um dieses Anzuzeigen, sondern erweitern die Klasse L.Icon mit der Klasse LeafIcon. Wir legen nur die gemeinsamen Optionen für das Objekt LeafIcon fest. Die URL des Bildes ist die Option, die von Marker zu Marker unterschiedlich ist. Deshalb geben wir diese beim Erweitern der Klasse in den Optionen noch nicht an.

```
var LeafIcon = L.Icon.extend({
  options: {
    shadowUrl: 'leaf-shadow.png',
    iconSize: [38, 95],
    shadowSize: [50, 64],
    iconAnchor: [22, 94],
    shadowAnchor: [4, 62],
    popupAnchor: [-3, -76]
  }
});
```

Im zweiten Schritt erstellen wir drei Instanzen des LeafIcon, also des erweiterten L.Icon

```
var greenIcon = new LeafIcon(
  {iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-green.png'}
);
var redIcon = new LeafIcon(
  {iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-red.png'}
);
var orangeIcon = new LeafIcon(
  {iconUrl: 'http://leafletjs.com
/examples/custom-icons/leaf-orange.png'}
);
```

Und zu guter Letzt fügen wir die Marker mit dem zugehörigen Icon an die passende Stelle auf der Karte zum Kartenobjekt hinzu.

```
L.marker(
[50.27264, 7.26469],
{icon: greenIcon}).addTo(mymap)
.bindPopup("Ich bin ein grüner Marker.")
);
L.marker(
[50.27264, 6.26469],
{icon: redIcon}).addTo(mymap)
.bindPopup("Ich bin ein roter Marker.")
);
L.marker(
[50.27264, 8.26469],
```

```
{icon: orangeIcon}).addTo(mymap)
.bindPopup("Ich bin ein oranger Marker.")
);
```

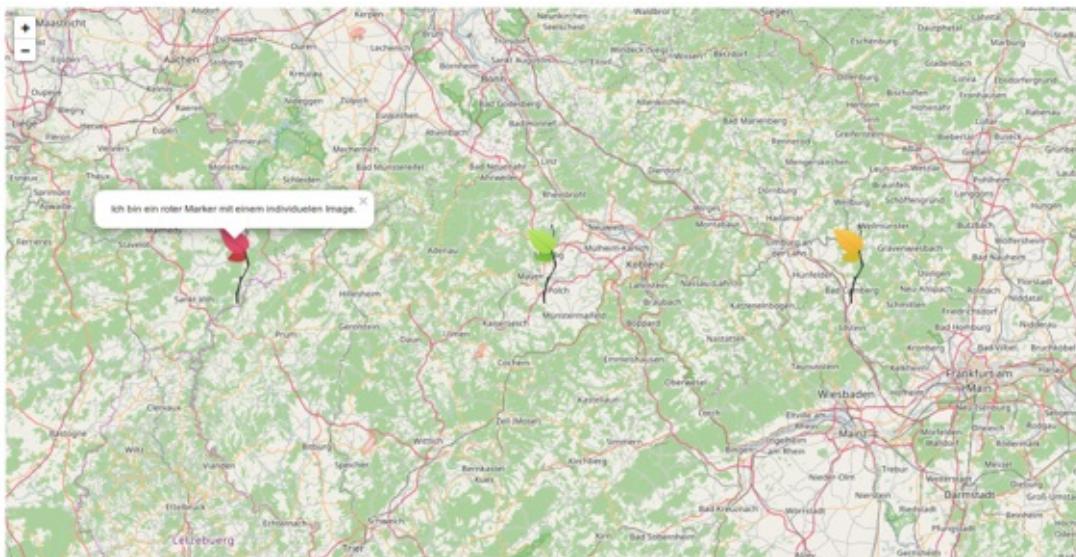
Sie haben vielleicht bemerkt, dass wir das Schlüsselwort `new` für die Erstellung von LeafIcon Instanzen verwendet haben. Warum haben wir vorher alle Leaflet Objekte ohne das Schlüsselwort `new` erstellt? Die Antwort ist einfach: Die echten Leaflet Klassen sind mit einem Großbuchstaben – beispielsweise `L.Icon` – benannt und diese müssen mit `new` erstellt werden. Es gibt aber Shortcuts mit Kleinbuchstaben – `L.icon` – die aus Bequemlichkeitsgründen von den Leaflet-Programmierern für Sie erstellt wurden:

```
L.icon = function icon(options) {
return new L.Icon(options);
};
```

Die Funktion `L.icon` können Sie sich auf Github in der Datei `icon.js` ansehen.

Leaflet setzt hier das Entwurfsmuster Fabrikmethode (englisch factory method) ein. Das Muster beschreibt, wie ein Objekt durch Aufruf einer Methode, anstatt durch direkten Aufruf eines Konstruktors, erzeugt wird.

Im nachfolgenden Bild sehen Sie das Ergebnis. Jeder Marker wird nun mit einem individuellen Icon erstellt. Die meisten Optionen sind gleich – allerdings hat jeder Marker seine eigene Farbe.



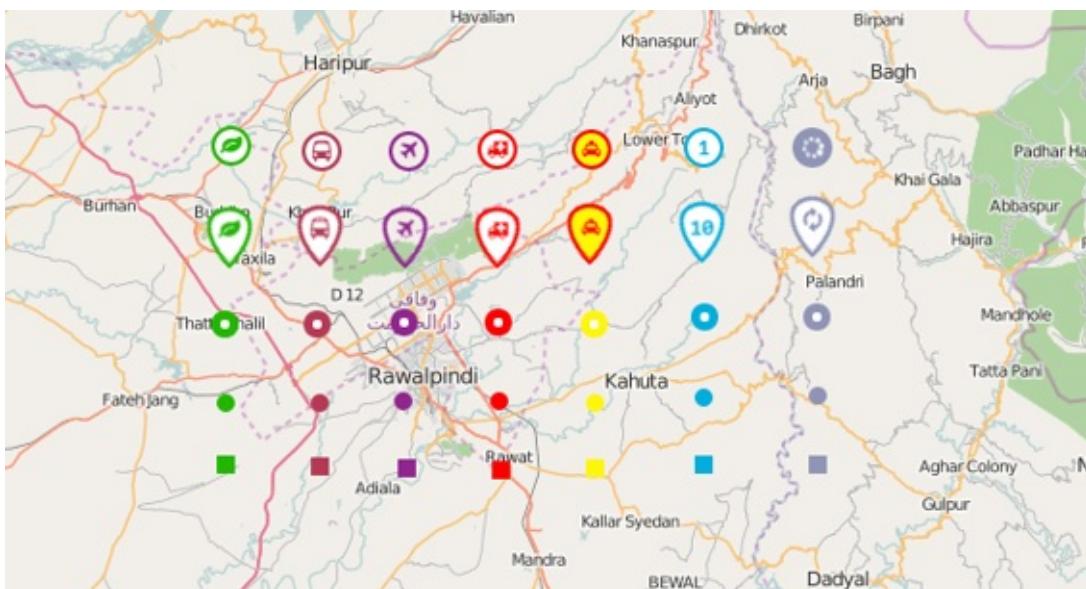
## Ein Marker Plugin

Sie wissen nun, wie Sie einen Marker mit einem standardisierten Aussehen einfügen und können einen Marker mit einem eigenen Image belegen. Leider haben Sie aber kein eigenes Image und haben auch nicht viel Erfahrung mit einem Grafikprogramm. Sie können kein professionell aussehendes individuelles Image hervor zaubern oder haben einfach nicht die Zeit dazu. Trotzdem möchten Sie Ihrer Karte ein besonderes

Aussehen verleihen. In diesem Fall bietet Ihnen dieses Kapitel eine Lösung. Ich stelle Ihnen zwei Plugins vor, die Sie bei der Erstellung von individuellen Marker Objekten unterstützen.

## BeautifyIcon

Leaflet.BeautifyIcon, ist ein einfaches Plugin, das bunte Marker ganz ohne eigene Grafik zu Leaflet hinzufügt. Trotzdem behalten Sie die volle Kontrolle über den Stil der Marker. Konkret heißt das: Sie können über unbegrenzte Farben und viele Eigenschaften verfügen. Das Plugin Leaflet.BeautifyIcon bietet auch die Möglichkeit, Schriftart und Glyphen, also die grafische Darstellung von Schriftzeichen, anzupassen.



Damit Sie sich dieses besser vorstellen können, habe ich ein Beispiel erstellt.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css"
>
<link rel="stylesheet"
href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css"
>
<link rel="stylesheet"
href="leaflet-beautify-marker-icon.css">
<script src="leaflet-beautify-marker-icon.js"
></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
```

```

<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 8);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

L.marker(
[50.27264, 7.26469],
{icon: L.BeautifyIcon.icon({iconSize: [50, 50]}),draggable: true}
).addTo(mymap).bindPopup("Ich bin ein beautify Marker");

options = {
icon: 'spinner',
spin: 'true',
borderColor: '#8A90B4',
textColor: 'white',
backgroundColor: '#8A90B4'
};
L.marker(
[50.27264, 6.26469],
{icon: L.BeautifyIcon.icon(options),draggable: true}
).addTo(mymap).bindPopup("Ich bin ein beautify Marker");

options = {
icon: 'plane',
iconShape: 'marker',
borderColor: '#8D208B',
textColor: '#8D208B',
backgroundColor: 'transparent'
};
L.marker(
[50.27264, 8.26469],
{
icon: L.BeautifyIcon.icon(options),
draggable: true
}
).addTo(mymap).bindPopup("Ich bin ein beautify Marker");

</script>
</body>
</html>
<!--index_953.html-->

```

Was müssen Sie tun, wenn Sie das Plugin Leaflet.BeautifyIcon verwenden möchten? Zunächst einmal müssen Sie die notwendigen Skripte und Stylesheet Dateien einbinden. Das ist zum einen das Skript und die CSS-Datei zum Plugin selbst. Zum anderen können Sie über Leaflet.BeautifyIcon Drittdienste nutzen. Sie können Font Awesome CSS und Bootstrap CSS einbinden. Ich habe im Beispiel die CSS-Dateien von Font Awesome und von Boostrap eingebunden, um Ihnen dies zu demonstrieren. Für dieses Beispiel wäre nur die CSS-Datei von Font Awesome CSS notwendig gewesen.

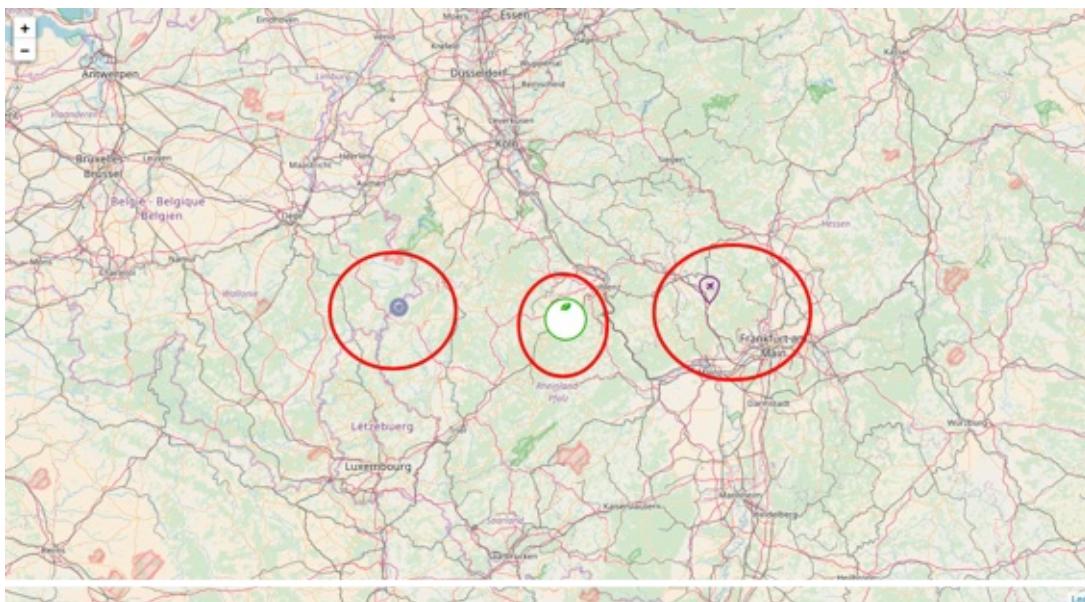
Mehr müssen Sie nicht tun. Sie können sofort einen Marker mit der Option `icon: L.BeautifyIcon.icon(options)` kreieren.

```

L.marker([50.27264, 8.26469], {
icon: L.BeautifyIcon.icon(options),
draggable: true
}).addTo(mymap).bindPopup("Ich bin ein beautify Marker");

```

Sehen Sie sich die Optionen des Plugin Leaflet.BeautifyIcon an. Es macht Spaß diese zu erkunden. Wie die Optionen wirken, die ich verwendet haben, können Sie sich im nächsten Bild teilweise ansehen. Das sich eines der Icons dreht, erkennen Sie nur, wenn Sie die Datei selbst im Browser öffne.



Ich hatte Ihnen ja schon geschrieben, dass es jede Menge Plugins für Leaflet gibt und dies gilt für den Bereich Marker besonders. Die meisten sind auf der Website von Leaflet aufgelistet. Diese Liste finden Sie unter der Adresse <http://leafletjs.com/plugins.html#markers—renderers>.

## Cluster

Je nachdem welche Informationen Sie mit Ihrer Karte weitergeben möchten, kann es vorkommen, dass Sie sehr viele Marker benötigen. Wenn Sie mit vielen Marker Objekten arbeiten, sollten Sie beachten, dass diese das Laden der Karte verlangsamen. Außerdem kann es vorkommen, dass Marker nahe nebeneinander liegen und sich beim Zoomen überschneiden. Dies ist nicht benutzerfreundlich. Schön wäre es, wenn bei einer vergrößerten Ansicht alle Marker zu sehen sind – diese aber beim Hineinzoomen in die Karte zu Clustern zusammengefasst werden. So hat der Benutzer alle Informationen passend zur Kartenanzeige.

In diesem Kapitel erfahren Sie, wie Sie das Plugin Leaflet.markercluster zum Clustern von Marker Objekten verwenden und so eine große Anzahl von Marker Objekten auf einer Karte benutzerfreundlich und übersichtlich darstellen können. Sehen Sie sich das nachfolgende Beispiel an, um zu verstehen, wie das Clustern von Marker Objekten funktioniert.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
```

```

<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<link rel="stylesheet" href="MarkerCluster.css"/>
<link rel="stylesheet" href="MarkerCluster.Default.css"/>
<script src="leaflet.markercluster-src.js">
</script><script src="points.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid',
{doubleClickZoom:false})
.setView([50.219264, 7.19469], 13);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var markers = L.markerClusterGroup(
{showCoverageOnHover : false}
);

for (var i = 0; i < points.length; i++) {
var a = points[i];
var title = a[2];

var marker = L.marker(new L.LatLng(a[0], a[1]),
{ title: title });

marker.bindPopup(title);
markers.addLayer(marker);
}

mymap.addLayer(markers);
</script>
</body>
</html>
<!--index_952.html-->

```

Das haben wir gemacht? Als erstes haben wir die notwendigen Dateien zum Plugin Leaflet.markercluster eingebunden. Damit wir auch genug Marker zum Clustern zur Verfügung haben, habe wir dann Punkte über eine externe Datei eingebunden. Die Datei `points.js` enthält 380 Punkte und sieht auszugsweise wie folgt aus:

```

var points = [
[50.2210922667, 7.2209316333, "2"],
[50.2210819833, 7.2213903167, "3"],
...
];

```

Zu guter Letzt mussten wir mit

```

var markers = L.markerClusterGroup(
{showCoverageOnHover : false}
);

```

ein Objekt vom Typ `markerClusterGroup` erzeugen und diesem Objekt jeden einzelnen Marker hinzufügen. Das Objekt `markerClusterGroup` übernimmt nun die

ganze Arbeit für uns.

So wie in der nachfolgenden Abbildung zu sehen ist, könnte Ihre Karte aussehen.



## Optionen, Methoden und Ereignisse

Sie können einem Cluster jede Menge Optionen mitgeben und sehr viele Methoden und Ereignisse nutzen. Zum Beispiel können sie die standardmäßig aktivierte Option `showCoverageOnHover` mit

```
var markers = L.markerClusterGroup(  
{showCoverageOnHover : false}  
);
```

ausschalten. Aktiviert bewirkt diese Option folgendes: Wenn Sie die Maus über einen Cluster bewegen, blendet sich ein Polygon ein, dass die Grenzen des Bereichs in dem die Marker sich befinden, anzeigt.

Alle Optionen, Methoden und Ereignisse finden Sie in der Dokumentation zum Plugin auf Github.

## Marker animieren

### Hüpfende Marker

Wenn Sie einen Marker animieren möchten, unterstützt Sie das Plugin <https://github.com/maximeh/leaflet.bouncemarker>. Wie Sie den hüpfenden Marker in Ihre Karte einbinden, zeigte ich Ihnen wieder anhand eines Beispiels.

```
| <!DOCTYPE HTML>
```

```

<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="bouncemarker.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 12);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

L.marker([50.27264, 7.26469],
{
bounceOnAdd: true,
bounceOnAddOptions: {duration: 5000, height: 100},
bounceOnAddCallback: function() {alert("Gelandet");}
})
.addTo(mymap);

</script>
</body>
</html>
<!--index_951.html-->

```

Als Erebnis sehen Sie einen Marker, der in die Karte springt. Nachdem der Marker gelandet ist, öffnet sich ein Pop-up-Fenster mit der Meldung: Gelandet!.

## Animierte Marker

### Ein Marker bewegt sich

Es gibt sehr viele spannende Ideen, die man mit einer Karte umsetzen kann. Möchten Sie vielleicht mit Ihrem Marker einen Weg beschreiten? Dann ist das Plugin <https://github.com/openplans/Leaflet.AnimatedMarker> vielleicht etwas für Sie. Mit diesem Plugin können Sie einen Marker so animieren, dass er einer Linie folgt. Vielleicht möchten Sie einen Marker in Form eines Autos darstellen, dass auf einer Straße fährt?

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="AnimatedMarker.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 9);

```

```

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var line = L.polyline(
[[50.68510, 7.94136],
[50.68576, 6.94149],[51.68649, 6.94165]
]);

animatedMarker = L.animatedMarker(line.getLatLngs(), {
distance: 2000,
interval: 1000,
});

mymap.addLayer(animatedMarker);
</script>
</body>
</html>
<!--index_950.html-->
```

## Einen Marker in Bewegung setzen und wieder stoppen

Und auch wenn Sie die Bewegung des Markers beeinflussen möchten, unterstützt Sie das Plugin <https://github.com/openplans/Leaflet.AnimatedMarker>. Bauen Sie in diesem Falle doch einfach zwei Schaltenflächen ein, über die Sie den Maker anhalten oder starten können. Das nächste Beispiel will Ihnen eine Idee zur Umsetzung dieser Aufgabenstellung geben.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="AnimatedMarker.js"></script>
</head>
<body>

<button onclick="start()">Start</button>
<button onclick="stop()">Stop</button>

<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 9);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var line = L.polyline(
[[50.68510, 7.94136],[50.68510, 7.84136],
[50.68510, 7.74136],[50.68510, 7.64136],
[50.68510, 7.5136],[50.68510, 7.44136],
[50.68510, 7.34136],[50.68510, 7.24136]]);

animatedMarker = L.animatedMarker(line.getLatLngs(), {
autoStart: false,
distance: 200,
interval: 100
});
```

```

mymap.addLayer(animatedMarker);

function start(){
animatedMarker.start();
}

function stop(){
animatedMarker.stop();
}

</script>
</body>
</html>
<!--index_949.html-->
```

Wie Sie sehen, können Sie die Funktionen `start()` und `stop()`, die Ihnen das Plugin <https://github.com/openplans/Leaflet.AnimatedMarker> zur Verfügung stellt, nutzen.

## Plugins kombinieren

Und natürlich können Sie Plugins auch kombinieren. Ihrer Kreativität sind keine Grenzen gesteckt. Schon allein mit den beiden gerade gezeigten Plugins Leaflet.AnimatedMarker und Leaflet.bouncemarker können Sie einen Marker auf der Karte Aktionen ausführen lassen und ihn am Schluss gekonnt aus dem Bild hüpfen lassen. Sehen Sie selbst, probieren Sie das nächste Beispiel aus!

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="AnimatedMarker.js"></script>
<script src="bouncemarker.js"></script>
</head>
<body>
<button onclick="start()">Start</button>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

b = new L.Marker([51.68649, 6.94165],
{bounceOnAdd: true});
var line = L.polyline(
[[50.68510, 7.94136],[50.68576, 6.94149],
[51.68649, 6.94165]]),
animatedMarker = L.animatedMarker(line.getLatLngs(), {
autoStart: false,
distance: 2000,
interval: 10,
onEnd: function() {
b.addTo(mymap);
b.bounce({duration: 100, height: 50});
```

```

mymap.removeLayer(animatedMarker);
setTimeout('mymap.removeLayer(b)', 900);
}
});
mymap.addLayer(animatedMarker);
function start(){animatedMarker.start();}
</script>
</body>
</html>
<!--index_948.html-->
```

## Leaflet Data Visualization Framework (DVF)

Möchten Sie gerne mit Ihrer Karte Daten visualisieren. Heatmaps und Choroplethenkarten sind aber nicht das Richtige für Sie? Sie können Daten mithilfe von Markern in überzeugende Karten verwandeln. Hierbei unterstützt Sie das Plugin Leaflet Data Visualization Framework (DVF). Leider ist dieses Plugin zu dem Zeitpunkt, zu dem ich dieses Kapitel geschrieben haben, nicht mit der neuesten Leaflet Version kompatibel. Deshalb verwende ich im Beispiel eine ältere Leaflet Version.

### Das Plugin Data Visualization Framework

Das erste Beispiel zeigt Ihnen, wie Sie Formen als Marker darstellen können. Hier geht es noch nicht um das Visualisieren von Daten, sondern eher darum, die Möglichkeiten diese Plugins zu sehen und ein Gefühl für die Anwendung zu bekommen.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet"
href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />

<script src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js">
</script>

<link rel="stylesheet" href="dvf.css" />
<script src="leaflet-dvf.js"></script>
<script src="leaflet-dvf.markers.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var marker = new L.MapMarker([50.27264, 7.26469],
{
radius: 30,
```

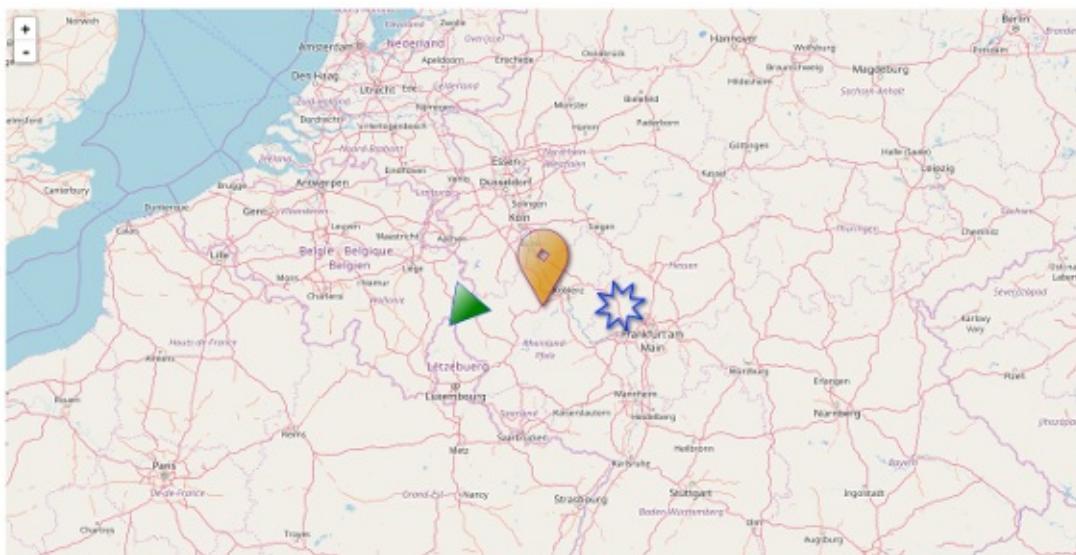
```

fillOpacity:0.5,
fillColor:'orange',
color:'purple',
innerRadius:7,
numberOfSides:4,
rotation:10
});
mymap.addLayer(marker);

var polygonmarker = new L.RegularPolygonMarker(
[50.27264, 6.26469],
{
numberOfSides: 3,
rotation: 10,
radius: 10,
fillColor:'green',
fillOpacity:1,
opacity:1,
weight:1,
radius:30
});
mymap.addLayer(polygonmarker);

var star = new L.StarMarker([50.27264, 8.26469],
{
numberOfPoints:8,
opacity:1,
weight:2,
fillOpacity:0,
radius:30});
mymap.addLayer(star);
</script>
</body>
</html>
<!--index_947.html-->
```

Die nächste Abbildung zeigt Ihnen die drei im vorherigen Beispiel erstellen Marker.



## Diagrammen als Marker

Das außergewöhnlichste an diesem Plugin ist meiner Meinung nach die Einfachheit, mit der Diagramme als Marker dargestellt werden können. Sehen sich dies im

nachfolgenden Beispiel an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet"
href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />

<script src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js">
</script>
<link rel="stylesheet" href="dvc.css" />
<script src="leaflet-dvc.js"></script>
<script src="leaflet-dvc.markers.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 7);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var options = {
data: {
'data1': 20,
'data2': 50,
'data3': 10,
'data4': 20
},
chartOptions: {
'data1': {
fillColor: 'blue',
minValue: 0,
maxValue: 50,
maxHeight: 30,
displayText: function (value) {
//return value.toFixed(2);
return 'Mein Text';
}
},
'data2': {
fillColor: 'red',
minValue: 0,
maxValue: 50,
maxHeight: 30,
},
'data3': {
fillColor: 'green',
minValue: 0,
maxValue: 50,
maxHeight: 30,
},
'data4': {
fillColor: 'yellow',
minValue: 0,
maxValue: 50,
maxHeight: 30,
}
},
weight: 1,
color: '#000000',
radius:30,
```

```

fillOpacity:1
};

var bar = new L.BarChartMarker(
[50.27264, 7.26469], options);
mymap.addLayer(bar);

var radial = new L.RadialBarChartMarker(
[50.27264, 8.26469], options);
mymap.addLayer(radial);

var pie= new L.PieChartMarker(
[50.27264, 6.26469], options);
mymap.addLayer(pie);

var cox = new L.CoxcombChartMarker(
[50.97264, 7.26469], options);
mymap.addLayer(cox);

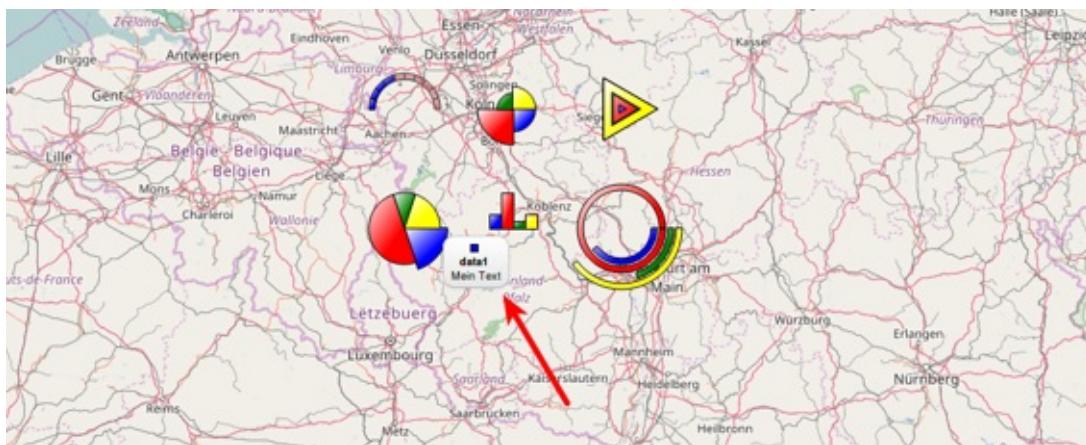
var stack = new L.StackedRegularPolygonMarker(
[50.97264, 8.26469], options);
mymap.addLayer(stack);

var radialmeter= new L.RadialMeterMarker(
[50.97264, 6.26469], options);
mymap.addLayer(radialmeter);

</script>
</body>
</html>
<!--index_946.html-->

```

Ich habe die Daten für dieses Beispiel künstlich erstellt. Das Prinzip wird so klar. Das Beispiel zeigt Ihnen, wie Sie Daten in Form eines Diagramms mithilfe des Leaflet Data Visualization Framework auf Ihrer Leaflet Karte einblenden können. Und das nachfolgende Bild zeigt Ihnen das Ergebnis.



## In diesem Kapitel haben wir ...

In diesem Kapitel haben wir der Leaflet Karte eine individuelle Note gegeben. Wir haben uns angesehen, wie Sie Marker beliebig gestalten können. Dabei haben wir einen Ausflug in die objektorientierte Programmierung gemacht. Insbesondere bei der Erstellung von vielen individuellen Marker Objekten kann man sich durch eine

mögliche Vererbung viel Arbeit und viele Programmcodezeilen sparen. Außerdem haben wir uns verschiedene Plugins angesehen – man muss ja nicht immer das Rad selbst neu erfinden.

Sie haben Ihre digitale Karte bereits im Griff. Sie können diese gestalten und Daten auf ihr visualisieren. Nun kann es sein, dass Sie selbst keine großen Datenbestände zum Anzeigen haben. Vielleicht bietet das ESRI Inc. (Environmental Systems Research Institute) Daten, die zum Thema Ihrer Website passen? Im nächsten Kapitel erfahren Sie, wie Sie Daten und Anwendungen dieses Institutes verwenden können.

# ESRI

Wenn Sie intensiver mit digitalen Landkarten arbeiten, möchten Sie sicherlich auch einmal Daten eines Geoinformationssystems (GIS) auf Ihrer Karte anzeigen.

Ein Geoinformationssystem (<https://de.wikipedia.org/wiki/Geoinformationssystem>) ist ein Informationssystem zur Erfassung, Bearbeitung, Organisation, Analyse und Präsentation von Daten, die einen Bezug zu einer Stelle auf unserer Erde haben. Zu einem Geoinformationssystem gehören dabei die notwendige Hardware, die notwendige Software, die Daten und die Anwendungen selbst.

Wenn Sie Daten eines Geoinformationssystems nutzen möchten, werden Sie früher oder später über den Begriff Shapefile (<https://de.wikipedia.org/wiki/Shapefile>) oder Shape Datei stolpern. Aber auch wenn Sie nie auf den Begriff Shape Datei stoßen werden Sie vielleicht einmal einen Webservice nutzen, der Endpunkt eines ARCServers ist. Außerdem werden Sie sicher auch das ESRI Inc., also das Environmental Systems Research Institute, kennenlernen. Dieses Institut ist ein US-amerikanischer Softwarehersteller. ESRI (<https://de.wikipedia.org/wiki/ESRI>) hat sich auf Geoinformationssysteme spezialisiert. Das Unternehmen hat unter anderem die Anwendung ArcGIS (<https://de.wikipedia.org/wiki/ArcGIS>) erstellt und das Dateiformat Shapefiles eingeführt.

Die wesentlichen Produkte des ESRI sind Geoinformationssysteme. Die Namen der verschiedenen Geoinformationssysteme des ESRI enthalten in der Regel den Namensteil ArcGIS. Es gibt ArcGIS Anwendungen für Server und für Clients.

Bitte lassen Sie sich durch die vielen neuen Begriffe nicht verunsichern. Natürlich gibt es Leaflet Plugins, die Ihnen beim Integrieren dieser Services zur Hand geht. Außerdem sehen wir uns alles Schritt für Schritt und nacheinander an.

## In diesem Kapitel werden wir ...

In diesem Teil werden wir uns als Erstes die Karten, die das ESRI (<https://www.esri.de>) anbietet, ansehen. Danach schauen wir, was sich hinter dem Begriff Shapefile verbirgt. Außerdem werden wir ESRI Webservices kennen lernen. Unter anderem einen L.esri.DynamicMapLayer, Geocoding und einen L.esri.FeatureLayer – hier konkret einen Query Layer. Mit letzterem können Sie Daten, die Sie nicht benötigen, einfach aus der gegebenen Datenmenge herausfiltern.

# L.esri.BasemapLayer erweitert L.TileLayer

Mit der Klasse L.esri.BasemapLayer können Sie Karten, die das Esri anbietet, auf Ihrer Website als Leaflet Karte anzeigen.

Die Nutzungsbedingungen für ESRI Dienste gelten für Leaflet Anwendungen. Diese können Sie auf der Website <https://github.com/esri/esri-leaflet#terms> nachlesen.

## Basemaps und optionale Layer von ESRI

Sie möchten Basemaps von ESRI verwenden. Wir haben im Kapitel *Schöne Kartenlayer* schon Karten von anderen Kartenanbietern eingebunden. Dieses Kapitel erweitert die Erklärungen im Kapitel *Schöne Kartenlayer* um ein weiteres Beispiel – nämlich die Karten des ESRI Instituts.

### Basemaps

Basemaps von ESRI bieten eine weltweite Abdeckung bei einer Vielzahl von Zoom Stufen. Dabei können die folgenden Themenbereiche wählen.

- Streets
- Topographic
- NationalGeographic
- Oceans
- Gray
- DarkGray
- Imagery
- ShadedRelief
- Terrain
- USATopo (Diese Karte wird nur für die USA angeboten)

Wenn Ihnen eine der Basiskarten grundsätzlich gut gefällt, diese Ihnen aber zu wenig Informationen bietet, können Sie weitere optionale Ebenen über diese Basisebene einblenden. ESRI bietet speziell für jede Basisebene eine optionale Ebene mit

weiteren Informationen.

## Optionale Label Layer

Label Layer sind optionale Ebenen, die zusätzliche Textbeschriftungen zu den Basiskarten hinzufügen. ESRI bietet Ihnen sieben verschiedene optionale Schichten. Jede Ebene ist speziell für eine Basiskarte kreiert worden. Wenn Sie möchten, können Sie diese aber auch quer miteinander kombinieren. Konkret bietet ESRI die Layer

- OceansLabels:
- GrayLabels:
- DarkGrayLabels:
- ImageryLabels:
- ImageryTransportation:
- ShadedReliefLabels
- TerrainLabels

Alle möglichen Basiskarten mit optionalen Ebenen können Sie auch in der Dokumentation des Leaflet Plugins nachlesen. Genau finden Sie diese Informationen unter der Adresse <https://esri.github.io/esri-leaflet/api-reference/layers/basemap-layer.html>

Wenn Sie möchten, können Sie die Anzeige von Basemaps bei höheren Auflösungen auf Retina-Geräten mit der Option `detectRetina` optimal einstellen. Im nachfolgenden Programmcodeausschnitt wird die Basemap auf einem Retina Display in Retina-Auflösungen ausgegeben. Was ein Retina Display ist, habe ich Ihnen im Kapitel Custom Markers schon in einem Hinweis erklärt. Die Option `detectRetina` ist eine Leaflet Funktion

```
...
L.esri.basemapLayer('DarkGray', {
  detectRetina: true
}).addTo(map);
...
```

Wenn Sie die Option `detectRetina` mit `true` belegt haben und ein Website-Besucher Ihre Karte auf einem Gerät mit einem Retina-Display öffnet, wird Leaflet anstelle der Bilddateien zur aktuellen Zoom-Stufe die vierfache Anzahl der Bildkacheln der nächsthöheren Zoom-Stufe anfordern und anzeigen. Wie die Bilddateien für eine Zoom-Stufe berechnet werden, können Sie im Kapitel *Wie weiß Leaflet welche Kacheln angezeigt werden sollen?* nachlesen.

# Ein erstes Beispiel

Das nachfolgende Beispiel zeigt Ihnen, wie Sie eine ESRI Basiskarte in Leaflet anzeigen können.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet-debug.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 7);

var gray = L.esri.basemapLayer('Gray').addTo(mymap);

var graylabels = L.esri.basemapLayer('GrayLabels')
.addTo(mymap);
gray.setOpacity(0.5);
gray.on('load', alertme);

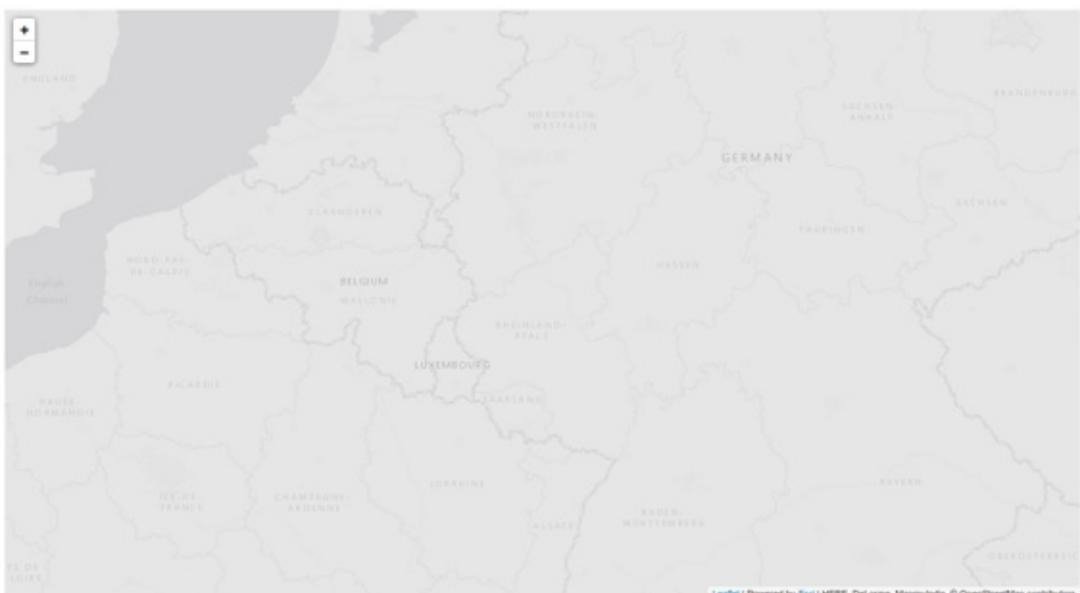
function alertme(){
alert("Fertig!");
}
</script>
</body>
</html>
<!--index_945.html-->
```

Das haben wir gemacht? Fangen wir mit dem wichtigen an: Wichtig ist, dass wir das Skript zum ESRI Leaflet Plugin einbinden. Ich habe dies mit der Zeile `<script src="esri-leaflet-debug.js"></script>` getan. Im Echtesystem können Sie die komprimierte Version verwenden, also `<script src="esri-leaflet.js"></script>`. Die aktuellste Version dieses Skripts können Sie von der Adresse <http://esri.github.io/esri-leaflet/download/> kopieren. Nach dem Einbinden des Plugins haben wir mit der Zeile `var gray = L.esri.basemapLayer('Gray').addTo(mymap);` das `L.esri.basemapLayer` Objekt erstellt und es auch sofort zur Karte hinzugefügt. Zusätzlich haben wir die Karte mit `gray.setOpacity(0.5);` transparent dargestellt und dem Ereignis `on('load')` die Funktion `alertme()` zugeordnet. Immer dann, wenn die Karte fertig geladen ist, soll die Funktion `alertme()` ausgeführt werden. Genau bedeutet das, dass immer, wenn die Karte fertig geladen ist, ein Hinweisfenster erscheint. Verschieben Sie die Karte einmal. Dabei werden Sie sehen, dass das Ereignis `on('load')` auch immer dann, wenn die Karte neu geladen wird, eintritt.

Die beiden nachfolgenden Abbildungen zeigen Ihnen die Basiskarte mit dem Namen `gray` – einmal mit und einmal ohne optionale Ebene.



*Die Basiskarte Gray mit optionaler Ebene*



*Die Basiskarte Gray ohne optionale Ebene*

## Switch Basemaps

Im nächsten Beispiel habe ich eine Auswahllist im HTML-Dokument über der Karte eingefügt. So können Sie zwischen den verschiedenen ESRI Karten schnell hin und her wechseln und sich so einen Überblick über das Kartenangebot verschaffen.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet-debug.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
```

```

<select
style ="position: absolute;top: 10px;right:10px;z-index: 400;"
id="basemaps"
onChange="changeBasemap(basemaps)"
>
<option value="Topographic">Topographic</option>
<option value="Streets">Streets</option>
<option value="NationalGeographic">National Geographic
</option>
<option value="Oceans">Oceans</option>
<option value="Gray">Gray</option>
<option value="DarkGray">Dark Gray</option>
<option value="Imagery">Imagery</option>
<option value="ShadedRelief">Shaded Relief</option>
<option value="Terrain">Terrain</option>
<option value="USATopo">USATopo </option>
</select>

<script>
var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 7);

var layer = L.esri.basemapLayer('Gray').addTo(mymap);

function changeBasemap(basemaps){
  var basemap = basemaps.value;

  if (layer){
    mymap.removeLayer(layer);
  }

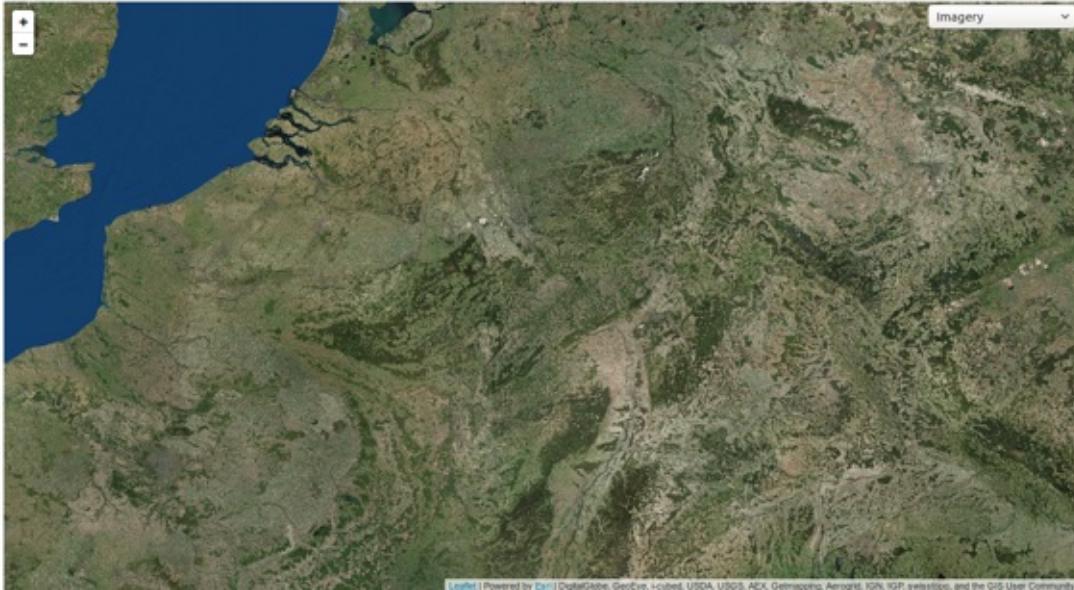
  layer = L.esri.basemapLayer(basemap);
  mymap.addLayer(layer);
}

</script>
</body>
</html>
<!--index_944.html-->

```

Was haben wir genau gemacht? Hier gibt es nichts weltbewegend Neues. Eine Auswahlliste haben wir schon oft im Buch verwendet, deshalb ist diese im Programmcodebeispiel auch nicht fett hervorgehoben. Wenn ein Website-Besucher den aktiven Listeneintrag der Auswahlliste ändert, wird die aktuelle Kartenebene mit `mymap.removeLayer(layer)` entfernt. Danach wird eine neue Ebene mit der gewünschten Option erstellt – `layer = L.esri.basemapLayer(basemap)` – und diese wird dann zum Kartenobjekt hinzugefügt `mymap.addLayer(layer)`.

Im Ergebnis können Sie sich mithilfe der Auswahlliste alle ESRI Karten ansehen. Oder Sie können es einem Website-Besucher ermöglichen die Karte, die dieser am besten findet, für seine Ansicht zu aktivieren.



# Shapefiles

Das Dateiformat Shapefile wurde von der Firma ESRI für die Software ArcView entwickelt. Shape Dateien sind speziell für die Verarbeitung von Geodaten entwickelt worden.

## Was genau verbirgt sich hinter dem Begriff Shapefile

Ein Shapefile ist keine einzelne Datei. In der Regel handelt es sich um mehrere Dateien, die zu einem Zip-Archiv zusammengefasst sind. In diesem Zip-Archiv müssen mindestens die drei nachfolgend genannten Dateitypen vorhanden sein:

- Eine Datei mit der Endung .shp dient zur Speicherung der Geometriedaten.
- Eine Datei mit der Endung .dbf enthält Attribute oder Eigenschaften.
- Eine Datei mit der Endung .shx dient als eine Art Index. Mithilfe dieser Datei sind die Geometriedaten in der Datei .shp mit den Eigenschaften in der .dbf-Datei verknüpft.

Weitere Informationen zu Shapefiles und wie Sie diese selbst erstellen können, finden Sie auf der Website <http://desktop.arcgis.com/de/arcmap/10.3/manage-data/shapefiles/creating-a-new-shapefile.htm>.

## Wie kommen Sie an ein Shapefiles

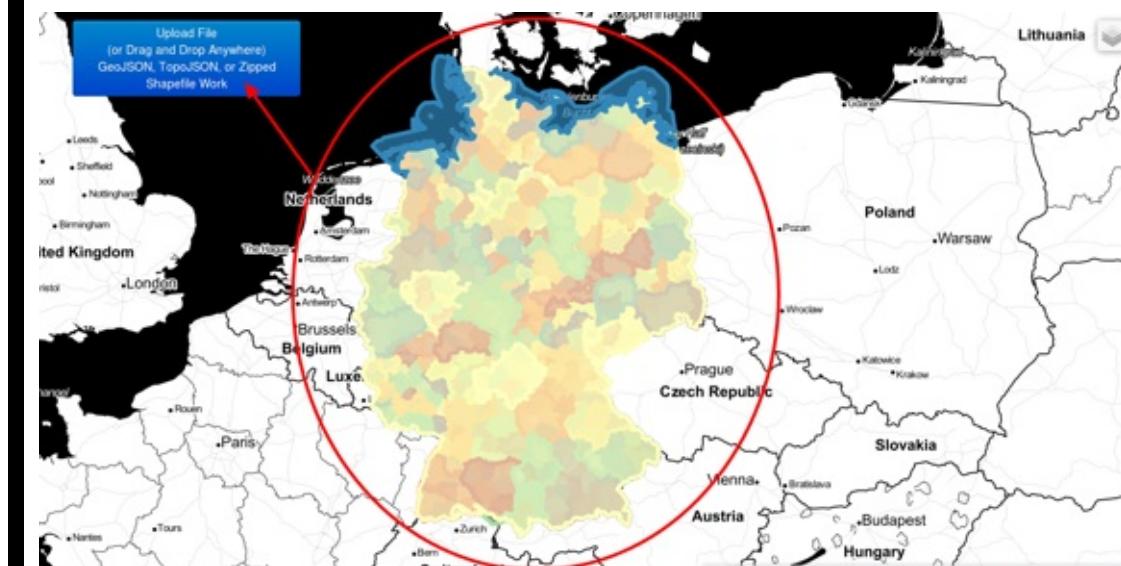
Natürlich können Sie selbst ein Shapefile erstellen. Das ist aber gar nicht einfach und außerdem nicht Thema dieses Buches. Einfach ist es aber, an Shapefiles von anderen

zu kommen. Insbesondere Daten, die von öffentlichem Interesse sind, werden oft zum Download als Shapefile angeboten. Suchen einfach einmal in einer Suchmaschine nach Websites, die den Text *Open Data* enthalten.

Bei einer Suche nach *Open Data* in einer Suchmaschine im Internet habe ich unter anderem das Schienennetz der Deutschen Bahn als Shapefile gefunden:  
<http://data.deutschebahn.com/dataset/geo-strecke>. Höchstwahrscheinlich interessieren Sie sich für Daten in der Nähe Ihres Wohnortes. Dann geben Sie zum Suchbegriff *Open Data* einfach den Namen einer Stadt in der Nähe Ihres Wohnortes ein. Eine Suche nach *Open Data* in der Nähe von *Koblenz* hat mich zu einer Website des Landesvermessungsamtes Koblenz geführt. Die Website [https://lvermgeo.rlp.de/fileadmin/lvermgeo/pdf/open-data/Download\\_von\\_Verwaltungsgrenzen.pdf](https://lvermgeo.rlp.de/fileadmin/lvermgeo/pdf/open-data/Download_von_Verwaltungsgrenzen.pdf) informiert über das Angebot des Landesvermessungsamtes - insbesondere über das Angebot an offenen Daten. Einige dieser Daten, zum Beispiel die Flurgrenzen, stehen als Shapefile zur Verfügung.

Wenn Sie per Internetsuche keine passenden Daten finden, können Sie auf der Website von Openstreetmap, insbesondere auf der Unterseite <http://wiki.openstreetmap.org/wiki/Shapefiles> nach weiteren Anbietern suchen. Neben weiteren Informationen zum Shapefile Format gibt es hier auch eine Liste mit Anbietern von Shapefiles.

Es muss nicht an Ihnen liegen, wenn Sie bei der Arbeit mit Geodaten auf einen Fehler stoßen. Insbesondere dann nicht, wenn Sie Daten von anderen in Ihre Arbeit einbinden. Da das Shapefile Format kompliziert ist, ist es gut, dass es im Internet Websites gibt, auf denen man eine Shapefile Datei testen kann. Eine dieser Websites ist [calvinmetcalf.com](http://calvinmetcalf.com).



## Wie binden Sie Shapefiles in Ihre Leaflet Karte ein?

**Deutsche Verwaltungsgrenzen als Shapefile in der Leaflet Karte**

Die *Shape Datei*, die ich im nächsten Beispiel verwende, habe ich von der Website <https://www.arcgis.com/home/item.html?id=ae25571c60d94ce5b7fcbf74e27c00e0> kopiert. Die Firma ESRI bietet auf dieser Website mit ArcGIS ein Geoinformationssystem, dass die Verwendung, Erstellung und Freigabe von Geodaten einfach macht. Ich habe mir den oben verlinkten Datensatz mit den Verwaltungsgrenzen von Deutschland kopiert. Die Datei landet unter dem Namen `vg2500_geo84.zip` bei mir auf dem Computer.

ArcGIS Features Pläne Galerie Karte Szene Hilfe Anmelden

## Verwaltungsgrenzen Deutschland (De, Länder, Rgbz, Kreise)

**Übersicht**

Datensatz mit 4 Shape Dateien mit den Verwaltungsgrenzen von Deutschland  
von moosmeier Zuletzt geändert: 28. Dezember 2011 **Herunterladen**

 Shapefile

**Beschreibung**

Datensatz mit 4 Shape Dateien mit den Verwaltungsgrenzen von Deutschland.  
Enthält:

- Deutschland
- Bundesländer
- Regierungsbezirke (soweit vorhanden)
- Landkreise

**Details**

★ ★ ★ ★ (2) Aufrufe: 20.709 Erstellt: 27. Dezember 2011 Größe: 1 MB [Facebook](#) [Twitter](#)

**Besitzer**  
moosmeier

Wie ich die Shape Datei mit Leaflet in eine Karte integriere, zeigt Ihnen das nachfolgende Beispiel. Natürlich gibt es dafür auch wieder ein Leaflet Plugin. Die Website zum Plugin Leaflet.Shapefile finden Sie auf Github unter der Adresse <https://github.com/calvinmetcalf/leaflet.shapefile>. Die Datei zum Plugin heißt `leaflet.shpfile.js`. Der gleiche Entwickler bietet auch ein Skript zum Analysieren der Shape Datei unter der Adresse <https://github.com/calvinmetcalf/shapefile-js> an. Diese Datei heißt `shp.js`. Diese beiden Dateien müssen Sie in Ihre HTML-Datei einbinden.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>

<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>

</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 6);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var shpfile = new L.Shapefile('vg2500_geo84.zip');
```

```

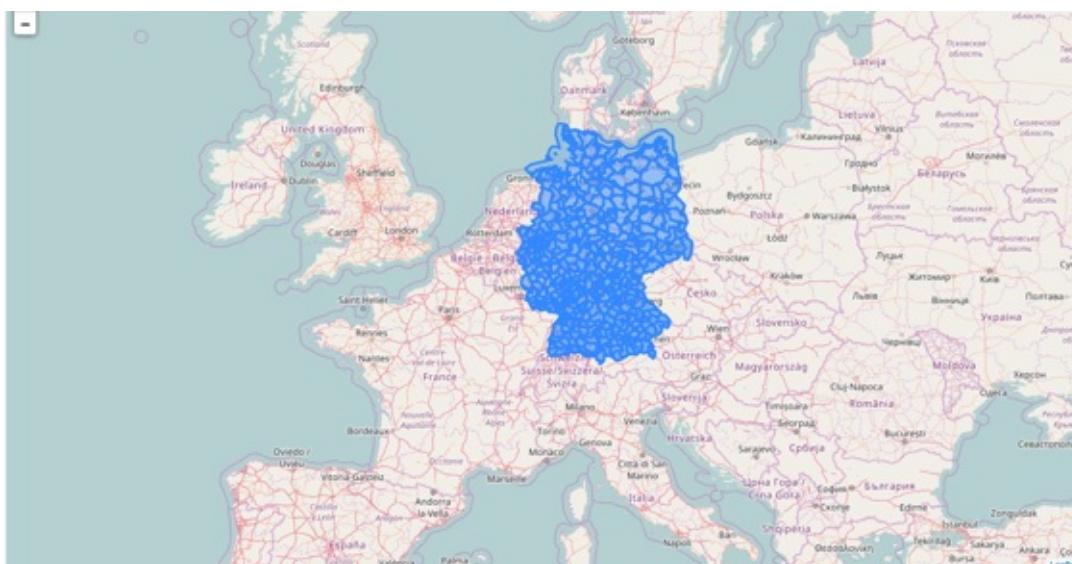
shpfile.addTo(mymap);

</script>
</body>
</html>
<!--index_943.html-->

```

Die vier fett formatierten Zeilen zeigen schon alles Notwendige. Und was haben wir genau gemacht? Im Kopfbereich unserer Beispiel HTML-Datei haben wir zwei Skripte eingebunden. Einmal das Skript `leaflet.shpfile.js` zum Leaflet Plugin und dann das Skript `shp.js` zum Analysieren der Shape Datei. Danach haben wir mit der Zeile `var shpfile = new L.Shapefile('vg2500_geo84.zip')` eine Ebene mit den Daten der Shape Datei erstellt. Zum Schluss haben wir diese Ebene zum Kartenobjekt hinzugefügt: `shpfile.addTo(mymap)`.

Als Ergebnis sehen Sie nun alle Verwaltungsgrenzen, also die reinen Geometrien. Eigenschaften, die in der Shape Datei enthalten sind, werden in diesem Beispiel noch nicht angezeigt. Das Anzeigen von Eigenschaften und das Zuordnen anderer Stile habe ich für das nächste Beispiel aufgehoben.



## Deutsche Verwaltungsgrenzen mit Optionen

Im nächsten Beispiel sehen Sie, wie Sie das Aussehen der Formen in der Shape Datei beeinflussen können.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="shp.js">
</script><script src="leaflet.shpfile.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>

```

```

<script>
var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 6);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);

var shpfile = new L.Shapefile(
'vg2500_geo84.zip',
{style:function(feature){return {
color:"black",fillColor:"red",fillOpacity:.75}}}
);
shpfile.addTo(mymap);

</script>
</body>
</html>
<!--index_942.html-->

```

Das vorhergehende Beispiel manipuliert die Option style. Die Belegung der Option style mit color:"black", fillColor:"red", fillOpacity:.75 färbt die Grenzen der Shape Datei schwarz und füllt die Formen mit der Farbe Rot – bei einer Transparenz von 75 Prozent.

Das rot gefärbte Deutschland können Sie sich in der nächsten Abbildung ansehen.



## Ein Pop-up-Fenster

Schön wäre es, wenn man für jede Verwaltungseinheit den Namen in einem Pop-up Fenster angezeigt bekommen würde – und zwar genau dann, wenn man die Fläche auf der Karte anklickt. Im nächsten Beispiel sehen Sie, wie Sie dies umsetzen können.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />

```

```

<script src="../leaflet/leaflet.js"></script>
<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 5);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var shpfile = new L.Shapefile(
'vg2500_geo84.zip',
{onEachFeature:function(feature, layer)
{layer.bindPopup(feature.properties.GEN);}}
);
shpfile.addTo(mymap);

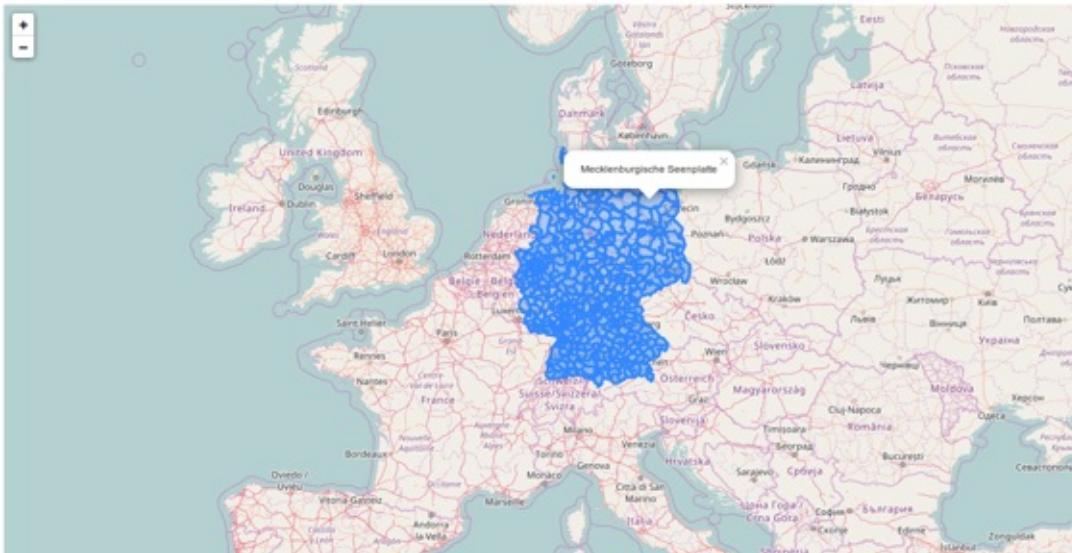
</script>
</body>
</html>
<-- index_941.html -->

```

In diesem Beispiel haben wir die Option `onEachFeature` verwendet. Wir haben dieser Option eine anonyme Funktion zugeordnet in der wir jedem Feature in der Shape Datei mithilfe von `layer.bindPopup(feature.properties.GEN)` ein Pop-up Fenster zugewiesen haben. Im Pop-up Fenster erscheint der Text, der bei dem Feature im Attribut `GEN` als Eigenschaft eingetragen ist. Wenn Sie sich nun fragen, woher Sie wissen können, dass es als Eigenschaft ein Attribut `GEN` gibt, dann warten Sie bis zum nächsten Beispiel. Dort werde ich Ihnen zeigen, wie Sie alle Optionen auslesen können.

Probieren Sie aber zunächst dieses Beispiel selbst aus. Öffnen Sie die Karte und klicken Sie einen Landkreis an. Wie in der nachfolgenden Abbildung zu sehen ist, sollte sich auch auf Ihrer Karte ein Pop-up Fenster öffnen.

Auf diese Art und Weise können Sie auch das Aussehen einer Form, abhängig vom Wert eines Attributes gestalten.



## Ein Pop-up-Fenster mit allen Informationen des Features

Sie wissen nicht, welche Eigenschaften in der Shape Datei genau für die Geometrien enthalten sind – Sie möchten sich aber gerne einen Überblick über diese Eigenschaften verschaffen. Das nächste Beispiel zeigt Ihnen, wie Sie die Informationen nur mit JavaScript in Erfahrung bringen können.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 5);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

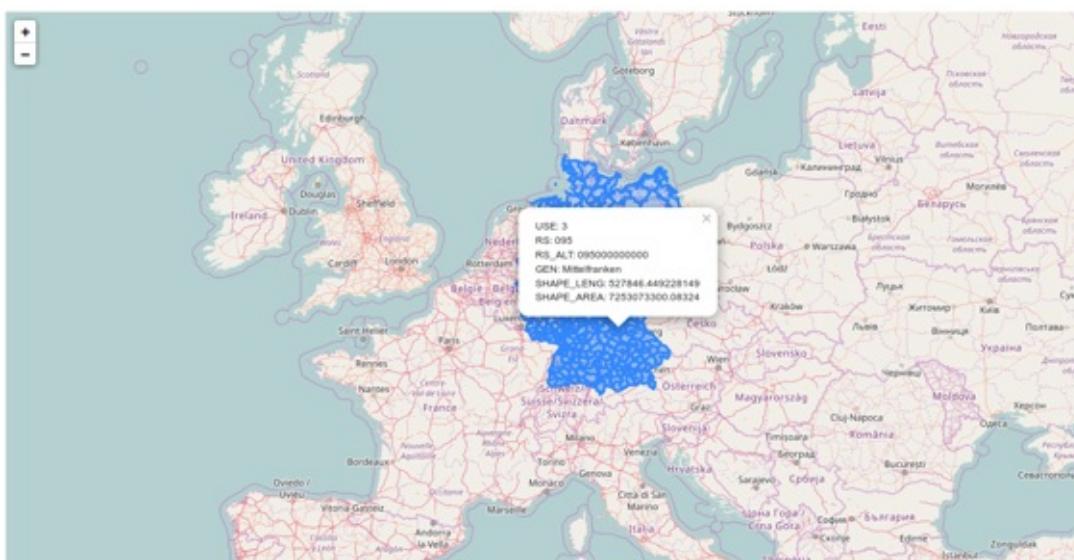
var shpfile = new L.Shapefile(
'vg2500_geo84.zip',
{
onEachFeature:function(feature, layer)
{var holder=[];for (var key in feature.properties){
holder.push(key+": "+feature.properties[key]+"<br>");
popupContent=holder.join("");
}

layer.bindPopup(popupContent);
}});

shpfile.addTo(mymap);
</script>
</body>
</html>
```

Für die Beantwortung dieser Fragestellung nutzen wir auch wieder die Option `onEachFeature`. In dieser Option erstellen wir eine Variable, genau ein Array, mit dem Namen `holder`. Als nächstes durchlaufen wir in einer Schleife alle Schlüssel, die in den Eigenschaften der Shape Datei, also in `feature.properties` enthalten sind. Wir fügen mit `push()` jeden Eintrag in den Array `holder` ein und separieren diesen von dem nächsten Eintrag mit `join()`. Den Text mit allen Schlüssel-Wert-Paaren fügen wir am Schluss als Pop-up Text zum Layer hinzu.

Als Ergebnis sehen Sie alle möglichen Schlüssel-Wert-Paare in einem Pop-up Fenster über Ihrer Karte, sobald Sie eine Fläche anklicken.



## ESRI Services

Sie wissen nun wie Sie eine ESRI Basiskarte laden, wie Sie mit dem Plugin ESRI-Leaflet grundsätzlich umgehen und wie Sie Dateien im Format Shapefile nutzen können.

ESRI bietet aber noch eine ganze Menge mehr. Sie können ArcGIS Server-Web-Services nutzen. Das ESRI Plugin unterstützt Sie auch bei der Verbindung mit einem Web Service, der sich auf einer ArcGIS Server-Website befindet und für Client-Anwendungen zur Verfügung gestellt wird. Mit einem ArcGIS Web Service kann entweder eine weitere Ebene für eine Karte zur Verfügung gestellt oder eine bestimmte Aufgabe bearbeitet werden.

Eine Ebene kann dabei neben dem `L.esri.BasemapLayer`, den Sie ja schon kennengelernt haben, ein `L.esri.DynamicMapLayer`, ein `L.esri.ImageMapLayer`, ein `L.esri.RasterLayer`, ein `L.esri.TiledMapLayer`, ein `L.esri.FeatureLayer`, ein `L.esri.Cluster.FeatureLayer` oder ein `L.esri.Heat.FeatureLayer` sein. Eine

Aufgabe könnte das Einschränken der Anzeige auf bestimmte Objekte mit `L.esri.Query` sein oder das Zuordnen von Adressen zu Koordinaten mit `L.esri.Geocoding`.

Im nächsten Kapitel beginnen wir damit, dass wir eine Ebene, nämlich einen `L.esri.DynamicMapLayer`, mithilfe eines Web Services nutzen. Wenn Sie dieses Beispiel durchgearbeitet haben, können Sie alle anderen Layer ebenfalls verwenden. Die Vorgehensweise ist bei jedem einheitlich. Der einzige Unterschied besteht im Angebot der möglichen Methoden und Optionen. Da aber die Dokumentation zum Plugin sehr gut ist, dürfte dies keine Schwierigkeit darstellen. Die Dokumentation zu den Layern finden Sie unter der Adresse <http://esri.github.io/esri-leaflet/api-reference/#layers>.

## **L.esri.DynamicMapLayer**

Für das nachfolgende Beispiel habe ich Daten des Geoportals Köln genutzt. Dieses Portal bietet den Zugriff auf offene Daten, Dienste und Anwendungen verschiedener Herkunft. Das Geoportal Köln zentralisieren nach eigenen Angaben Informationen aus den Bereichen Umwelt, Verkehr, Vermessung und Planung. Jeder hat die Möglichkeit die Geodaten des Geoportal Köln zu nutzen, anzusehen und zu analysieren. Informationen zum Geoportal Köln finden Sie auf der Website der Stadt Köln, genau unter der Adresse <http://www.stadt-koeln.de/politik-und-verwaltung/geoportal/>. Die Dokumentation zur Schnittstelle des Web Services finden Sie unter der Adresse <https://geoportal.stadt-koeln.de/arcgis/>. Wenn Sie dann auf den Eintrag *Behindertenparkplätze* klicken sehen Sie die Angaben, die speziell unser nachfolgendes Beispiel betreffen.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>

<script src="esri-leaflet.js"></script>

</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.97264, 7.00469], 12);

//L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
//.addTo(mymap);
L.esri.basemapLayer('Gray').addTo(mymap);

var url = "https://geoportal.stadt-koeln.de/arcgis
/rest/services/Behindertenparkplaetze/MapServer";

var dMap = L.esri.dynamicMapLayer({
```

```

url: url,
opacity : 0.25,
useCors: false
}).addTo(mymap);

dMap.bindPopup(
function(err, featureCollection, response){
return featureCollection.features[0]
.properties.Bezeichnung +
'<br>Anzahl: ' +
featureCollection.features[0].properties.Anzahl;
});

</script>
</body>
</html>
<!--index_939.html-->

```

Als Erstes haben wir im vorherigen Beispiel das Skript zum Esri Leaflet Plugin eingebunden. Als Nächstes haben wir die URL, unter welcher der betreffende Service angeboten wird, in die Variabel `url` gespeichert. Danach haben wir mit dem Code

```

var dMap = L.esri.dynamicMapLayer({
url: url,
opacity : 0.25,
useCors: false
}).addTo(mymap);

```

ein `L.esri.dynamicMapLayer`-Objekt erstellt und diesem gleichzeitig Optionen übergeben. Unter anderem die URL. Zuletzt haben wir dann mit dem Code

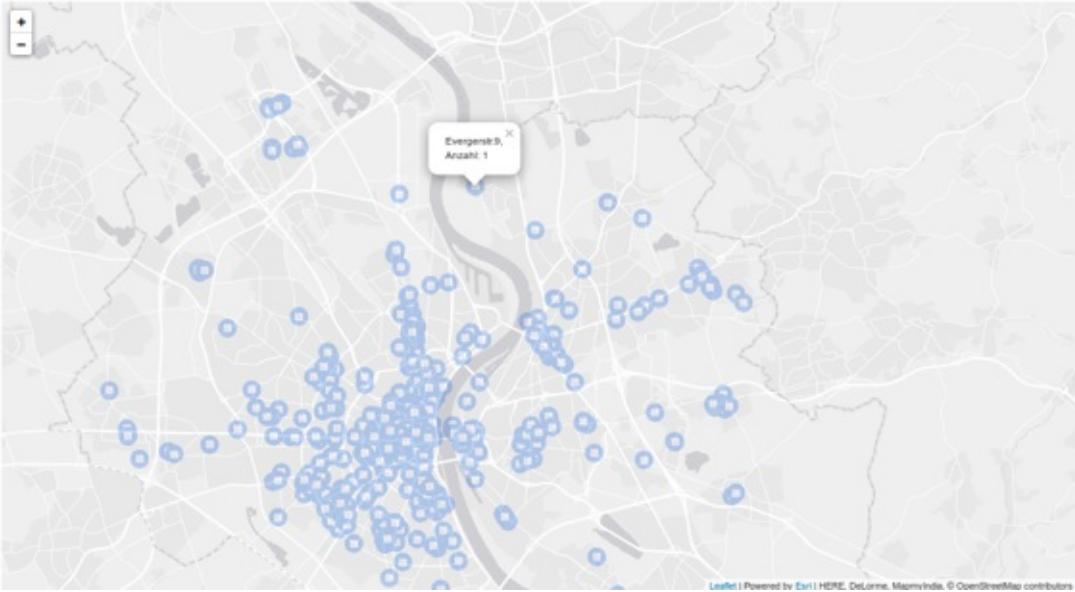
```

dMap.bindPopup(
function(err, featureCollection, response){
return featureCollection.features[0].properties.Bezeichnung +
'<br>Anzahl: ' +
featureCollection.features[0].properties.Anzahl;
});

```

jedem Feature ein Pop-up hinzugefügt und in dieses Pop-up die Eigenschaften Bezeichnung und Anzahl als Text eingetragen. Welche Eigenschaften Ihnen ein Web Service zur Verfügung stellt, finden Sie in der Dokumentation zum jeweiligen Service. Die Adresse zur Dokumentation des hier verwendeten Services hatte ich Ihnen weiter oben schon genannt.

Die nachfolgende Abbildung zeigt Ihnen alle – im Web Service Behindertenparkplätze des Geoportals Köln eingetragenen – Parkplätze.



## Geocoding

Was ist Geocoding? Geocoding bezeichnet die Umwandlung von Adressen wie *Kirchstraße 13, 56571 Muster* in geografische Koordinaten wie *50.423021 Breite und 7.083739 Länge*. Nur so können Sie nämlich einen Marker auf einer Karte platzieren oder die Karte an einer bestimmten Adresse zentriert öffnen.

Nicht nur ESRI bietet Geocoding Dienste an. Auch OpenStreetMap bietet Ihnen diesen Service. Nominatim ist eine Anwendung, mithilfe derer OpenStreetMap Daten anhand von Texten durchsucht werden können. Diese Texte sind in der Regel eine Adresse oder ein Teil einer Adresse. Falls der Suchtext mit einer Eigenschaften eines OpenStreetMap-Objektes, übereinstimmt, wird die Koordinate zu diesem Objekt auf eine Suchanfrage hin zurückgegeben. Sie finden die Anwendung Nominatim unter der Adresse <https://nominatim.openstreetmap.org/> im Internet.

Da Nominatim auf OpenStreetMap Daten beruht, können Sie sich nicht darauf verlassen, dass alle Adressen eingetragen sind. OSM ist keine kommerzielle Firma, sondern ein Projekt das auf der Mitarbeit von Freiwilligen beruht. Dafür können Sie aber selbst Einfluss auf den Datenbestand nehmen. Bei OpenStreetMap kann jeder mitmachen und korrekte Daten zu dem aktuellen Datenband hinzufügen.

Ein Plugin, dass Nominatim als Service nutzt, ist das Plugin Leaflet Control Geocoder.

Ich zeige Ihnen in diesem Kapitel anhand des ESRI Leaflet Geocoder Plugins folgendes:

- Ich zeige Ihnen, wie Sie eine Adresse in eine Koordinate umwandeln können.
- Ich zeige Ihnen, wie Sie eine Koordinate in eine Adresse umwandeln können.
- Ich zeige Ihnen, wie Sie eine Karte so öffnen können, dass eine bestimmte

Adresse zentriert angezeigt wird.

## Nach Eingabe einer Adresse den passenden Ort auf der Karte finden

Sie zeigen eine Karte auf Ihrer Website an und möchten es anderen ermöglichen, zu einer bestimmten Adresse auf der Karte zu navigieren, nachdem Sie diese Adresse in einem Textfeld eingetragen haben. Dann integrieren Sie doch einfach ein ESRI Leaflet Geocoder Control in Ihre Karte. Wie Sie dies tun können zeigt Ihnen das nächste Beispiel.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
<link rel="stylesheet" href="esri-leaflet-geocoder.css">
<script src="esri-leaflet-geocoder.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.97264, 7.00469], 12);
L.esri.basemapLayer('Gray').addTo(mymap);

var searchControl = L.esri.Geocoding.geosearch()
.addTo(mymap);
var results = L.layerGroup().addTo(mymap);
searchControl.on("results", function(data) {results.clearLayers();
for (var i = data.results.length - 1; i >= 0; i--){
results.addLayer(L.marker(data.results[i].latlng));
}
});

</script>
</body>
</html>
<!--index_938.html-->
```

Zur Darstellung des Textfeldes für die Eingabe der Adresse ist es notwendig zwei weitere Skripte einzubinden. Neben dem Skript esri-leaflet.js sollten Sie auch noch die Dateien esri-leaflet-geocoder.css und esri-leaflet-geocoder.js in Ihr HTML-Dokument einbinden. Die notwendigen Dateien hierfür finden Sie unter der Adresse <https://github.com/Esri/esri-leaflet-geocoder/>. Als Nächstes wird dann mit var searchControl = L.esri.Geocoding.geosearch().addTo(mymap); das Eingabefeld für die zu suchende Adresse erstellt und zur Karte hinzugefügt. Dieses Eingabefeld nennt Leaflet auch Control. Wir brauchen eine Ebene, die die Suchergebnisse richtig verarbeitet. Diese erstellen wir mit der Zeile var results = L.layerGroup().addTo(mymap);. Zum Schluss können wir dann das Ereignis on("results") des Objektes searchControl mit einer Funktion belegen.

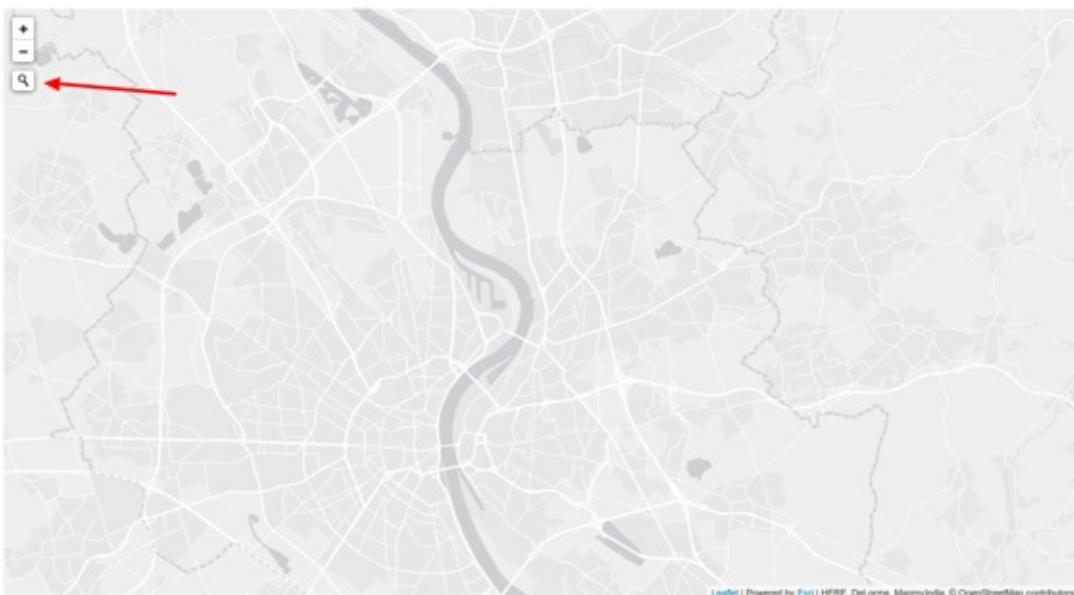
```
searchControl.on("results", function(data) {
```

```
results.clearLayers();
for (var i = data.results.length - 1; i >= 0; i--) {
  results.addLayer(L.marker(data.results[i].latlng));
}
});
```

Wenn es neue Suchergebnisse gibt, dann werden alle alten Suchergebnisse von der Ebene entfernt und die neuen Suchergebnisse als Marker auf der Ebene platziert.

Ihre Adresse wird nicht auf der Karte gefunden? Das ESRI Leaflet Geocoder Control findet mithilfe der Methode `geosearch()` standardmäßig nur Adressen, die aktuell auch sichtbar auf der Karte sind. Erst wenn Ihre sicher vorhandene Adresse auch bei einer Zoom-Stufe von 0 nicht gefunden wird, stimmt etwas nicht. Falls Sie lieber sofort die ganz Welt durchsuchen möchten, dann sollten Sie die Option `useMapBounds` des Objektes, das Ihnen die Funktion `L.esri.Geocoding.geosearch()` liefert, mit `false` belegen.

Die erste Ansicht der Karte nach dem Hinzufügen des ESRI Leaflet Geocoder Control Plugins ist nicht spektakulär. Sie sehen in der linken oberen Ecke eine kleine Lupe. Wenn Sie diese anklicken, öffnet sich ein Textfeld. In dieses Textfeld können Sie nun die Adresse eingeben, nach der Sie suchen möchten. Standardmäßig werden Ihnen alternative Eingaben vorgeschlagen. Sie können das Verhalten des Plugins mit vielen Optionen beeinflussen. Sehen sich die Dokumentation unter Adresse <http://esri.github.io/esri-leaflet/api-reference/controls/geosearch.html> an. Vielleicht ist da sogar eine Option dabei, auf die Sie selbst nicht gekommen wären.



## Mithilfe eines Parameters in der URL den passenden Ort finden

Manchmal kommt es vor, dass Sie eine Karte schon an einer bestimmten Adresse zentriert öffnen möchten. Wie Sie so etwas umsetzen können, zeigt Ihnen das nächste Beispiel.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
<script src="esri-leaflet-geocoder.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>

var x = location.search;var y = x.split("=");var temp=y[1];
var address = decodeURIComponent(temp);

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.97264, 7.00469], 3);

L.esri.basemapLayer('Gray').addTo(mymap);

L.esri.Geocoding.geocode().text(address)
.run(function(err, result, response){
L.marker(result.results[0].latlng).addTo(mymap);
mymap.setView(result.results[0].latlng, 13);
});

</script>
</body>
</html>
<!--index_937.html-->

```

Was haben wir genau gemacht? Angenommen, wir haben in der Adresszeile des Browsers den Text ?test=56751 Gering an die URL angefügt. Dieser Text würde im Beispiel als Erstes in der Variablen x gespeichert. Hierzu wird die Eigenschaft location.search zu Hilfe genommen. Diese Eigenschaft speichert eine Zeichenkette die, durch ein Fragezeichen getrennt, zur aktuellen URL gehört. Die Variable x enthält also nun den Text test=56751 Gering. Diesen Text haben wir dann mithilfe von x.split ("=") an der Position des Gleichheitszeichens getrennt und die beiden Teile als Array in der Variablen y gespeichert. Den Adressteil können wir nun über y[1] abrufen. Wir müssen die Adresse vor der Eingabe in eine Suchfunktion aber noch dekodieren. In der URL wurde diese nämlich kodiert. var address = decodeURIComponent(temp) macht aus 56751%20Gering wieder 56751 Gering und speichert den dekodierten Text in der Variablen address ab. Und diesen Text können wir nun als Suchtext in die Methode text() des Geocode Objektes eingeben und auf dem Ergebnis die Methode run() ausführen.

```

L.esri.Geocoding.geocode().text(address)
.run(function(err, result, response){
L.marker(result.results[0].latlng)
.addTo(mymap);
<br /> mymap.setView(result.results[0].latlng,13);
});

```

Die Rückruffunktion, die daraufhin ausgeführt wird, erstellt einen Marker für das

erste Ergebnis und zentriert diesen Marker, bei einer Zoom-Stufe von 13, in der Karte. Eine Rückruffunktion ist eine Funktion, die einer anderen Funktion als Parameter übergeben und von dieser erst später, unter definierten Bedingungen mit definierten Argumenten, aufgerufen wird. Eine solche Funktion kennen Sie sicher auch unter dem englischen Namen **callback function**.

Möchten Sie die Suchanfrage genauer formulieren? Dann ersetzen Sie den Text

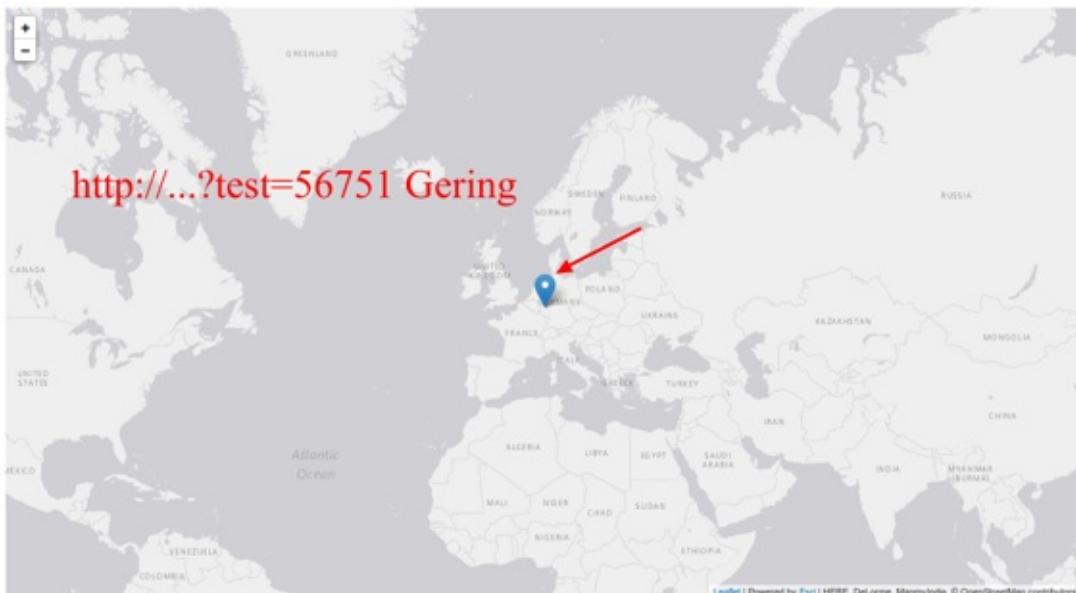
```
L.esri.Geocoding.geocode()  
.text('Amselweg 7, Koblenz, Rheinland Pfalz, 65065')  
.run(function(err, results, response){  
...  
});
```

mit

```
L.esri.Geocoding.geocode()  
.address('Amselweg 7')  
.city('Koblenz').region('Rheinland Pfalz')  
.postal(65065)  
.run(function(err, results, response){  
...  
});
```

Welche Funktionen Sie genau verwenden können, ist in der Dokumentation beschrieben:  
<https://esri.github.io/esri-leaflet/api-reference/tasks/geocode.html>.

Voilà! Wenn Sie an die URL, unter der Sie Ihre Leaflet Karte abgelegt haben, den Text `?test=56751 Gering` anhängen, wird Ihnen die in der nachfolgenden Abbildung dargestellte Karte angezeigt.



## Nach Klick auf die Karte die passende Adresse ausgeben

Umgekehrtes Geocoding ist die Bezeichnung für die Umwandlung geografischer

Koordinaten in, für Menschen gut lesbare, Adressen.

Sie wissen ja schon, dass Leaflet bei einem Klick mit der Maus auf die Karte die Koordinaten in einem Objekt speichert. Nun möchten Sie aber gerne anstelle der Koordinaten wissen, welche Adresse Sie gerade angeklickt haben. Das nächste Beispiel zeigt Ihnen, wie Sie dies erreichen können.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
<script src="esri-leaflet-geocoder.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.97264, 7.00469], 12);

L.esri.basemapLayer('Gray').addTo(mymap);

mymap.on('click', function(e){

var r = L.marker();

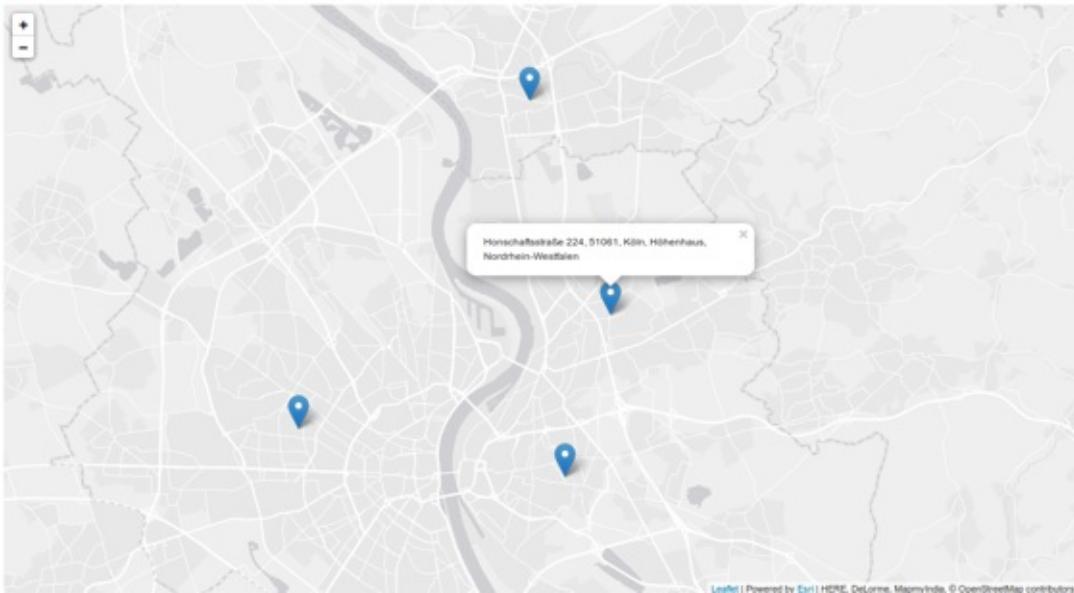
L.esri.Geocoding.reverseGeocode()
.latlng(e.latlng)
.run(function(error, result, response){
r = L.marker(result.latlng).addTo(mymap)
.bindPopup(result.address.Match_addr)
.openPopup();
});

});

</script>
</body>
</html>
<!--index_936.html-->
```

Was haben wir gemacht? Wir haben veranlasst, dass immer dann, wenn jemand auf die Karte klickt, eine umgekehrte Geocoding Suche gestartet wird. Genau habe wir dies mit dem Programmcode, der in der Funktion mymap.on('click', function(e) {}) ausgeführt wird, erreicht. Gleichzeitig wird ein Marker an der Position, die angeklickt wurde, erstellt. Das Ergebnis der Suche – also die gefundene Adresse – wird schlussendlich als Text in einem Pop-up zum Marker angezeigt.

Wie diese Karte aussieht, können Sie sich in der nächsten Abbildung ansehen.



Die Dokumentation zum ESRI Reverse Geocoding finden Sie unter der Internetadresse <http://esri.github.io/esri-leaflet/api-reference/tasks/reverse-geocode.html>

## Feature Services

Im vorletzten Kapitel haben wir mit dem `L.esri.dynamicMapLayer` Daten über einen Webservice geladen. Dabei wurden alle zur Verfügung stehenden Daten auf der Karte angezeigt. Das kann sehr unübersichtlich werden. Meist bietet ein Webservice große Datenmengen an. Sie möchten sicherlich nur die für Sie und Ihre Anwender relevanten Daten auf Ihrer Karte anzeigen. Eine Lösung für diese Aufgabe erarbeiten wir in diesem Kapitel.

In diesem Beispiel werden wir die Klasse `L.esri.featureLayer` instanzieren. Vorher haben wir mit der Klasse `L.esri.dynamicMapLayer` gearbeitet. Vielleicht fragen Sie sich nun, wie die beiden Klassen sich genau unterscheiden. ESRI erklärt dies wie folgt: Mit einem Feature-Service können Sie Funktionen über das Internet bereitstellen. Mit dem Karten-Service können Sie Karten im Web bereitstellen. Meiner Meinung nach ist ein *Feature Service* ein spezieller *Karten-Service* – denn ein Feature Service setzt eine Karte voraus.

Interessieren Sie sich für die technologische Entwicklung beim ESRI? Die wichtigsten Informationen zur ArcGIS Plattform und zu ergänzenden Technologien können Sie auf dem Blog GIS IQ – <https://gis-iq.esri.de/> – mitverfolgen.

## Attribute

So, nun kommen wir zum praktischen Beispiel. Wir konfigurieren eine Abfrage zum Filtern von Daten. Konkret verwenden wir Sicherheitseinrichtungen, deren Daten

über die Adresse <http://www.arcgis.com/home/item.html?id=59711f4ee839438299c90164115f129b> bereit gestellt werden.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
</head>
<body>
<select id="sicherheitseinrichtungenID">
<option value="">Reset</option>
<option value="Untertyp='pt_Notrufsaeule'">
Notrufsaeule</option>
<option value="Untertyp='pt_Feuerwache'">
Feuerwache</option>
<option value="Untertyp='pt_Rettungspunkt'">
Rettungspunkt</option>
<option value="Untertyp='pt_Polizeistation'">
Polizeistation</option>
<option value="Untertyp='pt_Sirene'">
Sirene</option>
<option value="Untertyp='pt_Rettungswache'">
Rettungswache</option>
</select>
<div style="height: 700px" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 12);

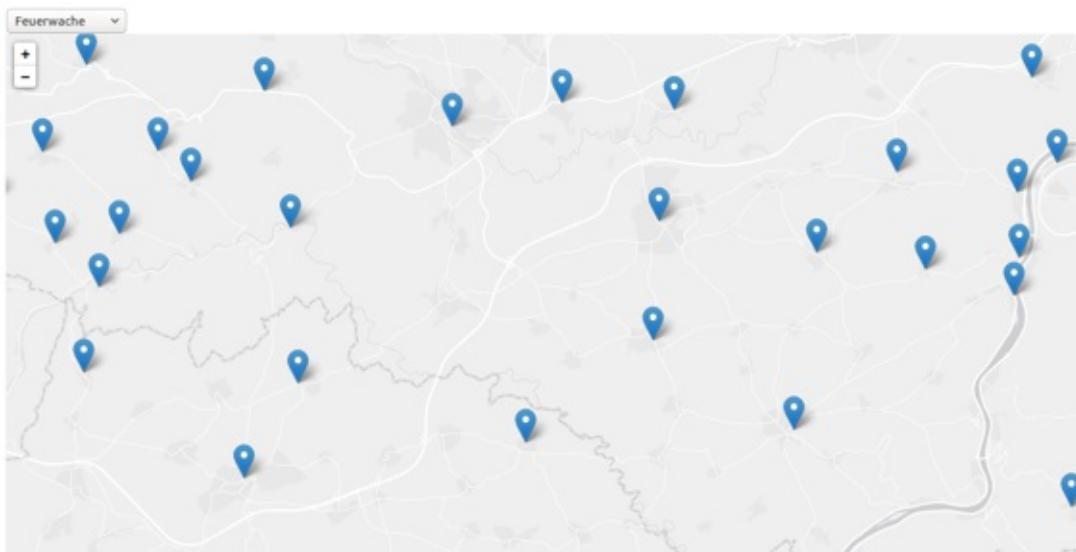
L.esri.basemapLayer('Gray').addTo(mymap);
var url = 'http://services.arcgis.com/OLiydejKCZTGhvWg
/arcgis/rest/services/OSM_DE_Sicherheitseinrichtungen
/FeatureServer/0';

var sicherheitseinrichtungen = document
.getElementById('sicherheitseinrichtungenID');

var sicherheitseinrichtungenLayer =
L.esri.featureLayer({
url: url
}).addTo(mymap);
var popupTemplate = "<h3>Details:</h3>\n\
Land: {Land}<br>\n\
Kreis: {Kreis}<br>\n\
Gemeinde: {Gemeinde}<br>\n\
Stadt: {Stadt} <br>\n\
Typ: {Typ}, {Untertyp}";
sicherheitseinrichtungenLayer
.bindPopup(function (layer){
return L.Util
.template(popupTemplate, layer.feature.properties)
});
sicherheitseinrichtungen
.addEventListener('change', function(){
sicherheitseinrichtungenLayer
.setWhere(sicherheitseinrichtungen.value);
});
</script>
</body>
</html>
<!--index_935.html-->
```

Was macht der Programmcode in diesem Beispiel konkret. Zunächst wird mit dem Befehl `var sicherheitseinrichtungenLayer = L.esri.featureLayer({ url: url }).addTo(mymap);` ein `L.esri.featureLayer` Objekt erstellt und dieses zur Karte hinzugefügt. Der nächste Schritt ist das Filtern der Daten. Dies erreichen wir mit der Zeile `sicherheitseinrichtungenLayer.setWhere(sicherheitseinrichtungen.value);`.

Das Ergebnis sehen Sie in der nächsten Abbildung. Wenn Sie in der Auswahlliste den Eintrag *Feuerwachen* auswählen, werden nur die Feuerwachen auf der Karte als Marker angezeigt.



Möchten Sie selbst einen Feature-Service nutzen? Das Thema Sicherheitseinrichtungen passt aber nicht zum Thema Ihrer Website. Das Team von Esri Deutschland hat verschiedene Datensätze aus OpenStreetMap als Feature-Services veröffentlicht. Vielleicht ist auf der Website <http://www.arcgis.com/home/group.html?id=0ef29288545f43e29d9a90581f8f5b20#overview> ein Thema dabei, dass Sie gerne auf Ihrer Website mit einer Leaflet-Karte visualisieren möchten.

## Abstand visualisieren

Ein weiteres Merkmal, das auf Landkarten oft wichtig ist, ist die Entfernung zu einem Punkt. Oft möchte man gerne wissen, welche Objekte sich innerhalb eines bestimmten Abstands zu einer bestimmten Koordinate befinden. Diese Fragestellung beantwortet das nächste Beispiel.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
</head>
<body>
```

```

<div style="height: 700px" id="mapid"></div>
<script>

var mymap = L.map('mapid', {doubleClickZoom:false})
.setView([50.27264, 7.26469], 12);

L.esri.basemapLayer('Gray').addTo(mymap);
var url = 'http://services.arcgis.com/OLiydejKCZTGhvWg
/arcgis/rest/services/OSM_DE_Sicherheitseinrichtungen
/FeatureServer/0 ';
var sicherheitseinrichtungen =
document.getElementById('sicherheitseinrichtungenID');
var sicherheitseinrichtungenLayer = L.esri.featureLayer({
url: url,
pointToLayer: function (geojson, latlng) {
return L.circleMarker(latlng);
},
style:{
color: '#5B7CBA',
}
}).addTo(mymap);
var popupTemplate = "<h3>Details:</h3>\n\
Land: {Land}<br>\n\
Kreis: {Kreis}<br>\n\
Gemeinde: {Gemeinde}<br>\n\
Stadt: {Stadt} <br>\n\
Typ: {Typ}, {Untertyp}";
sicherheitseinrichtungenLayer.bindPopup(function (layer){
return L.Util.template(
popupTemplate, layer.feature.properties)
});
var nearbyIds = [];

mymap.on('click', function(e){

sicherheitseinrichtungenLayer.query()
.nearby(e.latlng, 5000).ids(function(error, ids){

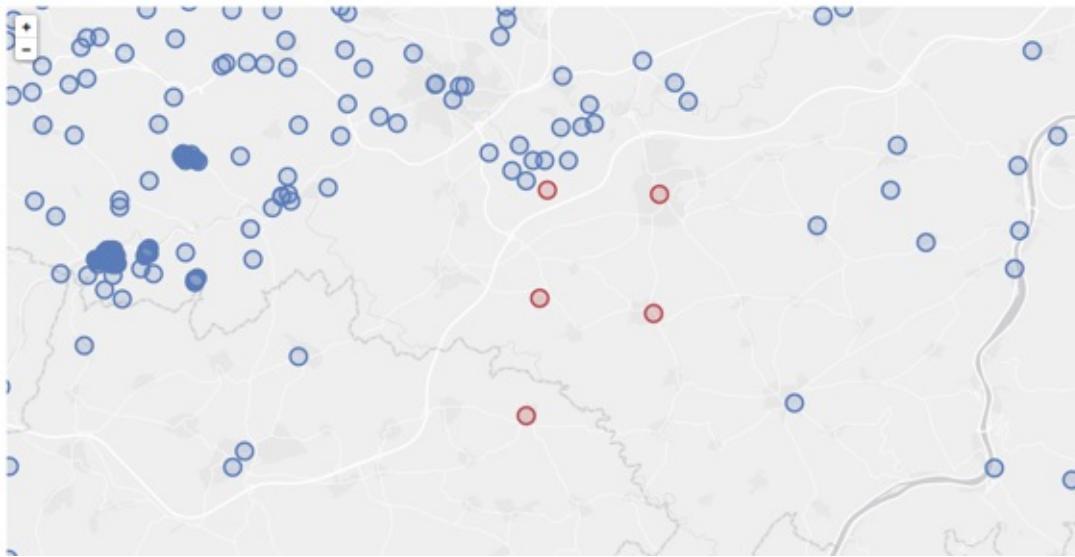
for (var j = 0; j < nearbyIds.length; j++) {
sicherheitseinrichtungenLayer.resetStyle(nearbyIds[j]);
}
nearbyIds = ids;
for (var i = 0; i < ids.length; i++) {
sicherheitseinrichtungenLayer.setFeatureStyle(ids[i], {
color: '#BA454E',
});
}
});

});

});
</script>
</body>
</html>
<!--index_934.html-->
```

Das haben wir gemacht? Zunächst werden im Beispiel Sicherheitseinrichtungen mithilfe des Objektes `L.esri.featureLayer` zur Karte hinzugefügt. Soweit gibt es keine Unterschied zum vorherigen Beispiel. Zur Abwechslung habe ich hier einen kreisförmigen Marker verwendet. Danach rufen wir die Funktion `nearby()` des Objektes `L.esri.Query()` für jeden Marker auf. Alle die Marker, die sich in einem Abstand von 5000 Metern befinden, werden im Anschluss rot gefärbt.

Das Ergebnis sehen Sie im nächsten Bild. Sobald Sie einen Punkt auf der Karte anklicken, werden alle Marker im Abstand von 5000 Metern um die angeklickte Stelle rot hervorgehoben.



## In diesem Kapitel haben wir ...

In diesem Teil haben wir uns als Erstes die Karten, die ESRI anbietet, angesehen. Danach haben wir mit einem Shapefile gearbeitet und ESRI Webservices genutzt. Sie wissen nun was ein `L.esri.DynamicMapLayer`, was Geocoding und was ein `L.esri.FeatureLayer` ist. Mit letzterem können Sie unter anderem Daten, die Sie nicht benötigen, einfach aus der gegebenen Datenmenge herausfiltern.

# Routing

Last, but not least möchte ich Ihnen ein Routing Plugin vorstellen, nämlich die Leaflet Routing Machine (<https://github.com/perliedman/leaflet-routing-machine>) von Per Liedmann.

Falls Sie lieber die Version des Environmental Systems Research Institute ESRI einsetzen möchten, finden Sie alle Informationen hierfür unter der Adresse <https://github.com/jgravois/lrm-esri> (<https://github.com/jgravois/lrm-esri>).

## In diesem Kapitel werden wir ...

Als Erstes sehen wir uns kurz an, was sich hinter dem Begriff Routing verbirgt. Danach werden wir das Plugin Leaflet Routing Machine in unsere Leaflet Karte integrieren. Dieses Plugin stellt zwei Textfelder zur Eingabe von Adressen zur Verfügung und berechnet auf Anforderung eine Route zwischen diesen beiden Adressen. Im weiteren Verlauf nutzen wir Optionen, um die Routenberechnung flexibler und benutzerfreundlicher zu machen.

## Allgemeines zum Thema Routing...

Fangen wir von vorne an. Es ist nämlich meiner Meinung nach auch spannend, einmal etwas über den Tellerrand zu blicken. Reicht es Ihnen, wenn sie zwei Adressen eingeben könnten und einen Weg zwischen diesen beiden Adressen als Ergebnis ausgegeben bekommen? Oder, möchten Sie etwas genauer wissen, was sich hinter dem Begriff Routing verbirgt? Wenn ersteres auf Sie zutrifft, dann lesen Sie am besten im nächsten Kapitel weiter. Ansonsten ist meine kurze Zusammenfassung hier im Kapitel vielleicht interessant für Sie.

Das Berechnen einer Route kann von einer Software oder einem Dienst effizient durchgeführt werden. Dazu müssen die Daten der Karte, auf der die Berechnung ausgeführt werden soll, bestimmte Informationen enthalten. Die Daten von OpenStreetMap beinhalten Informationen, um Routen für verschiedene Profile berechnen zu können. Eine Route kann zum Beispiel für die Nutzung eines Autos oder eines Fahrrads unterschiedlich berechnet werden. Auch Optionen wie zu Fuß oder mit dem Pferd sind mit OpenStreetMap Daten möglich.

Damit die Software zur Routenberechnung fehlerfrei funktioniert, müssen diese Daten qualitativ gut sein. Das bedeutet, dass Linien – also zum Beispiel Straßen – die verbunden sein sollen, auch wirklich verbunden sind. Umgekehrt dürfen Straßen, die sich auf verschiedenen Ebenen mithilfe von Brücken kreuzen, nicht verbunden sein.

Das bedeutet auch, dass Poller oder Absperrpfosten, an denen kein Auto vorbeikommt, auch als solche eingezeichnet sind. Ganz wichtig ist es, dass Abbiegeverbote an Kreuzungen auf der Karte klar gekennzeichnet sind. Wie OpenStreetMap das macht, können Sie unter anderem auf der Website [http://wiki.openstreetmap.org/wiki/OSM\\_tags\\_for\\_routing](http://wiki.openstreetmap.org/wiki/OSM_tags_for_routing) nachlesen.

Um bei der Routen-Berechnung die schnellste Route bestimmen zu können, sind Geschwindigkeitsangaben notwendig. OpenStreetMap speichert für jede Straße den Standardhöchstwert für deutsche Straßen. Dieser Standardwerte kann jedoch für bestimmte Straßen anders sein. In diesem Fall ist es notwendig, dass beim Erstellung des Straßen-Objektes die zulässige Höchstgeschwindigkeit hinzufügt wird. Dies geschieht mit einer speziellen Eigenschaft oder mit einem speziellen Tag. Nebenbei haben auch andere Faktoren, wie Kurven, Steigungen oder Straßenbelag einen Einfluss auf die Geschwindigkeit – und dieser ist je nach Profil noch einmal unterschiedlich. Eine Steigung verringert die Geschwindigkeit eines Fahrrades stärker, als die eines Autos. Dafür wird der Fußgänger in Kurven nicht so viel an Geschwindigkeit verlieren wie ein Fahrrad.

Aber auch dann, wenn die Kartendaten sehr gut und fehlerfrei sind, ist es nicht trivial die ideale Software zur Routen-Berechnung zu erstellen. Hinzu kommen nämlich noch die Besonderheiten des Personenkreises, für den die Route erstellt werden soll. Welche Wege sollen für Radfahrer berücksichtigt werden? Es gibt Radfahrer, die nicht gerne durch die Stadt fahren sondern lieber im Wald unterwegs ist. Andere ziehen eine asphaltierte Straße mitten durch die Stadt einem matschigen Feldweg vor. Wie wählen wir die beste Strecke für einen Radfahrer, der überwiegend in der Nacht trainiert? Oft kommt es sicher auch vor, dass eine Radtour geplant wird, bei der so viele Sehenswürdigkeiten wie möglich unmittelbar am Weg liegen sollten. Sie sehen: Die Berechnung einer optimalen Route ist etwas sehr Individuelles!

Sehen wir uns also an, inwieweit das Plugin Leaflet Routing Machine diese Anforderungen erfüllt.

## Leaflet Routing Maching

Nachfolgende finden Sie das erste Beispiel für diesen Themenbereich.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-routing-machine.js"></script>
<link rel="stylesheet" href="leaflet-routing-machine.css"/>
</head> <body> <div style="height: 700px;" id="mapid">
</div>
<script> var mymap = L.map('mapid').setView([50.27264, 7.26469], 13);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(mymap);
```

```

var control = L.Routing.control({ waypoints: [
L.latLng(50.273543, 7.262993), L.latLng(50.281168, 7.276211)
], router: L.Routing.mapbox('pk.ihrkey')
}).addTo(mymap);

</script>
</body>
</html>
<!--index_932.html-->
```

Was haben wir genau gemacht? Zunächst einmal haben wir die für die Verwendung der Leaflet Routing Machine erforderlichen Dateien eingebunden. Verantwortlich hierfür sind die Zeilen

```

<script src="leaflet-routing-machine.js"></script>

<link rel="stylesheet" href="leaflet-routing-machine.css"/>.
```

Die aktuellen Dateien zum Plugin finden Sie unter der Adresse <https://github.com/perliedman/leaflet-routing-machine/releases>. Als Nächstes haben wir mit dem Textblock

```

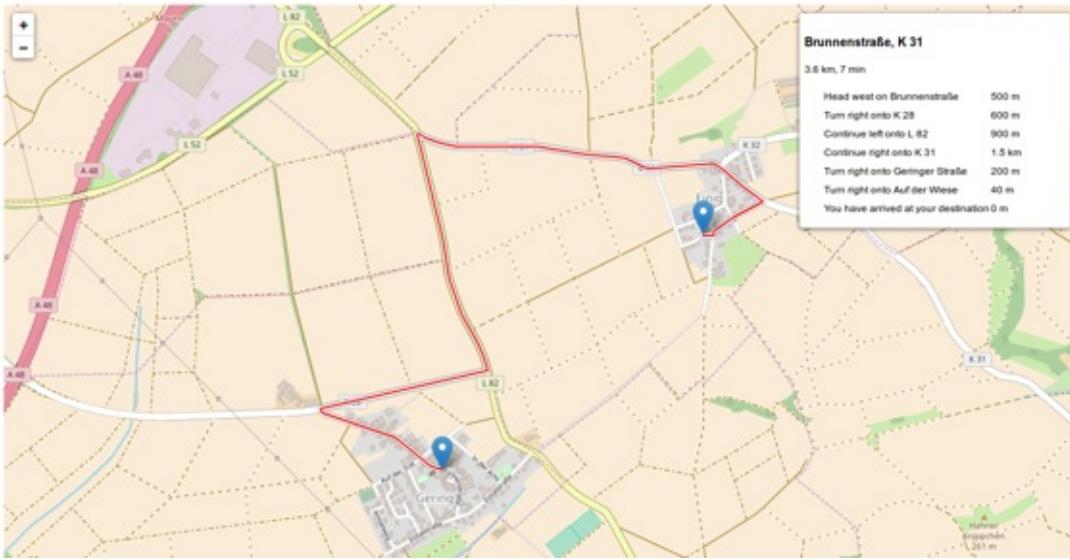
var control = L.Routing.control({
waypoints: [
L.latLng(50.273543, 7.262993),
L.latLng(50.281168, 7.276211)
],
router: L.Routing.mapbox('pk.ihrkey')
}).addTo(mymap);
```

ein `L.Routing.control` erstellt und dieses zur Karte hinzugefügt. Die Koordinaten, zwischen denen eine Route berechnet werden soll, haben wir mit der Option `waypoints` an das Control übergeben. Außerdem haben wir einen Router, also eine Software, die die Route berechnen soll, mit dem Eintrag `router`:

`L.Routing.mapbox('pk.ihrkey')` festgelegt. Für die Nutzung des Mapbox Routers benötigen Sie ein Zugriffstoken. Dieses können Sie unter der Adresse <https://www.mapbox.com/api-documentation/#access-tokens> anfordern. In dieser Dokumentation finden Sie auch weitere Informationen zum Router und zu den Lizenzbedingungen.

Wenn Sie die Option `router` nicht belegen, wird standardmäßig die Open Source Routing Machine – <http://project-osrm.org/> – genutzt. Für diesen freien Router benötigen Sie kein Zugriffstoken. Leider ist dieser aber nicht so zuverlässig wie Mapbox.

Die nächste Abbildung zeigt Ihnen unseren bisherigen Stand. Die Route wird berechnet und im rechten oberen Teil angezeigt. Leider werden Sie aber über eine Straße, auf der viele Autos fahren, geführt. Dabei möchten Sie gerne wandern. Außerdem sind die Texte in englischer Sprache. Das geht sicher besser!



## Options

Das nächste Beispiel zeigt Ihnen, wie Sie die Sprache und das Routing Profil verändern können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-routing-machine.js"></script>
<link rel="stylesheet" href="leaflet-routing-machine.css" />
</head> <body> <div style="height: 700px;" id="mapid">
</div>
<script>
var mymap = L.map('mapid')
.setView([50.27264, 7.26469], 13);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

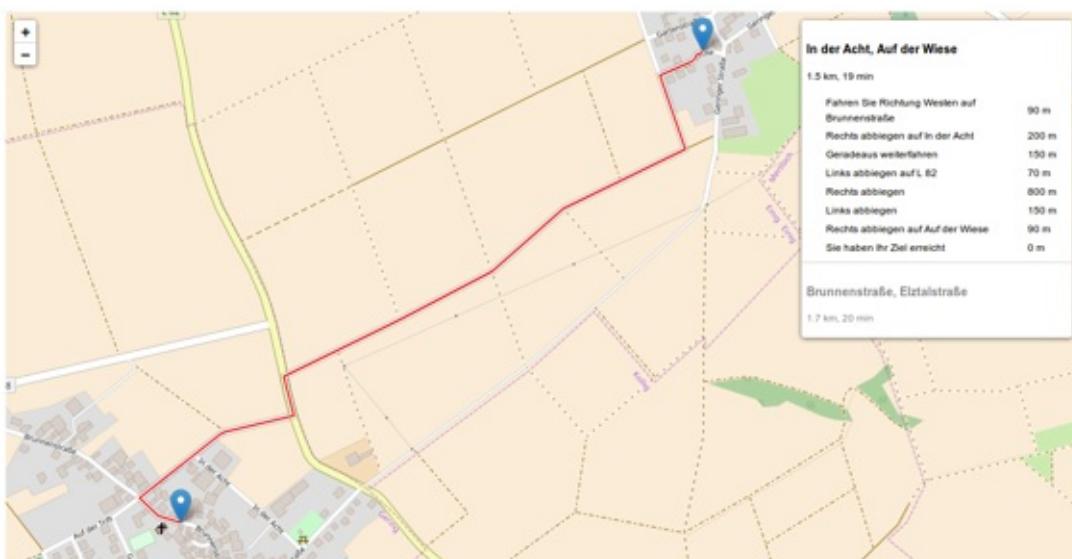
var control = L.Routing.control({
waypoints: [ L.latLng(50.273543, 7.262993),
L.latLng(50.281168, 7.276211) ],
router: L.Routing.mapbox('pk.ihrkey'),
{ profile: 'mapbox/walking', language: 'de' }), }
).addTo(mymap);

</script>
</body> </html>
<!--index_931.html-->
```

Der Mapbox Router bietet Ihnen unter anderem die Optionen `profil` und `language`. Die Namen sagen es schon aus. Als Profil können Sie einstellen, ob Sie wandern oder lieber mit dem Auto fahren möchten. Die Option `language` bietet Ihnen eine Menge unterschiedlicher Sprachen. Möchten Sie noch etwas anderes ändern? Die Dokumentation zum Router von Mapbox finden Sie unter der Adresse <https://www.mapbox.com/api-documentation/#introduction>.

Wenn Sie lieber eine andere Software zur Berechnung der Route nutzen möchten, können Sie dies tun. Die Leaflet Routing Machine bietet Ihnen viele Möglichkeiten. Weitere Informationen, Demos und Tutorials zur Leaflet Routing Machine finden Sie auf der Website: <http://www.liedman.net/leaflet-routing-machine/>.

Im nächsten Bild sehen Sie, dass das Routing nun eher zu einem Wanderer passt und die Texte sind auch in deutscher Sprache. Das ist schon einmal gut. Aber es geht ja meist noch besser. Wahrscheinlich möchten Sie, dass die Adressen, zwischen denen geroutet wird, variabel auf der Karte verändert werden können. Genau das wollte ich als Nächstes und deshalb habe ich dies im nächsten Beispiel umgesetzt.



## Geocoding und Routing

Bisher wurde die Route ausschließlich anhand von Koordinaten berechnet. Menschen nutzen aber lieber Texte in Form von Adressen. Das Geocoding Adressen in Koordinaten verwandelt – beziehungsweise Koordinaten in Adressen verwandelt – haben Sie im Kapitel *Geocoding* im Kapitel *ESRI* schon lesen können. Hier im Beispiel kombinieren wir nun Routing und Geocoding. Sehen Sie selbst:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OpenStreetMap Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-routing-machine.js"></script>

<script src="Control.Geocoder.js"></script>
<link rel="stylesheet" href="Control.Geocoder.css" />

<link rel="stylesheet" href="leaflet-routing-machine.css" />
</head>
<body>
<div style="height: 700px;" id="mapid">
</div>
<script>
```

```

var mymap = L.map('mapid')
.setView([50.27264, 7.26469], 13);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png')
.addTo(mymap);

var control = L.Routing.control({
waypoints: [ L.latLng(50.273543, 7.262993),
L.latLng(50.281168, 7.276211) ],
router: L.Routing.mapbox('pk.ihrkey'),
{ language: 'de' }),
geocoder: L.Control.Geocoder.nominatim({})
}).addTo(mymap);

</script>
</body>
</html>
<!--index_930.html-->

```

In diesem Beispiel haben wir das vorhergehende Beispiel mit dem Plugin Geocoder Control erweitert. Konkret haben wir mithilfe der Zeilen

```

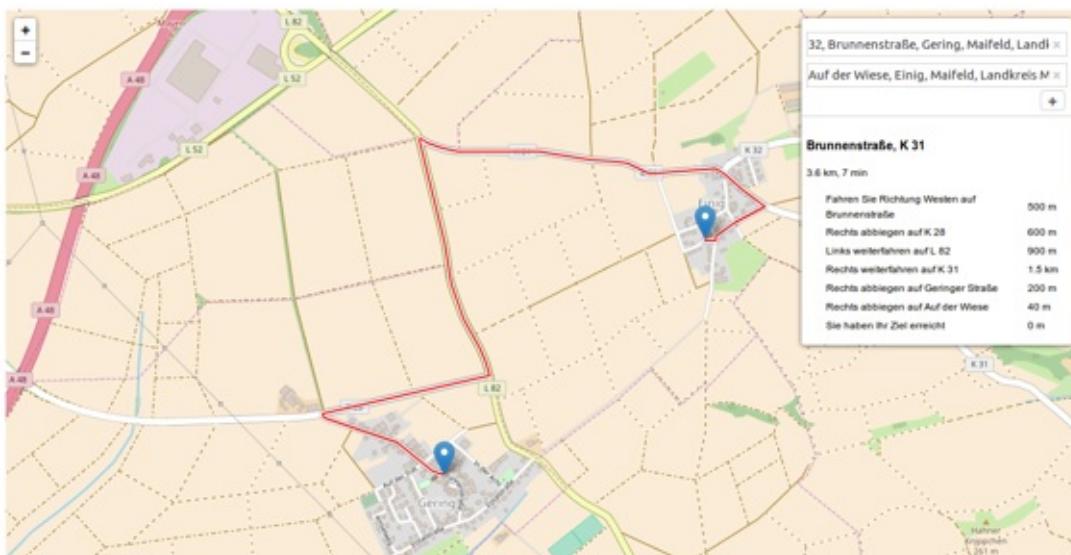
<script src="Control.Geocoder.js"></script>

<link rel="stylesheet" href="Control.Geocoder.css"/>

```

die Skript-Datei und die CSS-Datei des Geocoder Control von Per Liedmann eingebunden. Die aktuellste Version dieses Plugins können Sie unter der Adresse <https://github.com/perliedman/leaflet-control-geocoder/releases> herunter laden. Danach haben wir die Option geocoder des Plugins Leaflet Routing Machine mit dem Wert L.Control.Geocoder.nominatim({}) belegt. Und das war es schon!

Als Ergebnis sehen Sie nun zwei Textfelder im oberen rechten Bereich auf Ihrer Karte. In diese können Sie Adressen eingeben und so Ihre Route benutzerfreundlich abändern.



## In diesem Kapitel haben wir ...

Gegenstand dieses Kapitels war Routing. Wir haben uns kurz angesehen, was Kartendaten bieten müssen, um die Berechnung einer Route zu ermöglichen. Dass die Kartendaten aber nicht alles sind und eine Route etwas sehr Individuelles sein kann, wurde dabei auch klar. Danach haben wir das Plugin Leaflet Routing Machine in unsere Leaflet Karte integriert und mithilfe von Optionen und Geocoding flexibler und benutzerfreundlicher gemacht.

## Auf Wiedersehen

Ich hoffe, dass Ihnen der Rundgang durch LeafletJs gefallen hat und Sie jede Menge für sich mitnehmen konnten.

## Impressum

© 2017 Astrid Günther

Alle Rechte vorbehalten.

ISBN: 9783746015972

Herstellung und Verlag: BoD - Books on Demand, Norderstedt

