

Todo man kann alles in Doku lesen aber manchmal ist es spannender etwas an Beispielen gezeigt zu bekommen.

Willkommen

Das Arbeiten mit Geodaten und Landkarten hat durch das globale Positionsbestimmungssystem GPS (englisch: Global Positioning System) immer mehr an Relevanz gewonnen. 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.

Viele Anwendungen bieten geografische Informationen in Echtzeit an.

[Leaflet](#) ist eine freie JavaScript-Bibliothek, mit der WebGIS-Anwendungen erstellt werden können. Die Bibliothek verwendet HTML5, CSS3 und unterstützt somit die meisten Desktop- und Mobil-Browser. Neben OpenLayers und Google Maps API ist Leaflet eine der verbreitetsten Bibliotheken und wird von Webseiten wie GitHub,[2] FourSquare,[3] Pinterest und Flickr[4] eingesetzt. (<https://de.wikipedia.org/wiki/Leaflet>) todo

ESRI Inc. (Environmental Systems Research Institute) ist ein US-amerikanischer Softwarehersteller von Geoinformationssystemen (GIS). Das in Redlands in Kalifornien ansässige und vollständig im Privatbesitz befindliche Unternehmen wurde 1969 von Jack Dangermond gegründet.
(<https://de.wikipedia.org/wiki/ESRI>) Todo

Geoinformationssysteme, Geographische Informationssysteme (GIS) oder Räumliche Informationssysteme (RIS) sind Informationssysteme zur Erfassung, Bearbeitung, Organisation, Analyse und Präsentation räumlicher Daten. Geoinformationssysteme umfassen die dazu benötigte Hardware, Software, Daten und Anwendungen.
(<https://de.wikipedia.org/wiki/Geoinformationssystem>) Todo

Wichtiges zum Buch

Worum geht es in diesem Buch

Wer sollte dieses Buch lesen

Ich gehe davon aus, dass Sie über grundlegende HTML und CSS (Cascading Style Sheets) Kenntnisse verfügen. Sie sollten auf alle Fälle wissen, wie Sie Cascading Style Sheets und JavaScript Skripte 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

Fragen per E-Mail an die E-Mail-Adresse info@astrid-guenther.de

Eine Karte mit Leaflet erstellen

Todo was behandelt dieses Kapitel?

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 <div>-Element in dem die Karte angezeigt werden soll.
3. Erstellen Sie das Karten-Objekt.
4. Fügen Sie einen Karten-Layer zur Karte hinzu.

Zunächst erstellen wir aber eine HTML-Datei, die Grundlage für die Beispiele sein wird.

Todo Beispielcode?

```
<!DOCTYPE html>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
```

```
</head>
<body>
</body>
</html>todo 999
```

Ein Aufruf dieser Datei öffnet ein leeres Browser-Fenster. Nur der Titel der Seite ist abzulesen.

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

Sie haben zwei Möglichkeiten die notwendigen Dateien zu integrieren.

1. Erstellen Sie eine eigene lokale Kopie der Dateien
2. Laden Sie die Dateien über ein Content Delivery Network (CDN)

Ein Content Delivery Network (CDN) oder auch Content Distribution Network genannt, ist ein Netz regional verteilter und über das Internet verbundener Server, mit dem Inhalte – insbesondere große Mediendateien – ausgeliefert werden. Ein CDN stellt skalierende Speicher- und Auslieferungskapazitäten zur Verfügung und gewährleistet auch bei großen Lastspitzen einen optimalen Datendurchsatz. Vorteil und Nachteile (Todod)

Eine Kopie im Internet

Xx

```
<!DOCTYPE html>
<html>
<head>
<title>Eine OSM 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>
</body>
</html> todo 998
```

Todo Beispielcode? Und wie binde ich skripte am besten ein defer?? und
https://stackoverflow.com/questions/45281262/where-should-i-put-leaflet-script-tags-in-html-markup-and-can-i-use-async-or-d/45282437?noredirect=1#comment77528842_45282437 -
<http://wiki.selfhtml.org/wiki/JavaScript/DOM/Event/load> und mymap99_test.

Todo https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

Todo <https://dev.w3.org/html5/html-author/#void-elements-0>
<https://dev.w3.org/html5/html-author/#start-tag>

xx

Auch jetzt sieht man noch nichts, wenn man die HTML-Datei im Browser aufruft.

Todo Link <http://leafletjs.com/download.html>

Eine lokale Kopie

Xx

Todo Download <http://leafletjs.com/download.html>

Erstellen Sie ein <div>-Element in dem die Karte angezeigt werden soll

Xx

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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> todo 997
```

Todo Tipp css in html inline damit man unterschiedliche karten in unterschiedlichen größen anzeigen kann.

Erstellen Sie das Karten-Objekt

Nun wird es spannend.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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> 996
```

Todo: Tipp Center und Zoom immer zuweisen.

Fügen Sie eine Schicht mit Kacheln (Tile-Layer) zur Karte hinzu

Der letzte Schritt beim Erstellen der Karte ist das Hinzufügen der Kacheln. Diese Schicht oder dieser Layer, kann als eine Art Basis angesehen werden. Es

handelt sich um die grafische Darstellung auf der die Objekte die wir hier im Buch erarbeiten, dargestellt werden.

Kacheln in einem Layer werden als Service von unterschiedlichen Servern angeboten. Im nächsten Kapitel werde ich Ihnen genauer erläutert, dass diese Kacheln normalerweise als 256 x 256 Pixel Images angeboten werden und warum URL zum Aufruf der Kacheln /z/x/y.png lautet.

Ich verwende hier OpenstreetMap zur Darstellung der Karte. 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 OSM 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> todo 995
```

Exkurs: Längengrad und Breitengrad: Geographische Koordinaten

Todo: <https://www.timeanddate.de/geographie/laengengrad-breitengrad>

todo 964.png

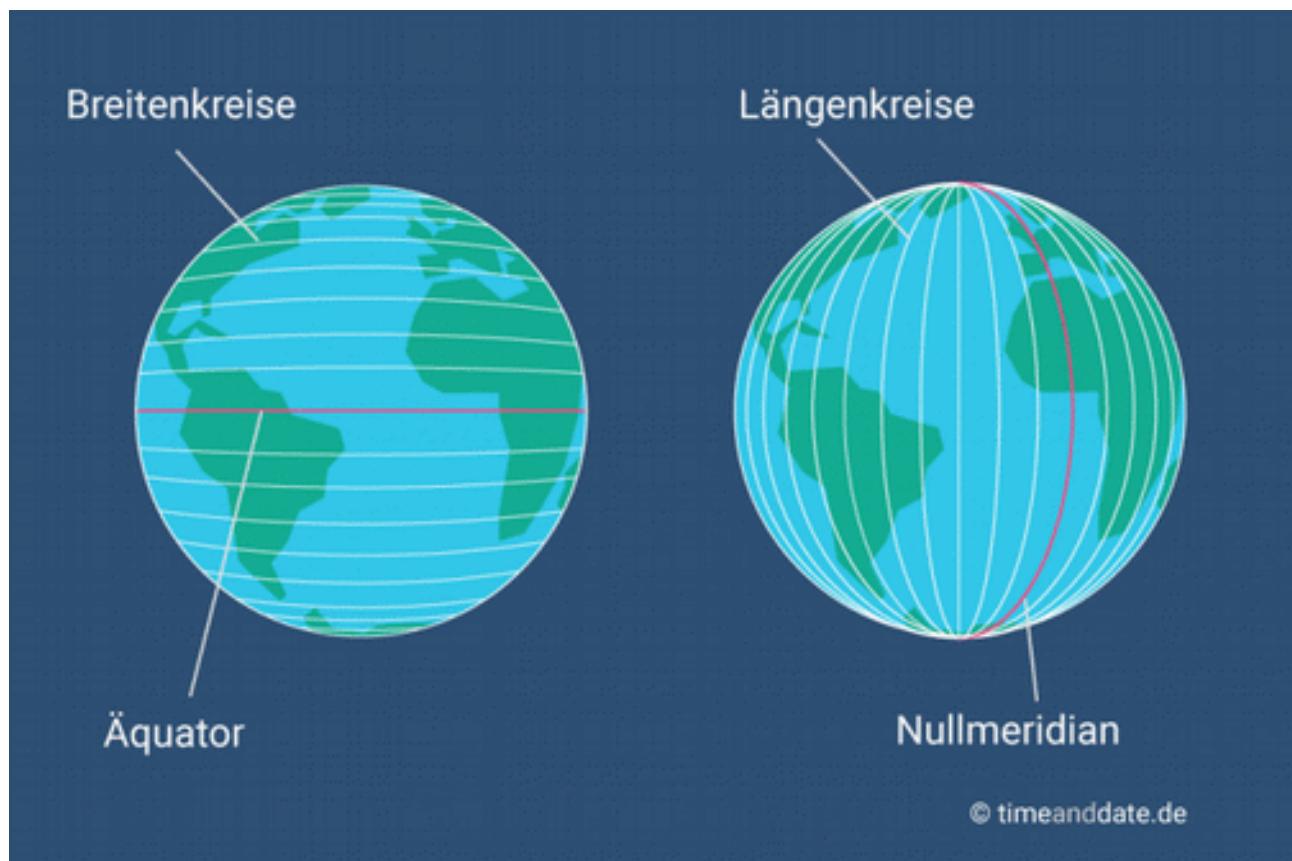
todo 964a.png, und

todo 964b.png noch ändern.

Todo Texte in diesem Kapitel ändern

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

964.png



Das Koordinatensystem

Zum Aufbau des Koordinatensystems wird unser Erdball zunächst in 180 Breitengrade und 360 Längengrade eingeteilt.

- Breitengrade verlaufen parallel zum Äquator.
- Auf einer eingenordeten Karte sind die Breitengrade etwa waagerecht.

- Längengrade verbinden Nord- und Südpol.
- Auf einer [eingenordeten](#) Karte sind die Längengrade etwa senkrecht.

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.

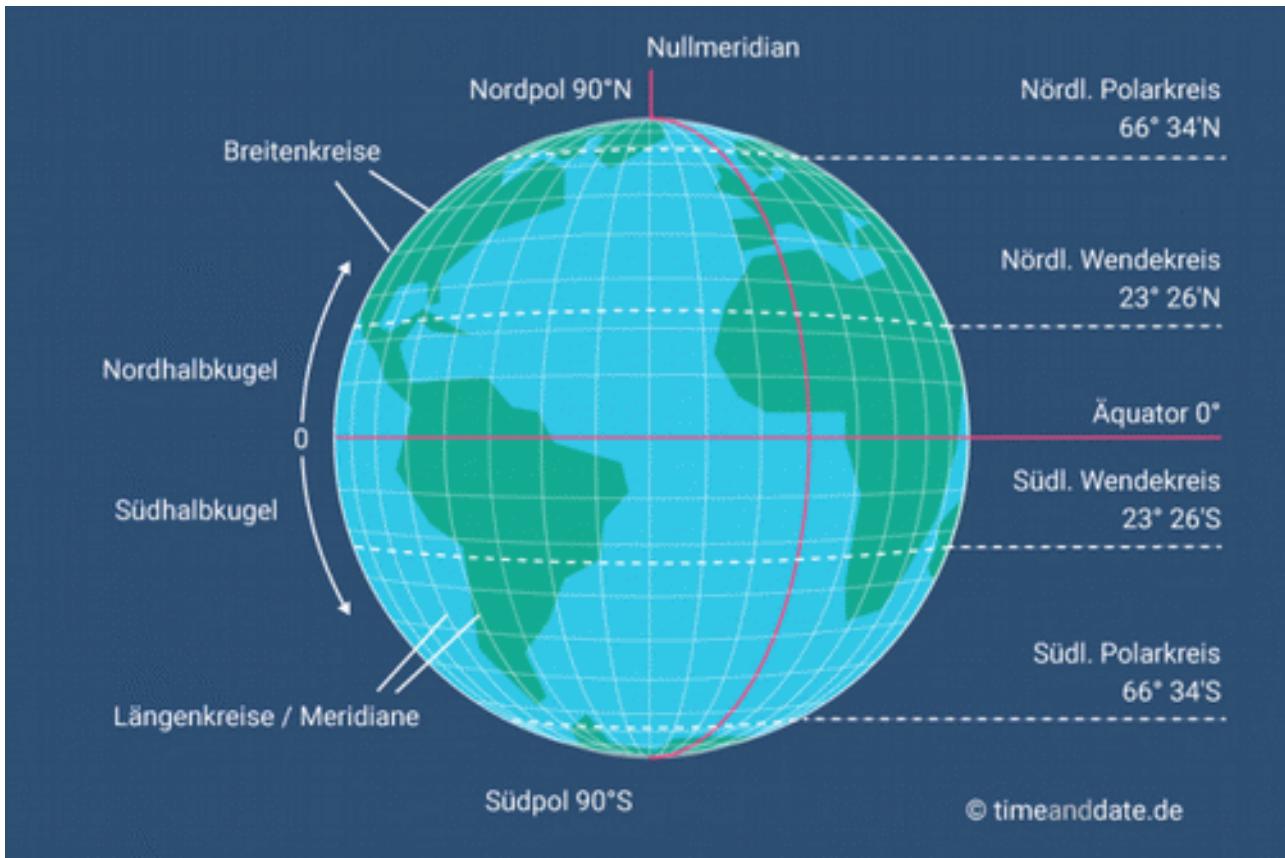
Breitengrade

Die Breitenkreise verlaufen auf einer Ost-West-Achse und bilden so die Nord-Süd-Dimension eines Ortes auf der Erdoberfläche ab. Sie wissen sicher noch aus dem Geografieunterricht in der Schule, dass der Äquator im rechten Winkel zur Erdachse verläuft. Er liegt ungefähr in der Mitte zwischen Nord- und Südpol. Mit einem Winkel von 0° gilt er im geografischen Koordinatensystem als Ausgangspunkt für die Berechnung der Breitenkreise.

Die geographische Breite wird angegeben als Winkel zwischen der Linie Erdmittelpunkt-Äquator und der Linie Erdmittelpunkt-Ort. Nord- und Südpol haben einen Breitenwinkel von 90° . Um Orte auf der Nordhalbkugel von jenen auf der Südhalbkugel zu unterscheiden, wird die Breite in der traditionellen Schreibweise zusätzlich mit einem N für Nord oder einem S für Süd versehen.

Beispiel: Der Alexanderplatz in Berlin liegt auf $52^\circ 31' 18''\text{N}$ (nördlicher Breite). Zöge man einen geraden Strich von dort zum Erdmittelpunkt, hätte er genau diesen Winkel zur Äquatorebene bzw. zu einem Strich auf demselben Längengrad vom Äquator zum Erdmittelpunkt.

Die wichtigsten Breitenkreise



964a.png

Polarkreis

Die Polarkreise liegen bei jeweils $66^\circ 34'$ nördlicher und südlicher Breite. Diese Breitenkreise begrenzen die Regionen um die Pole, in denen die Sonne zur Sommersonnenwende nicht untergeht, wo also dann Mitternachtssonne herrscht. Zur Wintersonnenwende geht die Sonne hier nicht auf, es herrscht Polarnacht.

Wendekreise

Die Wendekreise liegen bei jeweils $23^\circ 26'$ nördlicher und südlicher Breite. Diese Breitenkreise markieren den Sonnenzenit zu den Sommersonnenwenden der Nord- und Südhalbkugel.

Äquator

Der Äquator liegt bei 0° . Die Äquatorebene durchläuft in etwa den Mittelpunkt der Erdachse zwischen Nord- und Südpol.

Längengrade

Die Längenkreise verlaufen auf einer Nord-Süd-Achse und beschreiben so die Ost-West-Dimension eines Ortes auf der Erdoberfläche.

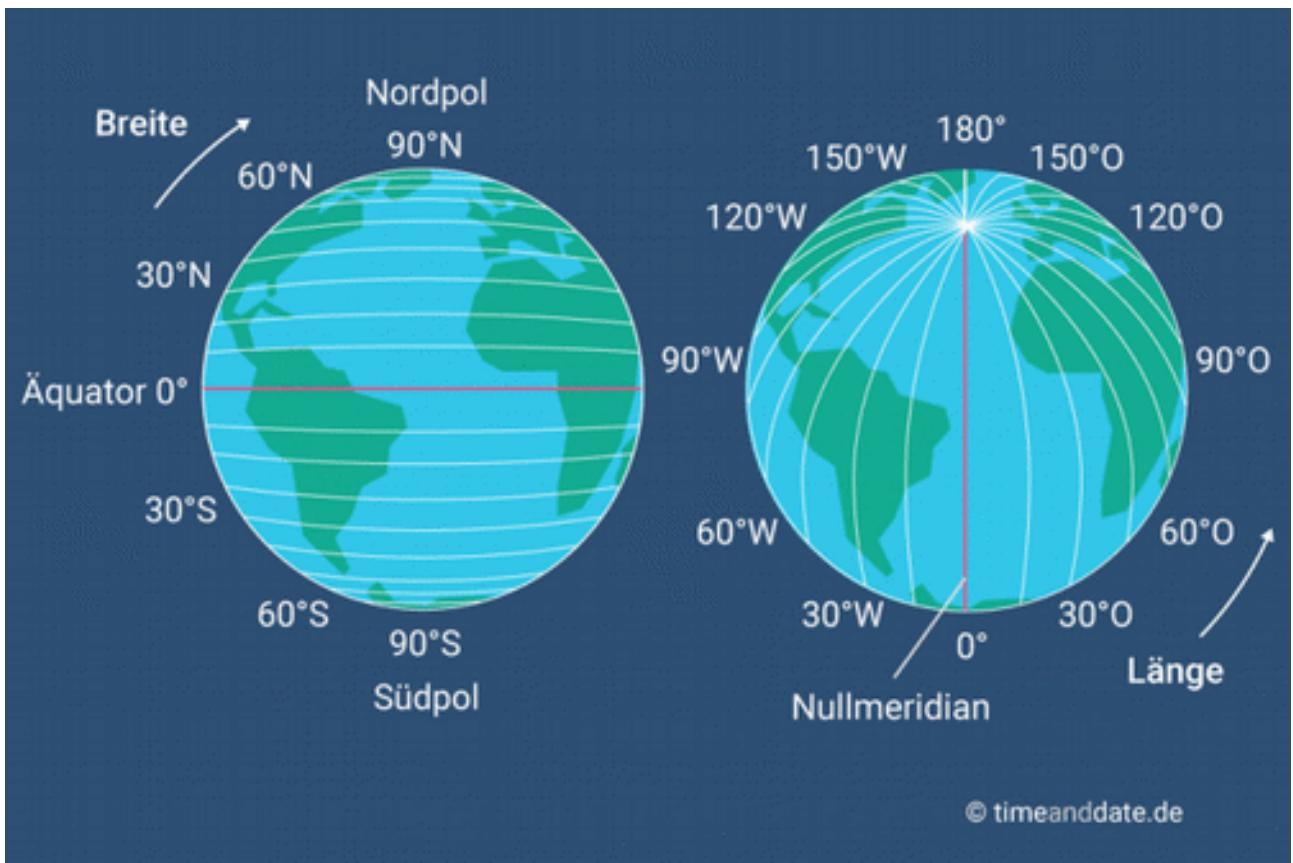
Als erdumspannende Kreise durchlaufen sie den Nord- und Südpol und kreuzen den Äquator. Eine die Pole verbindende Längenkreishälfte wird als Meridian bezeichnet.

Anders als die Breitengrade, deren Winkel sich vom Äquator ableiten, haben die Längengrade keinen natürlichen Nullpunkt. Deswegen wurden bis ins frühe 20. Jahrhundert je nach Land verschiedene Ausgangspunkte verwendet. Heute gilt der Meridian, der den Londoner Stadtteil Greenwich durchläuft, als Nullmeridian und somit als Ausgangspunkt für die Berechnung der Längengrade.

Die geographische Länge wird angegeben als Winkel zwischen der Linie Erdmittelpunkt-Nullmeridian und der Linie Erdmittelpunkt-Ort. Der Greenwich-Meridian hat einen Winkel von 0° , während der gegenüberliegende Längengrad, entlang dessen die Datumsgrenze verläuft, einen Winkel von 180° aufweist.

Der Nullmeridian unterteilt die Erdoberfläche in eine westliche und eine östliche Halbkugel. Um Orte auf den beiden Hemisphären voneinander zu unterscheiden, wird in der traditionellen Schreibweise die Länge zusätzlich mit einem W für West oder einem O für Ost versehen.

Beispiel: Das Openhaus von Sydney liegt auf $151^\circ 12' 50.5'' \text{O}$ (östlicher Länge). Zöge man einen geraden Strich von dort zum Erdmittelpunkt, hätte er genau diesen Winkel zu einem Strich auf demselben Breitengrad vom Nullmeridian zum Erdmittelpunkt.



964b.png

Schreibweisen

Bei der Angabe von geographischen Koordinaten wird heute normalerweise eine von zwei Schreibweisen verwendet.

Todo umrechner: <https://www.deineberge.de/Rechner/Koordinaten/Dezimal/51,10>

Das Sexagesimalsystem

Dies ist die traditionelle Schreibweise. Jeder Breiten- und Längengrad ($^{\circ}$) wird in 60 Minuten ('') mit je 60 Sekunden ("") unterteilt. Das Sexagesimalsystem ist ein Zahlensystem, das auf der Zahl 60 basiert. Hier bestehen die Koordinaten aus drei Komponenten:

- Längen- und Breitengrade werden als Winkel ($^{\circ}$) angegeben.
- Jeder Grad hat 60 Minuten. Diese werden durch eine Prime ('') gekennzeichnet.
- Jede Minute hat 60 Sekunden, die anhand einer Doppelprime ("") erkennbar sind.

So hat die Zugspitze die Koordinaten 47°25'16"N, 10°59'7"O.

Die Dezimalschreibweise

Parallel zum traditionell gebräuchlichen Sexagesimalsystem hat sich zusätzlich die Angabe der Koordinaten im Dezimalsystem etabliert. Das Dezimalsystem basiert auf der Zahl 10. Hier wird der Bruchteil des jeweiligen Längen- und Breitengrades durch beliebig viele Nachkommastellen angegeben. Orten auf der West- und Südhalbkugel wird ein Minus (-) vorangestellt.

Im Dezimalsystem werden die Koordinaten der Zugspitze so angegeben:
47.4211, 10.9852.

Machu Picchu, auf dem westlichen Teil der Südhalbkugel gelegen, hat die Koordinaten

-13.163333, -72.545556.

Exkurs: Wie werden Landkarten auf einer Webseite angezeigt?

Vektoren und Bitmaps

Dies ist ein ziemlich großes Thema und wie Sie wahrscheinlich wissen, gibt es viele Alternativen für die Anzeige von Karten im Internet. Aber letztlich müssen sie alle ein Bild auf einen Bildschirm stellen. Es gibt mehr als einen Weg, dies zu tun, mit dem größten Unterschied ist die Präsentation des Bildes auf dem Bildschirm entweder ein Vektor-Bild oder eine Bitmap.

Vektoren

Todo bilder und <https://medium.com/@mbostock/command-line-cartography-part-1-897aa8f8ca2c>

<http://t3n.de/magazin/javascript-bibliothek-d3-237224/>

Bitmaps

Todo bilder

Vektoren und Bitmaps für Karten: Aufbau einer digitalen Vektorkarte

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. Aus diesem Grund werden die Karten in der Regel in kleinen

Bereichen im Vorfeld berechnet und in einem Grafikformat gespeichert. Bei der Anzeige werden dann immer nur die Kartenbereiche neu berechnet, bei denen sich etwas geändert hat.

OSM teilt die Welt in Kacheln

Die Karte von OpenStreetMap zerlegt die Welt in Quadrate und nennt diese Quadrate Tiles. Jedes Quadrat ist 256 x 256 Pixel groß.

Die Google Maps API unterteilt ihr Kartenbilder auch in quadratische Kartenkacheln, die in einem Raster angeordnet sind. Wenn Sie die Website google.de/maps aufrufen und eine neue Vergrößerungsstufe wählen wird ermittelt, welche Daten erforderlich sind. Diese Daten werden dann in einen Kachelsatz übersetzt und angezeigt.

Die Zoomstufe 0 bildet die ganze Welt auf ein Quadrat ab. Teilt man den Erdumfang von 40.038 Kilometern durch die 256 Pixel sieht man im Ergebnis, dass ein Pixel 156,4 Kilometer darstellt. Das ist noch nicht sehr detailliert. Bis Zoomstufe 19 ändert sich aber eine ganze Menge. Auf der nachfolgenden Tabelle sehen Sie, dass bei Zoomstufe 19 ein Pixel einem Bereich von 0,3 Metern auf der Erde entspricht.

Zoomstufe	Kachelanzahl	Kachelbreite entspricht	Pixel entspricht
0	1	40.038 km	156 km
1	4	20.019 km	78 km
..
18	69 Mrd.	153	0,6 m
19	275 Mrd.	76 m	0,3 m

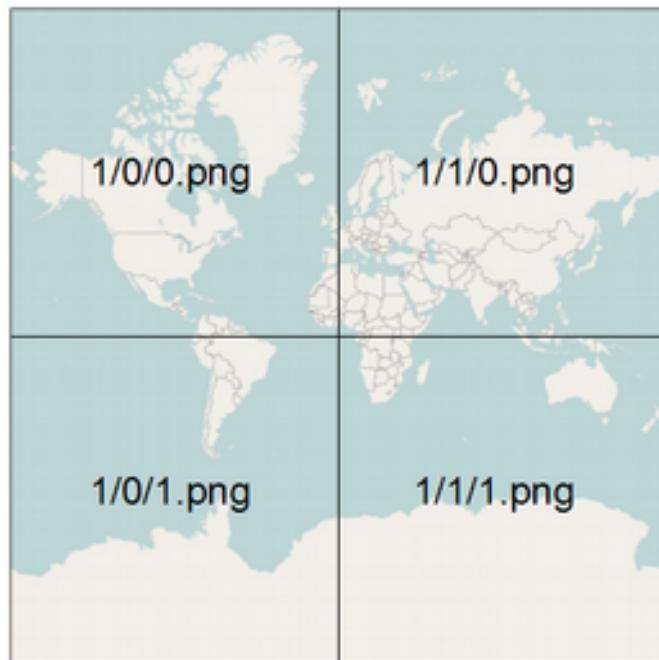
Die Tabelle können Sie unter der Adresse wiki.openstreetmap.org/wiki/Zoom_levels vollständig und mit weiteren Angaben im Internet abrufen.

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

Wie weiß Leaflet welche Kacheln angezeigt werden sollen?



998.png



998a.png

todo exkurs tiles, Seite 96

todo subdomains: [http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames
{s}](http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames_{s})

Schöne Kartenlayer

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

Ich zeige Ihnen hier zwei weitere Provider, nämlich Thunderforest (<https://www.thunderforest.com/>) und Stamen (<https://stamen.com/>).

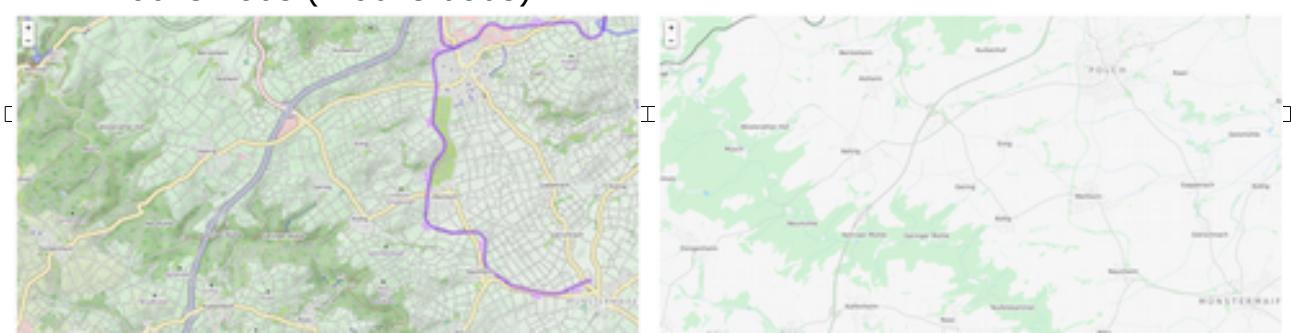
Thunderforest erweitert von Openstreetmap angebotenen Kacheln und Stamen bietet etwas völlig anderes. Die Kacheln von Stamen sind eher künstlerischer Natur. Beide Anbieter bieten Ihnen die Möglichkeit einen Alternativen Stile zu Ihrer Leaflet-Karte hinzuzufügen.

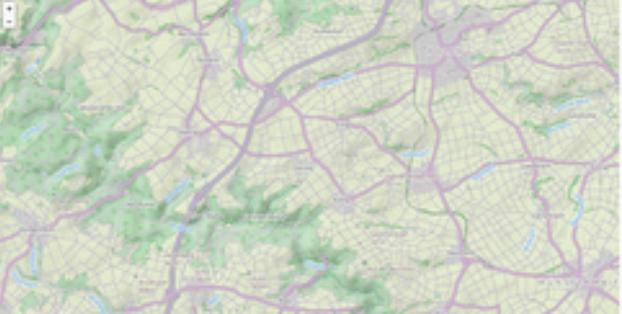
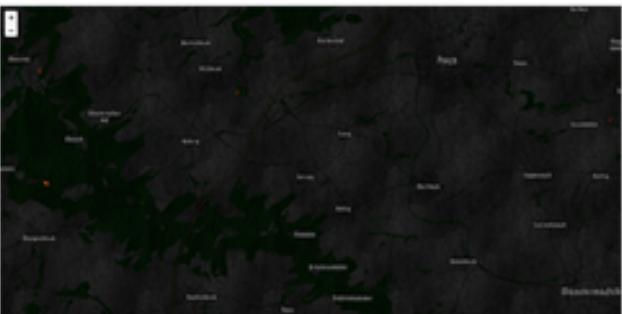
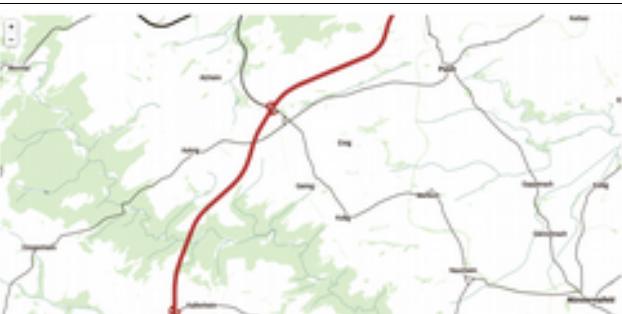
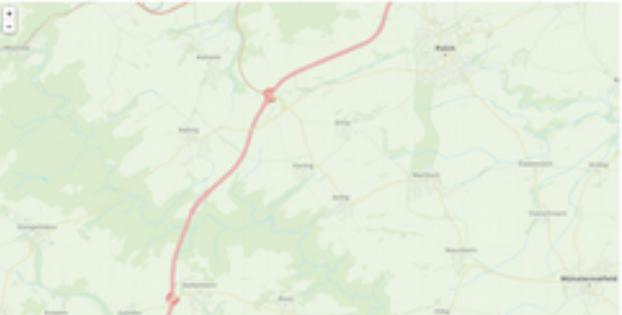
Thunderforest

Todod bei Anmeldung apikey

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

- OpenCyclemap (cycle):
997.png todo
- Transport (transport)
- Landscape (landscape):
Diese Darstellung konzentrierte sich auf Informationen in der Natur – ideal für den ländlichen Kontext.
- Outdoors (outdoors)
- Transport Dark (transport-dark)
- Spinal Map (spinal-map)
- Pioneer (pioneer)
- Mobile Atlas (mobile-atlas)



Cycle 997.png	transport
	
landscapre	outdoor
	
Transport dark	Spinal
	
pionier	Mobileatlas
	
neigborhood	

Wenn Sie Thunderforest verwenden möchten, müssen Sie als Tile Layer nur die im nachfolgenden Beispiel zu sehende URL angeben. Der nachfolgende Beispiel-Programmcode zeigt Ihnen die genaue Syntax.

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

```

<title>Eine OSM 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('https://tile.thunderforest.com/cycle/{z}/{x}/
{y}.png?apikey=YourApiKey').addTo(mymap);
</script>
</body>
</html> todo 994

```

Stamen

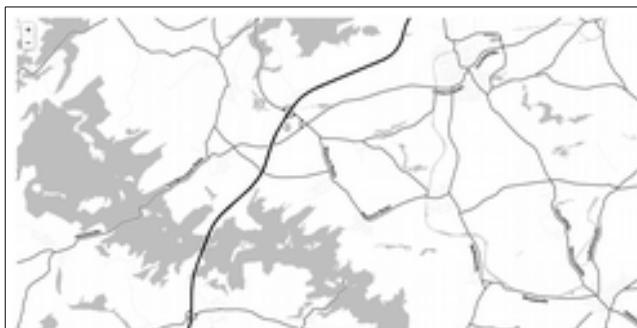
Todo

Man muss Skript einbinden und speziellen Layer erstellen.

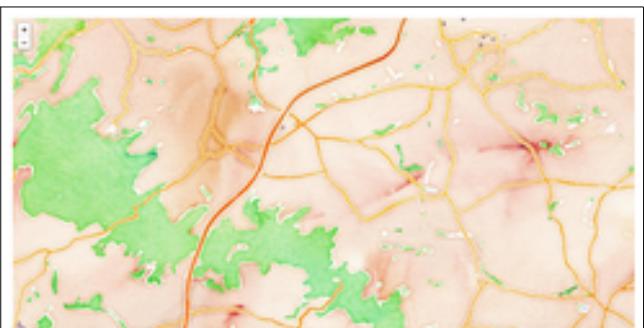
```

"toner":      MAKE_PROVIDER("toner", "png", 0, 20),
"terrain":    MAKE_PROVIDER("terrain", "png", 0, 18),
"terrain-classic": MAKE_PROVIDER("terrain-classic", "png", 0, 18),
"watercolor": MAKE_PROVIDER("watercolor", "jpg", 1, 18),

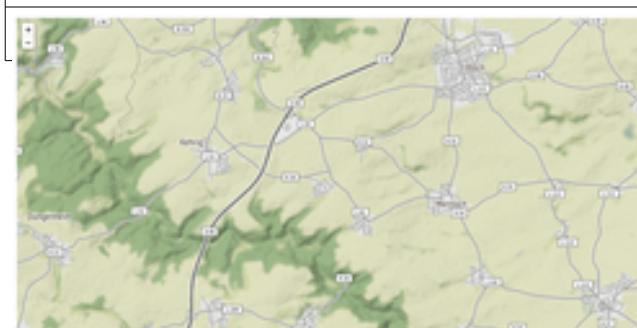
```



Toner 996.png



Watercolor 996.png



Terraint 996.png	
------------------	--

Bild 996.png

Todo Achtung zoom 19 funktioniert nicht.

(<https://stackoverflow.com/questions/45299088/exclude-zoom-level-in-leaflet-stamen-map/45307170#45307170>)

Weiter Anbieter von Kachel finden Sie unter der Adresse:

<http://wiki.openstreetmap.org/wiki/Tiles>

Tipp <http://mapstack.stamen.com/> Image machen

Todo: Wer google nutzen will:

<https://gitlab.com/IvanSanchez/Leaflet.GridLayer.GoogleMutant>

Todo <https://gis.stackexchange.com/questions/53011/difference-between-a-wmts-and-a-wms>

https://de.wikipedia.org/wiki/Web_Map_Service

https://de.wikipedia.org/wiki/Web_Map_Tile_Service

Todo Leaflet unterstützt WMS und WMTS – Es gibt noch tms

todo <http://leafletjs.com/examples/wms/wms.html>

Images als Layer – Web-Map-Service

Sie haben eine Satellitenaufnahme und möchten diese als Layer verwenden. Nun wäre ein Umwandeln dieser Satellitenaufnahme in 275 Milliarden Kacheln, wie es der im vorherigen Kapitel beschriebenen wurde, zwar möglich – es gibt aber alternative Techniken.

Eine Alternative zur beschriebenen Kachel-Technik ist der Web-Map-Service. GetMap: Diese Anfrage liefert ein georeferenziertes Rasterbild (Karte) vom WMS zurück. Innerhalb der Anfrage können u.a. Optionen über die gewünschten Kartenlayer, die gewünschte Darstellung der Layer, das zugrundeliegende Koordinatensystem, den Kartenausschnitt, die Größe der Kartenausgabe und das AusgabefORMAT gemacht werden.

Ein [Web-Map-Service](#) (WMS) ist eine Schnittstelle zum Abrufen von Auszügen aus Landkarten über das Internet. Der WMS ist ein Spezialfall eines [Web](#)

Services und ermöglicht die Maschine-zu-Maschine-Kommunikation auf Basis von HTTP oder HTTPS über Rechnernetze wie das World Wide Web.

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

Todo https://de.wikipedia.org/wiki/United_States_Geological_Survey

todo <https://basemap.nationalmap.gov/arcgis/rest/services>

todo deutscher Wetterdienst:

http://www.dwd.de/DE/wetter/warnungen_aktuell/objekt_einbindung/einbindung_karten_geowebservice.pdf?blob=publicationFile&v=2

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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",
{
format: 'image/png',
transparent: true,
layers:'dwd:bluemarble',
attribution: "Deutscher Wetterdienst"
}).addTo(mymap);
</script>
</body>
</html>
```

index_992.html



995.png

todo <http://qgis.org/de/site/>

Open Source Software Geoserver - DWD GeoWebservice: WMS Diensten für die eigene Website

Layer über Layer

Todo transparenz

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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",
{
```

```

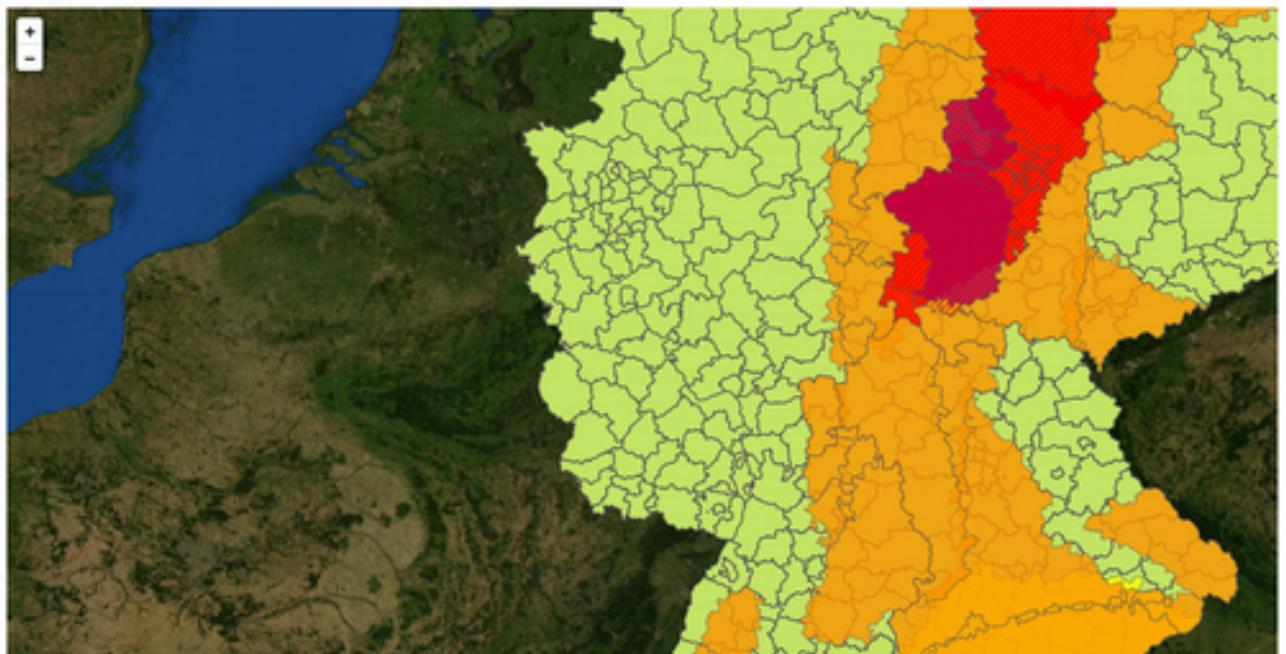
        format: 'image/png',
        transparent: true,
        layers:'dwd:bluemarble',
        attribution: "Deutscher Wetterdienst"
    }).addTo(mymap);

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

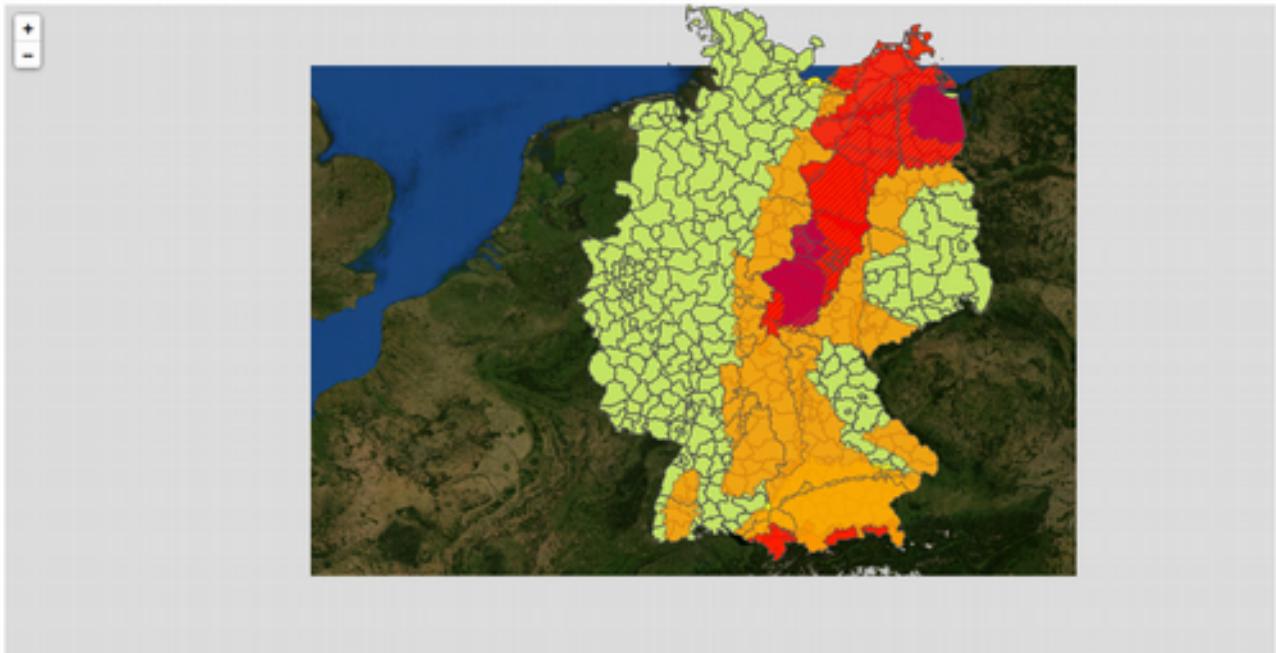
var dwd = L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
    format: 'image/png',
    transparent: true,
    layers:'dwd:Warnungen_Gemeinden_vereinigt',
    attribution: "Deutscher Wetterdienst"
}).addTo(mymap);
</script>
</body>
</html>

```

index_991.html



994.png



994a.png

todo auch wmts layer kann verwendet werden.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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 dwd = L.tileLayer.wms("https://maps.dwd.de/geoserver/dwd/wms",
{
format: 'image/png',
transparent: true,
```

```

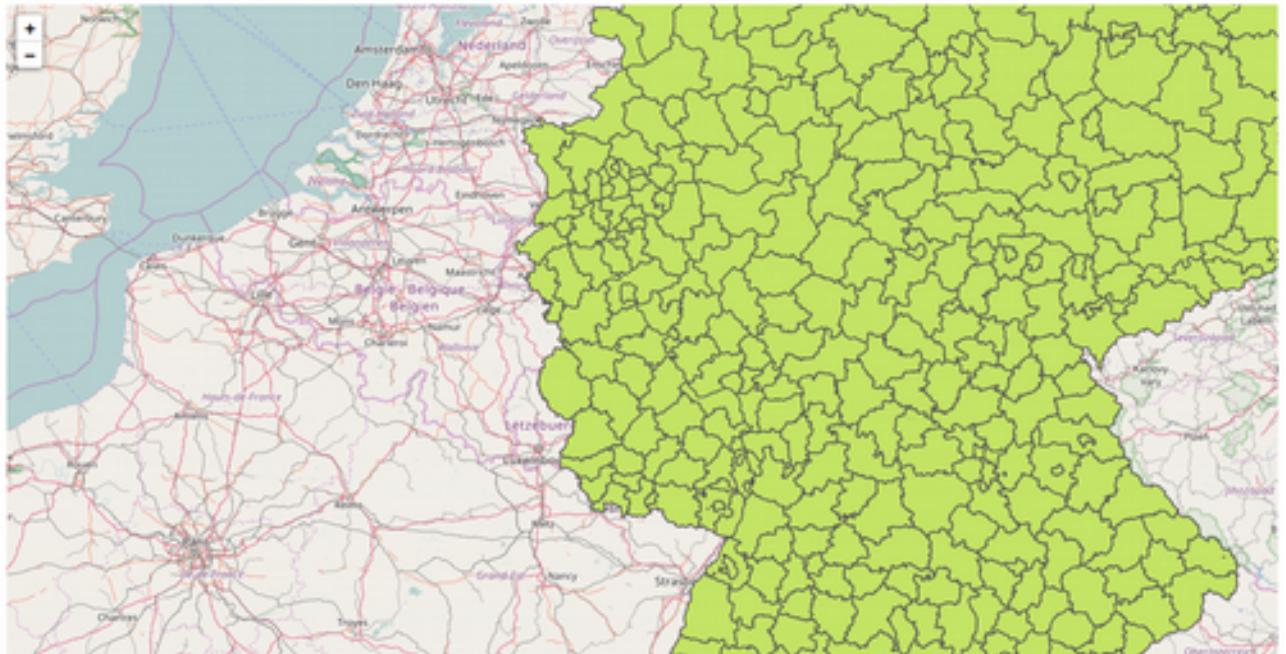
layers:'dwd:Warngebiete_Kreise',
attribution: "Deutscher Wetterdienst"
}).addTo(mymap);

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

</script>
</body>
</html>

```

index_990.html



994b.png

Todo Achtung manchmal wird nur etwas angezeigt, wenn Daten vorhanden sind.

Die Karte mit Daten bestücken

Bisher haben wir ausschließlich vollständige Layer zur Karte hinzugefügt. Wie das Beispiel mit den Warnhinweisen vom Deutschen Wetterdienst 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 geometrischen Objekten, die in einem zweidimensionalen Raum definiert.

- Punkt/Marker:
Das grundlegende Objekt ist der Punkt, den wir als ein Paar ganzer Zahlen – den Koordinaten – ansehen. In Leaflet gibt es Punkte und *komfortablere* Punkte – die Marker.
- Linie:
Eine Linie ist ein Paar von Punkten, von denen wir annehmen, dass sie durch einen geraden Linienabschnitt miteinander verbunden sind.
- Polygon und besondere Polygone:
Ein Polygon ist eine Liste von Punkten. Wir nehmen an, dass benachbarte Punkte durch Linien miteinander verbunden sind und dass der erste Punkt mit dem letzten Punkt verbunden ist, so dass eine geschlossene Figur entsteht.

Sehen wir uns also nun an, wie Sie Punkte, Marker, Linien und Polygone auf die Karte malen können.

Von Punkten, Markern, Linien, Polygonen, Rechtecken und Kreisen

Punkte und Marker

Gehen wir noch einmal von unserer Basiskarte aus.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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>

```

989.html



993.png

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.

Punkte

Die Leaflet [Klasse Point](#) stellt einen Punkt mit X- und Y-Koordinaten in Pixeln dar. Point-Objekte sind in Leaflet nicht zur Anzeige gedacht. Vielmehr wird mit Ihnen gearbeitet. Zum Beispiel gibt es die [Methode panBy\(<Point> offset, <Pan options> options?\)](#) mit der die Karte um eine gegebene Anzahl von Pixeln geschwenkt werden kann (animiert todo link zu events).

Das ist nicht das, was wir möchten, wenn wir einen Punkt auf der Karte anzeigen möchten. Hier möchten wir vielmehr einen Marker mit Informationen hinzufügen. Und hierzu gibt es in Leaflet ein spezielles Objekt – das Marker Objekt.

Todod was ist lat und was ist lon und vielleicht formate

```

<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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);
//mymap.panBy([400, 600]);
//mymap.panBy(L.point(400, 600));
</script>
</body>
</html>

```

index_988.html

Marker

Die Leaflet [Klasse Marker](#) wird verwendet, um anklickbare und/oder verschiebbare Symbole auf einer Karte anzuzeigen. Die Klasse erweitert die Klasse [Layer](#).

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

```

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

```

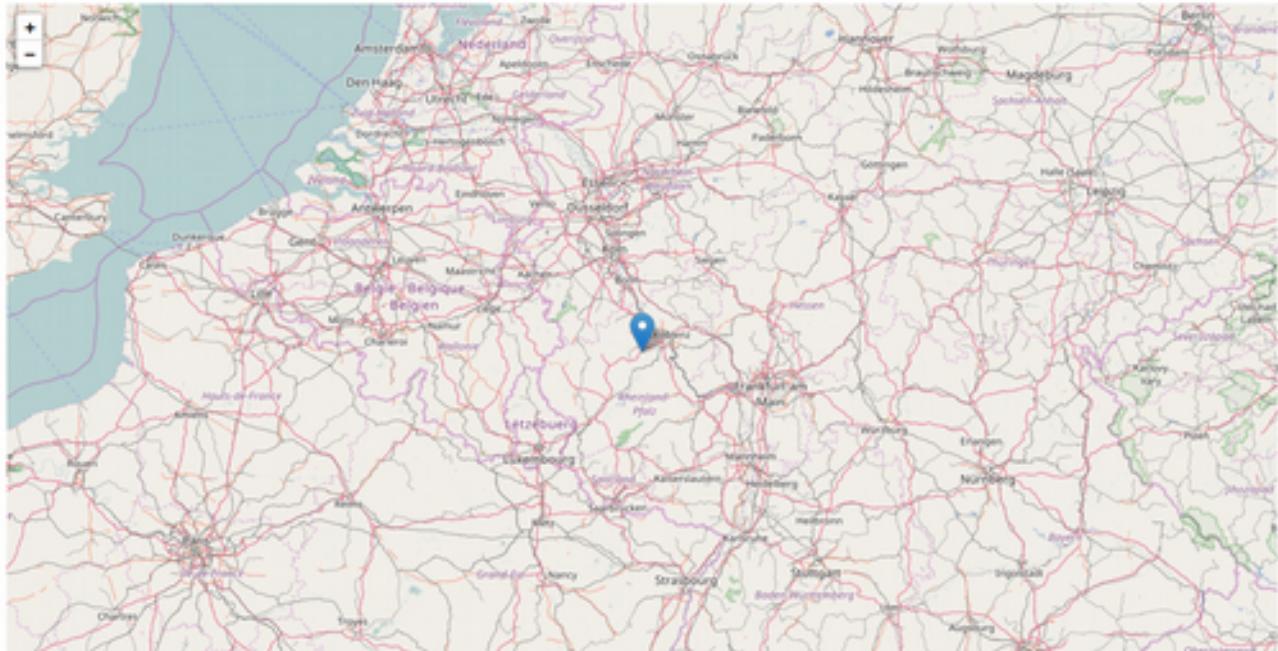
```

<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> 987.html

```

todo was haben wir genau gemacht und was sind Mindestanforderungen



992.png

Todo link zu ausführlicherem Markerbeispiel

Sie müssen für einen Marker keine Variable instanziieren. Sie können den Marker wie folgt 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);
```

Falls aber später den Marker modifizieren möchten, benötigen Sie einen Namen um den Marker zu Identifizieren.

todo alle optionen beim markererstellen <http://leafletjs.com/reference-1.1.0.html#marker>.

- icon (Standardwert: *):

Mit dieser Option können Sie ein benutzerdefiniertes Icon für den Marker verwenden. Weitere Informationen zur Anpassung des Icons finden Sie in der Dokumentation zum [Icons-Objekts](#). Wenn Sie kein eigenes Icon beim Marker angegeben, wird das [Standardsymbol](#) verwendet.

- D

Objekte mit mehr als einem Punkte

Linien

Todo erster vector layer

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 aneinander gereihte Liniensegmente zu erstellen.

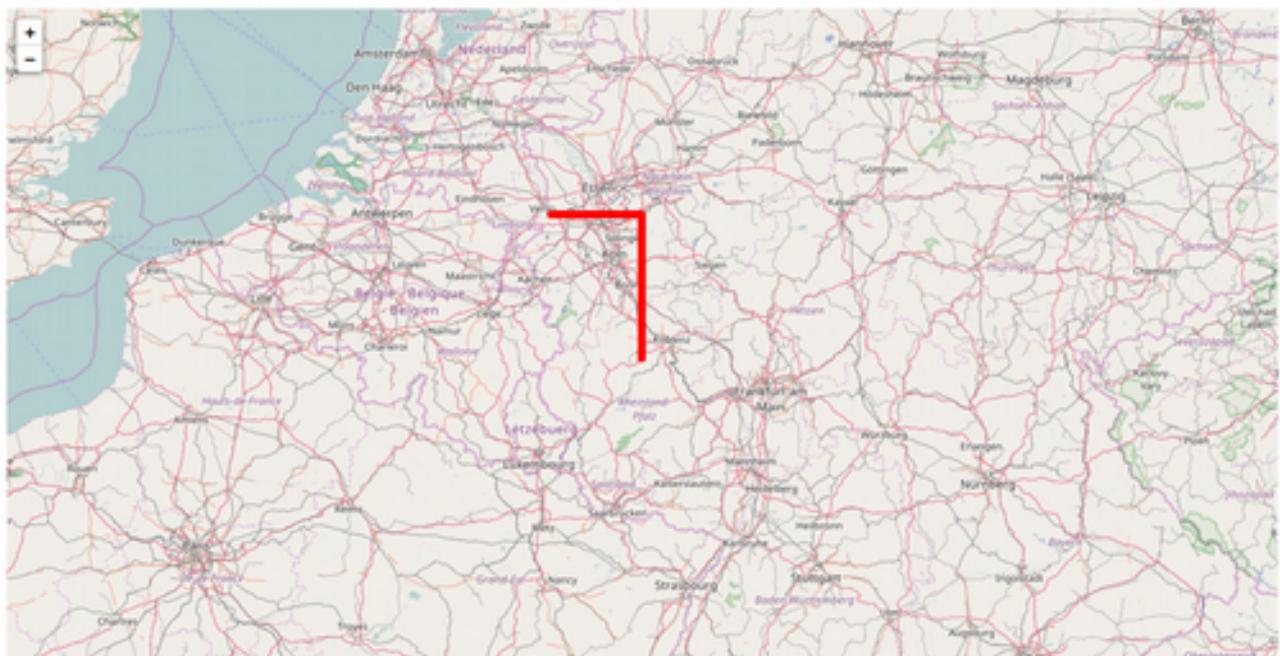
```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
```

```

<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>

```

986.html



991.png

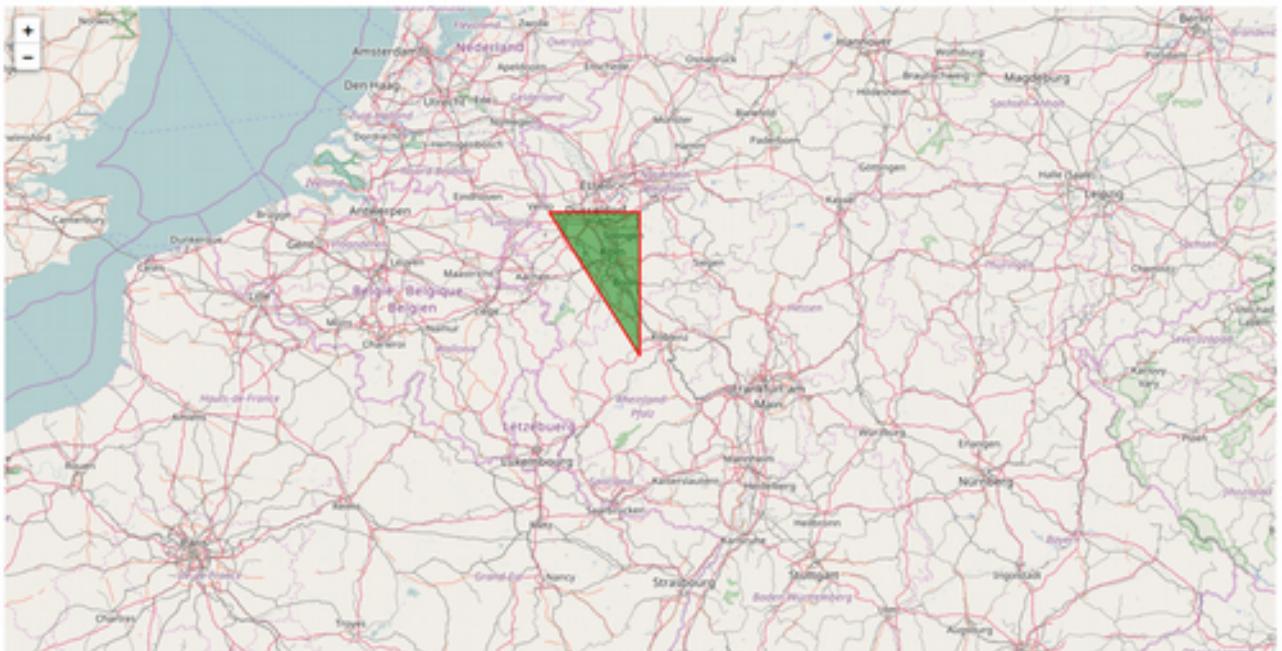
todo: Mindestanforderungen und was haben wir genau gemacht.

Polygone

Ein Polygone ist eine Linie, die geschlossen ist. Mit der Klasse [Polygone](#) können Sie, wie der Name schon sagt, Polygone auf Ihrer Karte zeichnen. Die Klasse erweiterte die Klasse Polylines.

Beachten Sie, dass Punkte, die Sie beim Erstellen eines Polygons übergeben, keinen zusätzlichen letzten Punkt haben, der dem ersten entspricht – es ist besser, solche Punkte herauszufiltern. Leaflet schließt Ihr Polygon automatisch.

Todo Mindesanforderungen und was haben wir getan. Und Optionen → vielleicht klassendiagramm



990.png

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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>
```

985.html

Rechtecke und Kreise

Kreise und Rechtecke könnten man als Polygon darstellen. Dies wäre aber beim Kreis 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 aber in den nächsten beiden Kapiteln ein Beispiel für beide Formen.

Rechtecke

Todo Mindeseingaben, was haben wir gemacht

```
<!DOCTYPE HTML>

<html>
<head>
<title>Eine OSM 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);
```

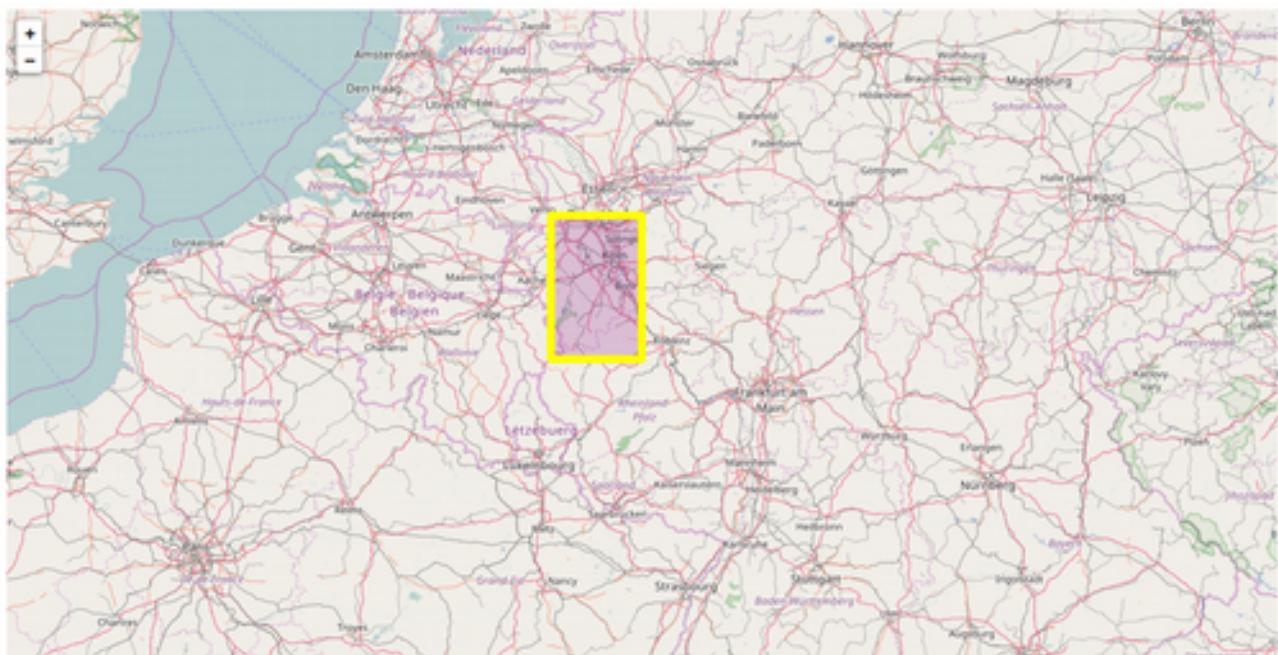
```

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>

```

984.html



989.png

Kreise

Todo Mindesangaben was habe ich gemacht.

```

<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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);

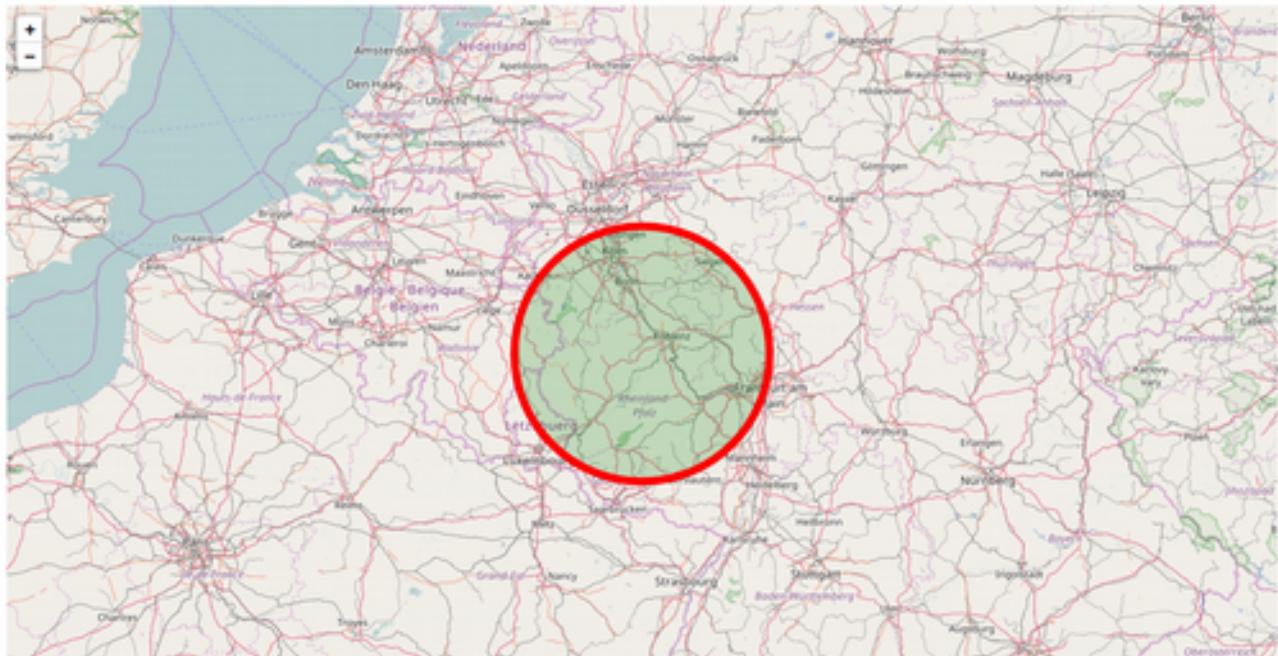
L.circle(
[50.27264, 7.26469],
100000,
{color: "red", weight: 8, fillColor:"green"}).addTo(mymap);

</script>

</body>

</html>
```

983.html



988.png kreis

todo zeigen was beim zoomen passiert.

MulitPolylines und Multipoligones

Im vorhergehenden Beispiel haben wir jedes Element auf einem separaten Layer abgebildet. Spätestens, wenn Sie selbst unterschiedliche Geodaten auf einer Karte darstellen möchten werden Sie sich wünschen, diese auf einem Layer gruppieren zu können. Denn nur so können sie alle Elemente gleichzeitig 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 schon Angebote für eine bestimmte Region heraus suchen. Wenn Sie die Aktivtouren auf Layer gruppiert haben, können Sie es dem Kunden zusätzlich ermöglichen, nur die für Ihn relevanten Touren einzublenden. Todo link wo erklärt ist wie das geht.

Todo <https://de.wikipedia.org/wiki/Layertechnik>

<http://www.duden.de/rechtschreibung/Layer>

Das Setzen von Klammern kann beim Arbeiten mit MultiPolygons und MultiPolylines sehr herausfordernd sein.

- Sie müssen zum einen das MultiPolyelement – MultiPolygon oder Multipolyline – mit einer Klammer versehen.
- Außerdem müssen Sie jedes Polyelement – Polyline oder Polygone – mit einer Klammer umgeben.
- Und zuletzt werden auch die Koordinaten noch eingeklammert. (todo immer richtig geschrieben)

MultiPolylines

Todo Hat sich seit 0.7 geändert:

<https://stackoverflow.com/questions/45371611/creating-multipolyline-with-leaflet-js>

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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>
```

981.html



987.png

todo fitBounds()

todo mindesanforderung und was habe ich gemacht

todo multipolylines hat sich in aktueller version geändert

Multipolygones

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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 multipolygone = 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>
```

982.html



986.png

todo marker (merh als eins öffnen?)

todo mindestanforderung und was habe ich gemacht

Daten mit Layern gruppieren

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

Die Klasse Leaflet Klasse LayerGroup wird verwendet, um mehrere Layer oder Ebenen zu gruppieren und sie als eine zu behandeln. Wenn Sie eine LayerGroup der Karte hinzufügen, werden alle zur Gruppe gehörenden Layer auf der Karte hinzugefügt oder entfernt.

Layergruppen

Todo <http://leafletjs.com/reference-1.1.0.html#layergroup>

todo mindestanforderung, was habe ich gemacht. Ich kann später noch hinzufügen, ich muss nur einmal add to map; remove

Eine Layer ...

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

```

<head>
<title>Eine OSM 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 marker2 = L.marker([51.27264, 6.26469],
{
title:"Marker2",
alt:"Ich bin ein anderer Marker"
}
.bindPopup('Marker2');
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);
myLayerGroup.addLayer(marker2);
//myLayerGroup.removeLayer(polyline);
</script>
</body>
</html>

```

980.html (add?)

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 Bearbeiten von Ereignissen und Stylen.

Todo <http://leafletjs.com/reference-1.1.0.html#featuregroup>

todo mindestanforderung, methoden überall aufzahlen?

todo was habe ich gemacht, popup nicht mehr bei Marker sondern bei Feature

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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 marker2 = L.marker([51.27264, 6.26469],
{
    title:"Marker2",
    alt:"Ich bin ein anderer 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!');
myfeatureGroup.setStyle({color:'red', opacity:0.5, weight:8})
//myfeatureGroup.on('click', function() { alert('Ein
Gruppenmitglied wurde angeklickt!'); })
//myfeatureGroup.on('click', function()
{ myfeatureGroup.removeLayer(polyline); })
myfeatureGroup.addLayer(marker2);
</script>
</body>
</html>
```

979.html

Popups

<http://leafletjs.com/reference-1.1.0.html#popup>

```

<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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>");

// popup offen zeigen
// marker1.openPopup();

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

var popup2 = L.popup({keepInView:true})
.setLatLng([49.27264, 5.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein anderes
Popup.</p>')
.openOn(mymap);
*/
/* mehrerer popup
var popup1 = L.popup()

```

```

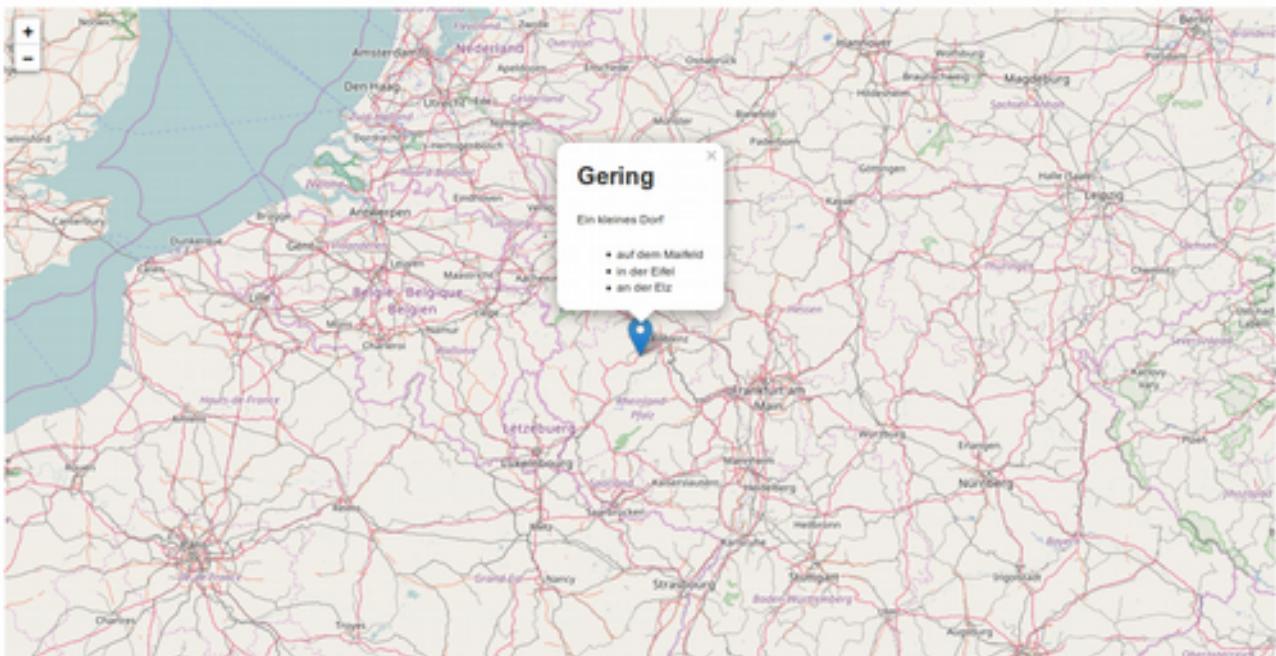
.setLatLng([49.27264, 6.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein Popup.</p>')
.addTo(mymap);

var popup2 = L.popup({keepInView:true})
.setLatLng([49.27264, 5.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein anderes
Popup.</p>')
.addTo(mymap); */

</script>
</body>
</html>

```

978.html



985.png

todo <https://stackoverflow.com/questions/45389015/how-to-show-many-popups-for-leaflet-markers>

todo link zu später makrers

Mobil

Ein großer Vorteil von JavaScript ist es, dass Landkarten in einem aktuellen Standardbrowser ohne externe Applikationen oder Plugins angezeigt werden können. Leaflet unterstützt eine Menge Geräte (todo link). Jede Website, die

eine Leaflet Karte anzeigt, kann gleich programmiert werden. Es muss nichts besonders für ein Gerät beachtet. Es sei denn, Sie möchten auf einem Gerät etwas anders anzeigen.

Das Spannende bei mobilen Anwendungen ist sicherlich die Funktion `locate()` <http://leafletjs.com/reference-1.1.0.html#map-locate>.

Diese Funktion versucht, den Benutzer mit der W3C Geolocation API zu lokalisieren. Die [W3C Geolocation API](#) ist eine einheitliche Webbrowser-Programmierschnittstelle zum Ermitteln des geografischen Standorts des zugehörigen Endgeräts.

Todo <https://www.w3.org/TR/geolocation-API/>

HTML und CSS

Sie wollen Ihre Leaflet Karte auch für mobile Geräte optimal konfigurieren? Als Erstes sollten Sie sicherstellen, dass die Karte passend angezeigt wird. Sie möchten nicht, dass jemand der die Karte über ein Gerät mit einem kleinen Display aufruft, nur einen Teil der Karte sieht und er diese verschieben muss, um Randbereiche zu sehen. Ideal ist es, wenn die vollständige Karte im Display angezeigt wird.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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);
<!DOCTYPE HTML>
<html>
<head>
```

```
<title>Eine OSM 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, user-scalable=no" />
<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 Punkt entfernt.").openPopup();
L.circle(e.latlng, radius).addTo(mymap);
//console.log(e);
}
```

```
}

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

</script>
</body>
</html>
```

977.html

Todo was habe ich gemacht

- Ich habe die Abstände, also padding und margin, auf null gesetzt. So passt sich alles genau an die Seite an und es wird kein Platz mit Rändern verschwendet.
- Außerdem haben wir die Höhe auf 100% gesetzt, damit diese voll ausgenutzt wird. Den fixen Wert für die Höhe habe ich entfernt. So verschwindet auch die Bildlaufleiste am rechten Rand.
- Viewport. Die Seite kann nicht verkleinert oder vergrößert werden – zoomen ist aber immer noch möglich!

Todo Exkurs padding margin. Media query.

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 einen am meisten interessiert. Ideal wäre es also, wenn die Karte sich sofort so öffnet, dass der aktuelle Standort in der Mitte der Karte dargestellt wird. Und hier müssen Sie das Rad natürlich nicht selbst neu erfinden – Leaflet bietet Ihnen Funktionen die Sie nutzen können.

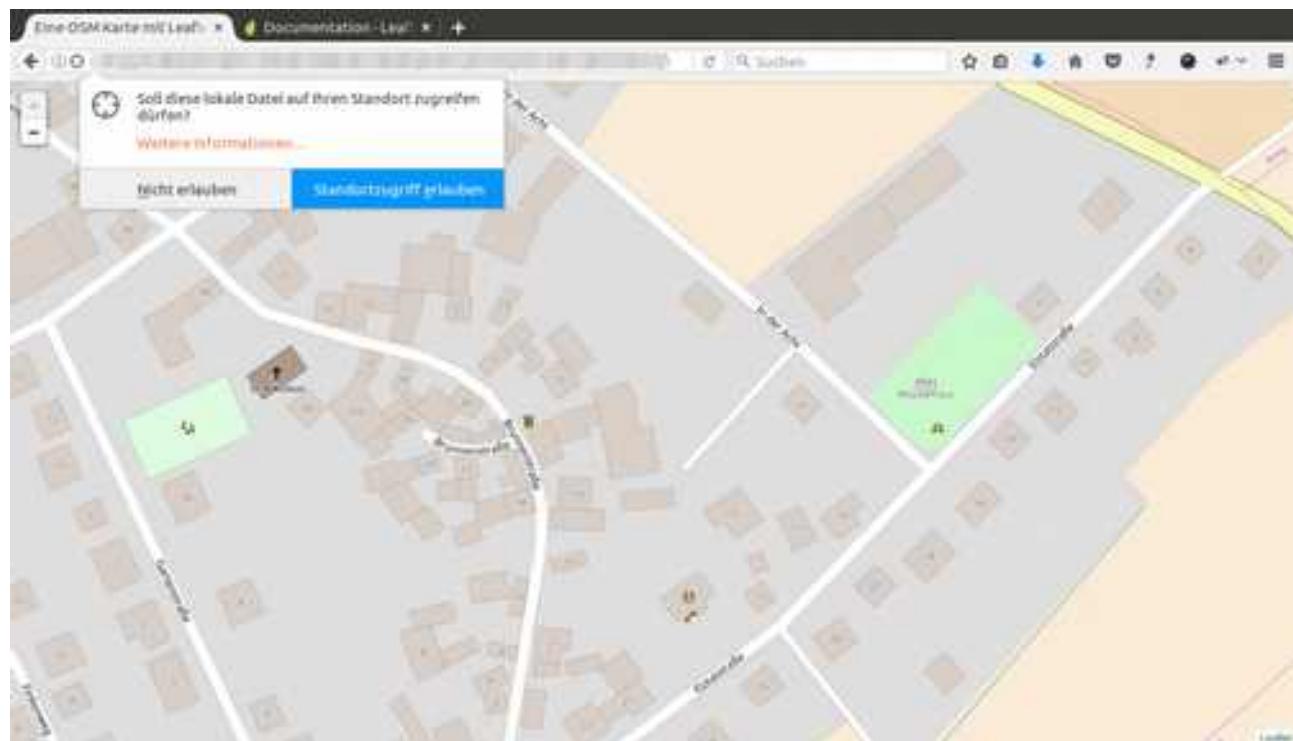
Todo maxzoom1 = ganze welt

Todo Auch 977.html

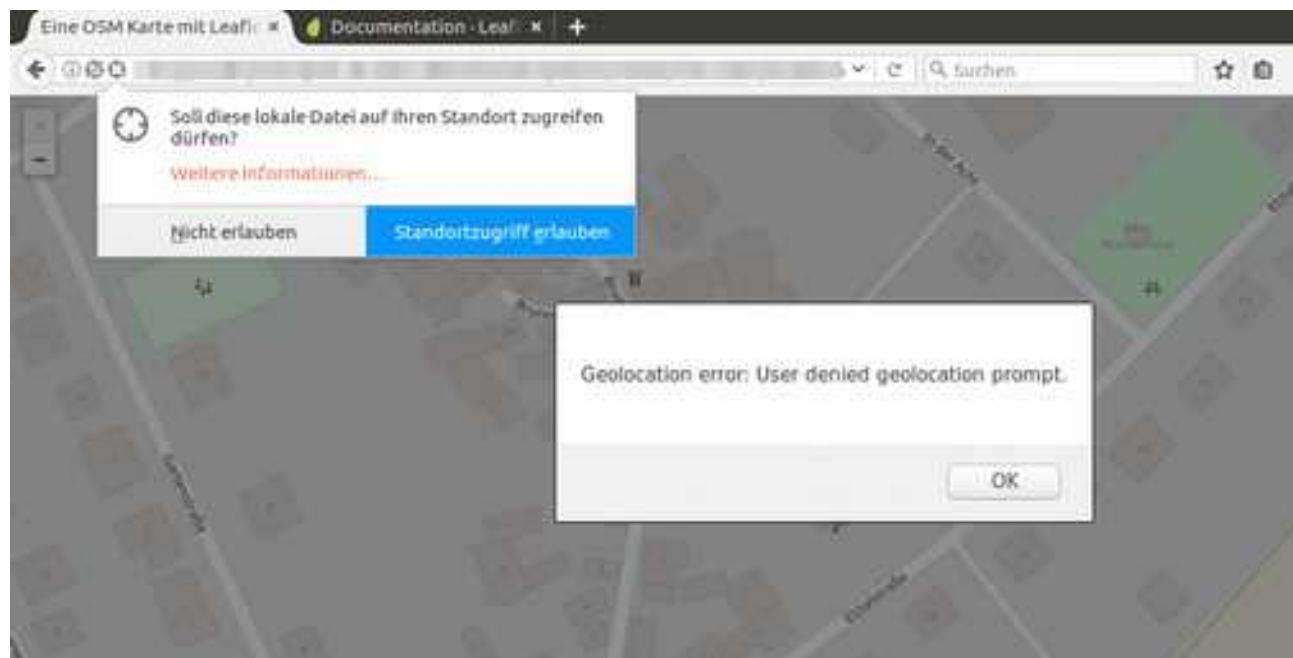
todo <http://leafletjs.com/reference-1.1.0.html#locate-options> watch() →
überleitung ...

todo <http://leafletjs.com/examples/mobile/>

todo <https://www.mozilla.org/de/firefox/geolocation/>



984.png



984a.png



984b.png

Exkurs Variabel e Werte abfragen: 984c → aber auch in Dokumentation:
<http://leafletjs.com/reference-1.1.0.html#locationevent>

Events

Bisher haben wir nur mit statischen Daten gearbeitet. (todo habe ich schon beispiel gebracht mit dem marker dynamisch erstellt werden?) In der Welt passiert aber ganz schön viel und Sie möchten mit Ihrer Karte sicherlich auf das ein oder andere Ereignis reagieren. Zum Beispiel möchten Sie vielleicht eine Pop-up Text ändern, wenn der Marker zum Pop-up bewegt wird.

Leaflet bietet Ihnen 34? Ereignisse, die Sie nutzen können. (Todo ereignisse auflisten)

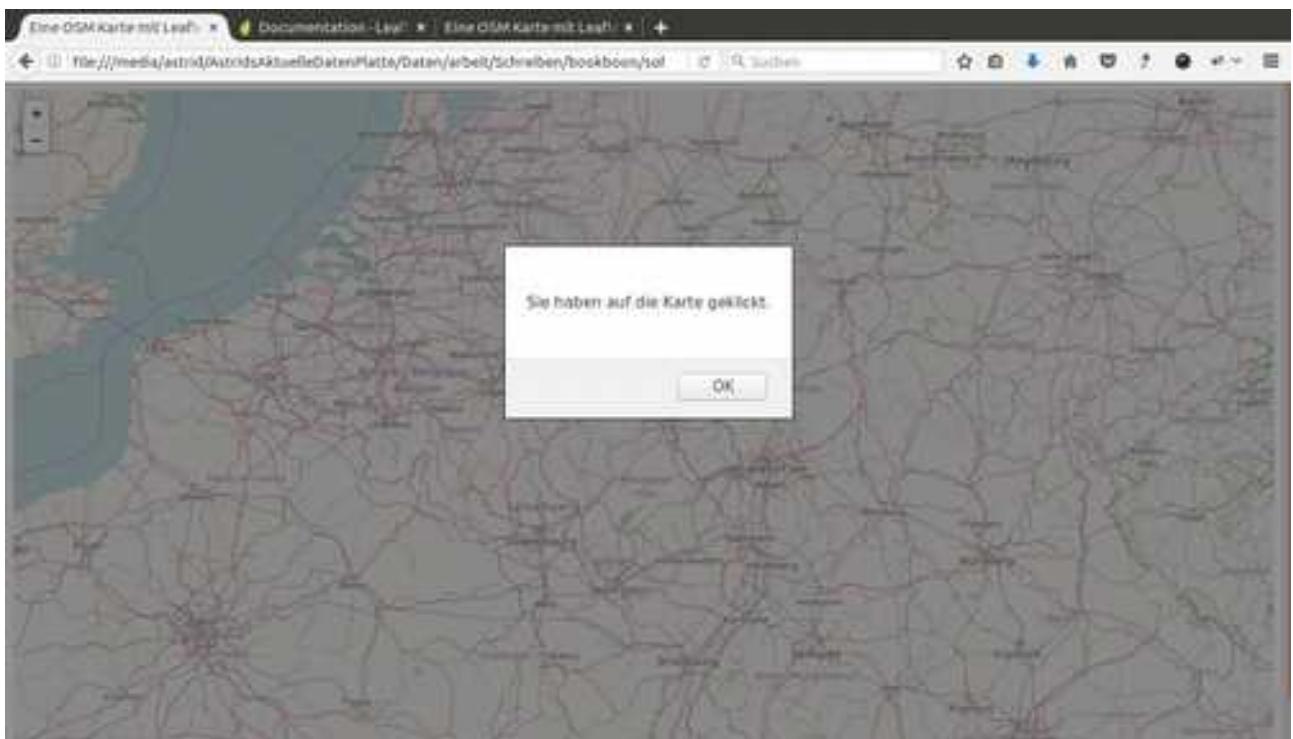
- Layer events
 - baselayerchange LayersControlEvent Fired when the base layer is changed through the layer control.
 - overlayadd LayersControlEvent Fired when an overlay is selected through the layer control.
 - overlayremove LayersControlEvent Fired when an overlay is deselected through the layer control.

- layeradd LayerEvent Fired when a new layer is added to the map.
 - layerremove LayerEvent Fired when some layer is removed from the map
- Map state change events
 - zoomlevelschange Event Fired when the number of zoomlevels on the map is changed due to adding or removing a layer.
 - resize ResizeEvent Fired when the map is resized.
 - unload Event Fired when the map is destroyed with remove method.
 - viewreset Event Fired when the map needs to redraw its content (this usually happens on map zoom or load). Very useful for creating custom overlays.
 - load Event Fired when the map is initialized (when its center and zoom are set for the first time).
 - zoomstart Event Fired when the map zoom is about to change (e.g. before zoom animation).
 - movestart Event Fired when the view of the map starts changing (e.g. user starts dragging the map).
 - zoom Event Fired repeatedly during any change in zoom level, including zoom and fly animations.
 - move Event Fired repeatedly during any movement of the map, including pan and fly animations.
 - zoomend Event Fired when the map has changed, after any animations.
 - moveend Event Fired when the center of the map stops changing (e.g. user stopped dragging the map).
- Popup events
 - popupopen PopupEvent Fired when a popup is opened in the map
 - popupclose PopupEvent Fired when a popup in the map is closed

- autopanstart Event Fired when the map starts autpanning when opening a popup.
- Tooltip events
 - tooltipopen TooltipEvent Fired when a tooltip is opened in the map.
 - tooltipclose TooltipEvent Fired when a tooltip in the map is closed.
- Location events
 - locationerror ErrorEvent Fired when geolocation (using the locate method) failed.
 - locationfound LocationEvent Fired when geolocation (using the locate method) went successfully.
- Interaction events
 - click MouseEvent Fired when the user clicks (or taps) the map.
 - dblclick MouseEvent Fired when the user double-clicks (or double-taps) the map.
 - mousedown MouseEvent Fired when the user pushes the mouse button on the map.
 - mouseup MouseEvent Fired when the user releases the mouse button on the map.
 - mouseover MouseEvent Fired when the mouse enters the map.
 - mouseout MouseEvent Fired when the mouse leaves the map.
 - mousemove MouseEvent Fired while the mouse moves over the map.
 - contextmenu MouseEvent Fired when the user pushes the right mouse button on the map, prevents default browser context menu from showing if there are listeners on this event. Also fired on mobile when the user holds a single touch for a second (also called long press).
 - keypress KeyboardEvent Fired when the user presses a key from the keyboard while the map is focused.

- preclick MouseEvent Fired before mouse click on the map (sometimes useful when you want something to happen on click before any existing click handlers start running).
 - Other Methods
 - zoomanim ZoomAnimEvent Fired on every frame of a zoom animation
-

Leaflet bietet die Methode on() um auf ein Ereignis zu reagieren.



983.png

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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>
```

976.html

Todo einfaches Beispiel (dirket hierüber) erklären anonyme funktion.

Todo Ein anderes Beispiel erklären link zu wie man e ausgibt. 983a.png - 976a.html → Marker Event.

Todo Beispiel nach closePopupOnClick: false durchsuchen.

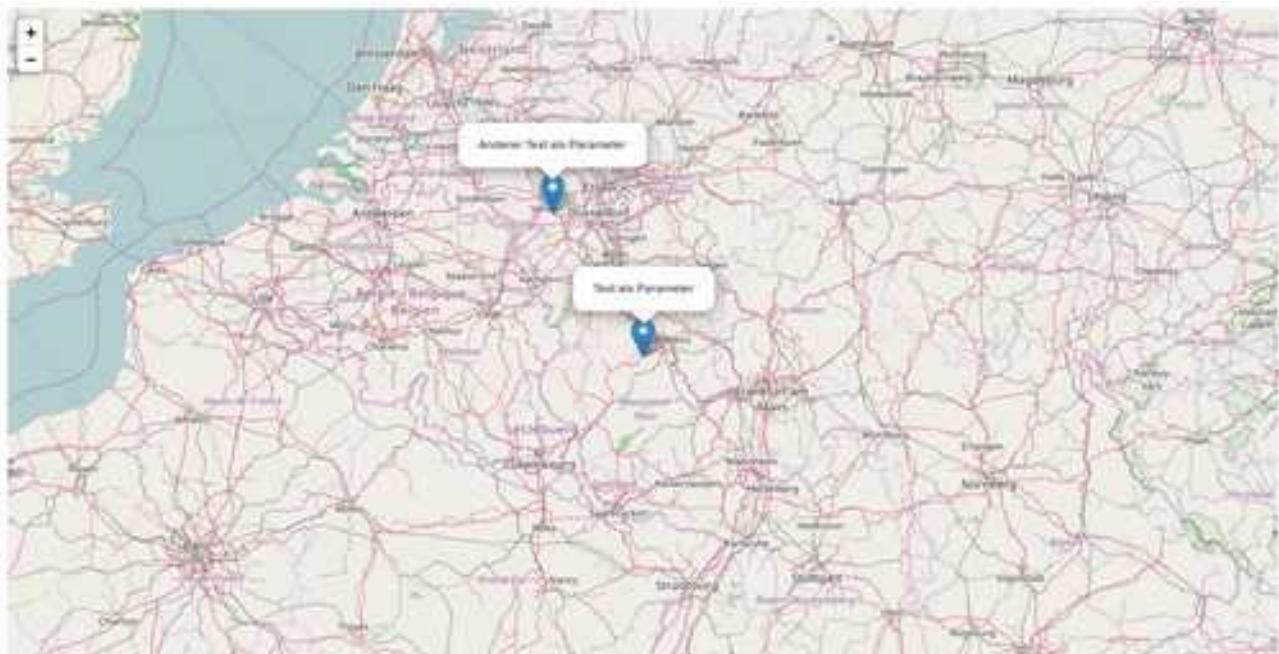
Todo Vielleicht link zu spielplatz: <http://playground-leaflet.rhcloud.com/fijo/edit?html,console,output>

Benutzerdefinierte Funktionen

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

Todo autoClose:false man kriegt die nicht mehr zu :) und keepInView:true

Probieren wir das doch sofort einmal aus. Wir erstellen einen Marker und verbinden diesen mit einem Pop-up.



982.png

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM 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 als
Parameter"));
var marker2 = L.marker([51.27264,
6.26469]).addTo(mymap).bindPopup(createPopup("Anderer Text als
Parameter"));
function createPopup(popuptext) {
```

```
        return L.popup({autoClose:false, keepInView:true,
closeButton:false}).setContent(popuptext);
    }
</script>
</body>
</html>
```

975.html (

todo charset und sprach de überall im code

Interaktives Beispiel

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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>
<button onclick="clean()">Löschen</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);
```

```

var myStorage = localStorage;
function save()
{
myStorage.setItem(document.getElementById("name").value,
document.getElementById("lat").value+", "+document.getElementById("long").value);

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);

for (var i=0; i <localStorage.length;i++)
{
    //alert(localStorage.key(i));
}

}

function clean()
{
for (var i=0; i <myStorage.length;i++)
{
    alert(myStorage.length);
    myStorage.removeItem(localStorage.key(i));
}
}

</script>
</body>
</html>

```

974.html

Todo vielleicht heiß kalt beispiel machen und heiß kalt in id ausgegeben.

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 erst mit dem Schließen des Browsers, sie übersteht das Neuladen und Wiederherstellen einer Webseite. Das Öffnen einer Webseite in einem neuen Tab oder Browserfenster erzeugt jedoch eine neue Sitzung; ein Unterschied zur Funktionsweise von Session-Cookies.

<https://developer.mozilla.org/de/docs/Web/API/Window/sessionStorage>

Todo zusammenfassung?

GeoJson

Todo <https://github.com/johan/world.geo.json>

todo <https://github.com/blog/1528-there-s-a-map-for-that>

todo <http://leafletjs.com/examples/geojson/>

todo <http://geojson.org/>

todo GeoJSON überall gleich geschrieben?

GeoJson verstehen

GeoJson ist ein

Bevor es GeoJSON gab, gab es JSON (JavaScript Object Notation) und bevor es JSON gab, 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 dass Menschen nacharbeiten müssen. Dies wurde in Zeiten des Internets immer wichtiger.

Todo https://www.opendata-hro.de/group/geo?res_format=GeoJSON

Wie und warum ist das Format GeoJSON entstanden?

GeoJSON, JSON und XML können

- ... von einem Menschen gelesen und verstanden werden.

- ... hierarchisch gegliedert werden. Das bedeutet, dass Werte innerhalb von anderen Werten dargestellt werden können.
- ... leicht zu erlernen.
- ... von vielen Programmiersprachen analysiert und genutzt werden.
- ... mit einem XMLHttpRequest ausgetauscht werden.

Schauen wir uns in den nächsten Kapiteln die einzelnen Formate genauer an und Ihnen wird klar, welche Vorteile das Format JSON – beim Arbeiten mit [Geodaten](#) das Format GeoJSON – gegenüber XML bringt.

Todo habe ich geodaten schon definiert? Dann link sonst definition:
<https://de.wikipedia.org/wiki/Geodaten>

XML

XML ist eine Sprache, die die Struktur von Daten beschreibt. Anhand der Tags werden die Daten beschrieben und erklärt. Durch das Tag-System von XML werden oft kleine Datenbestände sehr aufgebläht und unübersichtlich. Zusätzlich ist das Ansprechen einzelner Knoten (XML-Nodes) in einer XML-Datei nicht immer leicht.

JSON

JSON ist die Kurzform für den englischen Begriff JavaScript Object Notation.

JSON ist im Grunde genommen nichts anderes als die Festlegung auf eine bestimmte Syntax – also eine Syntax-Konvention. Mit JSON werden Objekte aus strukturierten Daten definiert.

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. (todo link wo wir das machen) Der Zugriff auf die einzelnen Eigenschaften kann dann über normalen Attribut Zugriff geschehen.

XML muss mit einem XML-Parser analysiert werden. JSON kann durch eine einfache JavaScript Funktion analysiert werden. In JSON gibt es kein Ende-Tag, deshalb ist JSON kompakter. Dies hat zur Folge, dass JSON schneller gelesen und verarbeitet werden kann.

(todo umschreiben <http://www.webmasterpro.de/coding/article/json-als-xml-alternative.html>)

Ein konkretes Beispiel

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

95 Zeichen

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

73 Zeichen

Und warum nun auch noch **GeoJSON**?

GeoJSON ist ein JSON Format, dass sich auf Geodaten spezialisiert hat. Es kann gut mit Punkten, Linien und Polygonen in einem Koordinatensystem umgehen. Im Kapitel todo haben wir gesehen, dass das Arbeiten mit Geodaten im Grunde genommen nichts anderes ist.

Todo Sphäre: [https://de.wikipedia.org/wiki/Sph%C3%A4re_\(Mathematik\)](https://de.wikipedia.org/wiki/Sph%C3%A4re_(Mathematik))

GeoJSON ist zu einem sehr beliebten Datenformat unter vielen GIS-Technologien und -Diensten – es ist einfach, leicht, unkompliziert. Hier erwähne ich es, weil Leaflet (todo gucken ob leaflet überall gleich geschrieben.) sehr gut im Umgang mit Daten im Format GeoJSON ist. Im Kapitel *GeoJson in Leaflet* erfahren Sie, wie Sie Elemente mithilfe von GeoJSON auf Ihrer Karte anzeigen können.

Todo <http://jsoneditoronline.org/>

GeoJson erkunden

Mit GeoJSON können Sie eine Vielzahl von geografischen Datenstrukturen in einer maschinenlesbaren Sprache kodieren. Ein GeoJSON-Objekt kann eine Geometrie, ein Feature oder eine Sammlung von Features darstellen.

Schauen wir uns [GeoJSON](#) hier in diesem Kapitel einmal ganz genau an. Das Verständnis dieser Konzepte bringt viele Vorteile. Es hilft Ihnen auch, die Arbeit mit Geodaten im Allgemeinen zu verstehen: Die Grundkonzepte hinter GeoJSON sind schon seit Anfang ein Teil von [Geoinformationssystemen](#).

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

Geometrie

Geometrien sind Formen. Alle Geometrien in GeoJSON bestehen aus einer oder mehreren Koordinaten.

Position

Das wichtigste Element beim Arbeiten mit Geodaten ist die Beschreibung des Punktes auf der Erde, dem die Geodaten zugeordnet werden. Diesen Wert kennen wir auch unter dem Namen Koordinate.

Eine Koordinate ist eine Zahlenkombination. Jede Zahl in einer Koordinate steht für eine Dimension. Wir beschränken uns hier im Buch auf zwei Dimensionen: Längengrad und Breitengrad. GeoJSON kann aber drei Dimensionen aufnehmen, nämlich den Längengrad, den Breitengrad und die Höhe.

todo GPS Exkurs zwei weitere Dimensionen Höhe und Zeit und vielleicht Link zu Beginn mit Exkurs langen und breitengrade es gibt noch eine dritte Dimension die bei GPS Koordinaten eine Rolle spielt

Die Koordinaten werden in JSON in im Dezimalformat formatiert. In Kapitel (todo) hatte ich Ihnen schon erklärt, dass es auch andere Formate zum Beschreiben eines Punktes auf der Erde gibt. Aber, auch wenn die Formatierung $8^{\circ}10'23''$ schön aussieht, können Sie sich vorstellen, dass die Handhabung dieses Formats bei Berechnungen mit einem Computer umständlich ist. Darum also das Dezimalformat, welches einen Kompromiss für Menschen und Maschinen bietet.

Eine Position in GeoJSON ist ein Array von Zahlen. Diese Zahlen stellen einen Punkt auf der Erde dar. Konkret sieht eine Position in GeoJSON so aus:

[Länge, Breite, Höhe]

Vielleicht 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. Um dies 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 Längengrad dem X-Wert und der Breitengrad 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. todo <https://macwright.org/lonlat/>

Früher erlaubte GeoJSON die Speicherung von mehr als 3 Zahlen pro Position. Teilweise wurde diese Möglichkeit auch genutzt. Es wurden beispielsweise Sportdaten wie die Herzfrequenz mit der Position gespeichert. Da dies nicht der Sinn einer Position ist, führte es teilweise zu Problemen in GeoJSON Anwendungen. In der [neuen Spezifikation](#) ist das Speichern von mehr als drei Werten pro Position nun verboten.

Point

Ein Point – also ein Punkt – ist die einfachste Geometrie in GeoJSON. Die genaue Schreibweise ist

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

Multipoint

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

LineStrings

Um eine Linie darzustellen, benötigen Sie mindestens zwei Punkte. Die Linie ist dann die Verbindung zwischen diesen Punkten.

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

MultiLineString

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

Linien und Punkte sind die beiden einfachsten Geometriegerüste. Bei beiden müssen Sie nicht viele geometrische Regeln beachten. Ein Punkt kann irgendwo an einem Ort liegen und eine Linie kann beliebige Anordnungen von Punkten enthalten. Eine Linie kann sich sogar 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!



963.png

todo Lizenz oder eigene Bilder machen: By Mwtoews (Own work) [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

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

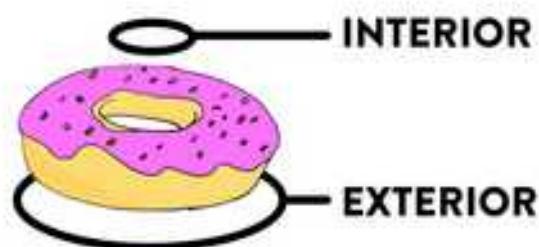
Die Koordinatenliste für Polygone enthält eine Ebene mehr, als die der LineStrings. Bei einfachen Polygonen ist der Sinn dieser Ebene nicht offensichtlich. Auf den ersten Blick könnten Sie meinen, dass es einfacher wäre das Polygon genau wie die Linie zu erstellen. Dass es sich um ein Polygon

handelt, ist über den Eintrag bei der Eigenschaft Typ klar. Wenn es sich um den Typ Polygon handelt, wird die Linie einfach geschlossen! Der erste Blick ist oft trügerisch. Wir benötigen die zusätzliche Verschachtelung um Löcher in die Fläche zeichnen zu können. Polygone sind in GeoJSON mehr als nur geschlossene Linien. Sie haben auch einen Innenbereich und dieser kann Löcher haben.

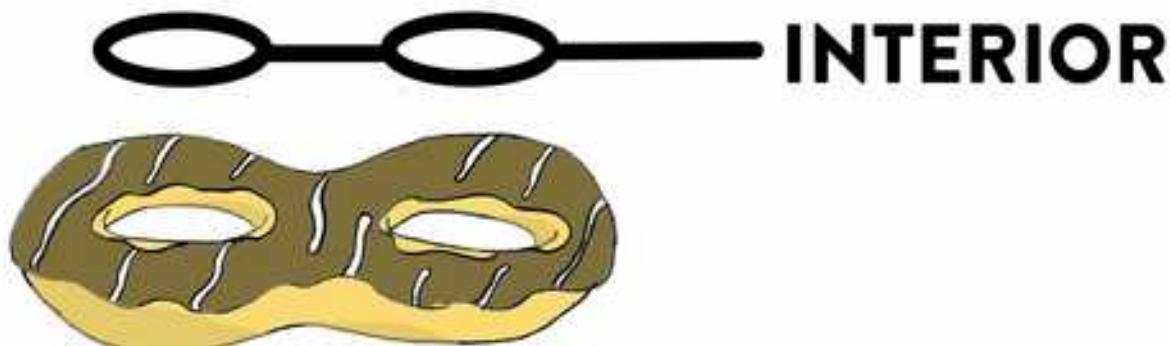
Aus diesem Grund führen Polygone einen neuen Begriff ein, nämlich den Begriff LinearRing.

Ein LinearRing ist ein geschlossener LineString mit 4 oder mehr Positionen. Obwohl ein LinearRing nicht explizit als GeoJSON-Geometrie-Typ eingeführt ist, wird der Begriff in der [Polygon-Geometrie-Typ-Definition](#) erwähnt.

Ein LinearRing ist entweder die äußere Ringposition, die die äußere Kante des Polygons bildet und definieren, welche Teile gefüllt sind – oder ein Innenring, der die Teile des Polygons definieren, die leer sind. Es kann nur einen äußeren Ring geben und dies ist immer der erste LinearRing. Es kann eine beliebige Anzahl von Innenringen geben, einschließlich null Innenringen. Wenn das Polygon über keinen Innenring verfügt bedeutet dies, dass das Polygon keine Löcher hat.



962a.png



und 962b.png → noch jpg

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

{ "type": "Polygon",
  "coordinates": [
    [[35, 10], [45, 45], [15, 40], [10, 20], [35, 10]],
    [[20, 30], [35, 35], [30, 20], [20, 30]],
    [[15, 21], [20, 23], [20, 17]]
  ]
}
```

Vielleicht ist Ihnen aufgefallen, dass die erste und die letzte Koordinate jedes LinearRings gleich ist. Die Wiederholung der Koordinate ist heute im Format GeoJSON nicht mehr notwendig. Ein LinearRing wird automatisch geschlossen.

Multipolygon

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

{ "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]]
        ]
    ]
}

```

Mehrere Geometrien

Nun, da wir reden darüber, wie Daten die Welt beschreiben können, können Sie einige Einschränkungen dieses Ansatzes bemerken. Jeder der grundlegenden LineString-, Polygon-, Point-Typen ist ideal für die Darstellung einer einzigen Form, aber oft die physische Welt enthält Objekte, die nicht nur eine einzige zusammenhängende Sache sind. Zum Beispiel haben die Vereinigten Staaten, zusammen mit vielen anderen Ländern, mehrere getrennte Teile. Wir verweisen auf alle von ihnen als "Die Vereinigten Staaten", und Software, die Highlight "Die Vereinigten Staaten" sollte in der Lage sein, dies zu wissen und auch Highlight Alaska, Hawaii und der Rest.

Hier kommen die verschiedenen Geometrien herein. GeoJSON hat Versionen von jedem der drei Grundtypen mit Multi auf der Vorderseite: MultiPolygon, MultiLineString, MultiPoint. Gemeinsam geben sie uns etwas von einer Lösung für dieses Problem.

Die Art, wie Multi-Features erstellt werden, ist das gleiche über alle Typen: Alles bewegt sich einen Schritt des Verschachtelns. Die Koordinaten eines einzelnen Punktes werden als [0, 0] dargestellt, so dass ein MultiPoint davon und ein anderer Ort aussehen könnte [[0, 0], [1, 1]].

In selteneren Fällen haben Sie eine Reihe von verschiedenen Arten von Geometrien, die alle auf die gleiche Sache beziehen. Dazu gibt es in GeoJSON den Typ GeometryCollection. Im nachfolgenden Programmcode habe ich diesen schon ein Feature Objekt eingebunden. Das Feature Objekt erkläre ich Ihnen im nächsten Kapitel.

```
{
  "type": "Feature",

```

```

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

```

Geometriekollektionen gibt es in der Praxis nur sehr selten: Meist ist es so, dass eine Geometrien unterschiedliche Eigenschaften besitzt. Aus diesem Grund empfiehlt die aktuelle GeoJSON-Spezifikation GeometryCollections nur in besonderen Fällen zu verwenden.

Features

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 ist Reichstag. Ein anderes Polygone ist die Grenze von Deutschland. Und bei der Arbeit mit den Geometrien ist es wichtig zu wissen, welche Geometrie was ist und, welche Eigenschaften sie hat. In GeoJSON kann genau dies mit einem Objekt des Typs Feature erreicht werden.

Das nachfolgende Programmcodebeispiel enthält ein ganz einfaches Feature:

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

```
}
```

Das nachfolgende Programmcodebeispiel enthält ein etwas komplexeres Feature:

```
{
  "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
  }
}
```

Die Eigenschaften, die mit einem Feature angehängt werden, können jede Art von [JSON-Objekt](#) sein.

FeatureCollection

Wir haben alle Arten von Dingen abgedeckt, die in GeoJSON sein können, aber eins: FeatureCollection ist die häufigste Sache, die du auf der obersten Ebene von GeoJSON Dateien im Feld sehen wirst.

Eine FeatureCollection mit unserem Beispiel einer Feature sieht aus wie:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [0, 0]
      }
    }
  ]
}
```

```

        },
        "properties": {
            "name": "Der Name des Features"
        }
    }
]
}

```

Ein Objekt vom Typ FeatureCollection enthält ein Schlüssel-Wert-Paar. Der Wert ist ein Array mit Feature-Objekten 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 Objekten vom Typ? Ein Objekt vom Typ FeatureCollections macht viel Sinn in Bezug auf die Gemeinsamkeit zwischen verschiedenen GeoJSON-Typen.

- GeoJSON-Objekte sind Objekte, keine Arrays oder Primitive
- GeoJSON-Objekte haben eine "type" -Eigenschaft
Das ist wirklich nett für Implementierungen: Sie brauchen nicht zu erraten, welche Art von GeoJSON Objekt sie suchen – Sie lesen einfach die Eigenschaft Type aus.

Todo gucken ob ich überall diese Geometrien alle verwende

todo <http://geojsonlint.com/>

Die Grenzen von GeoJSON

Die Vorteile von GeoJSON hatte ich Ihnen in Kapitel todo schon beschrieben. 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 werden die Datumswerte von Shapefiles unterstützt, aber nicht in JSON. (Todo link zu shapefiles)
- GeoJSON hat kein Konstrukt für die Anzeige von Pop-up-Fenstern wie Titel oder Beschreibung. Sie werden aber später sehen, wie Leaflet sie bei der Erstellung von Pop-up-Fenstern unterstützt.
- 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 eine LineString-Darstellung eines Trainingslaufs haben und Ihre GPS-Uhr 1.000 verschiedene Punkte während dieses Laufs protokolliert hat, zusammen mit Ihrer Herzfrequenz und der Dauer zu diesem Zeitpunkt gibt es keine klare Antwort für die Darstellung dieser Daten. Sie können zusätzliche Eigenschaften als Array mit der gleichen Länge wie das Koordinaten-Array speichern, aber keine Option wird durch das Ökosystem der Werkzeuge gut unterstützt.

Todo irgendwo Problem, warum <https://macwright.org/2015/03/23/geojson-second-bite.html#the-180th-meridian> Vielleicht bei multiline

GeoJson in Leaflet

Leaflet unterstützt alle GeoJSON-Typen. In der Regel werden Sie aber überwiegend mit Features und FeatureCollections arbeiten. Und diese beiden Typen unterstützt Leaflet bevorzugt. Sie möchten ja sicher nicht nur Geometrie Objekte auf Ihrer Karte anzeigen, sondern auch die Eigenschaften – also weitere Informationen – zu diesen Objekten.

Beginnen wir mit einem übersichtlichen Beispiel: Der einfachste Weg GeoJSON in Ihrer Karte zu nutzen, ist die Verwendung als Variable direkt – fest programmiert. Das nachfolgende Programmcodebeispiel enthält einen Punkt. Es ist der gleiche Punkt, den wir (todo link) als Marker auf der Karte angezeigt haben. Sobald Sie diesen in GeoJSON formatierten Punkt in einer Variablen gespeichert haben, können Sie diesen ganz einfach zur Karte hinzufügen. (todo TeilCodebeispiele auch in Text.)

todo Achtung lat und lon sind vertauscht!

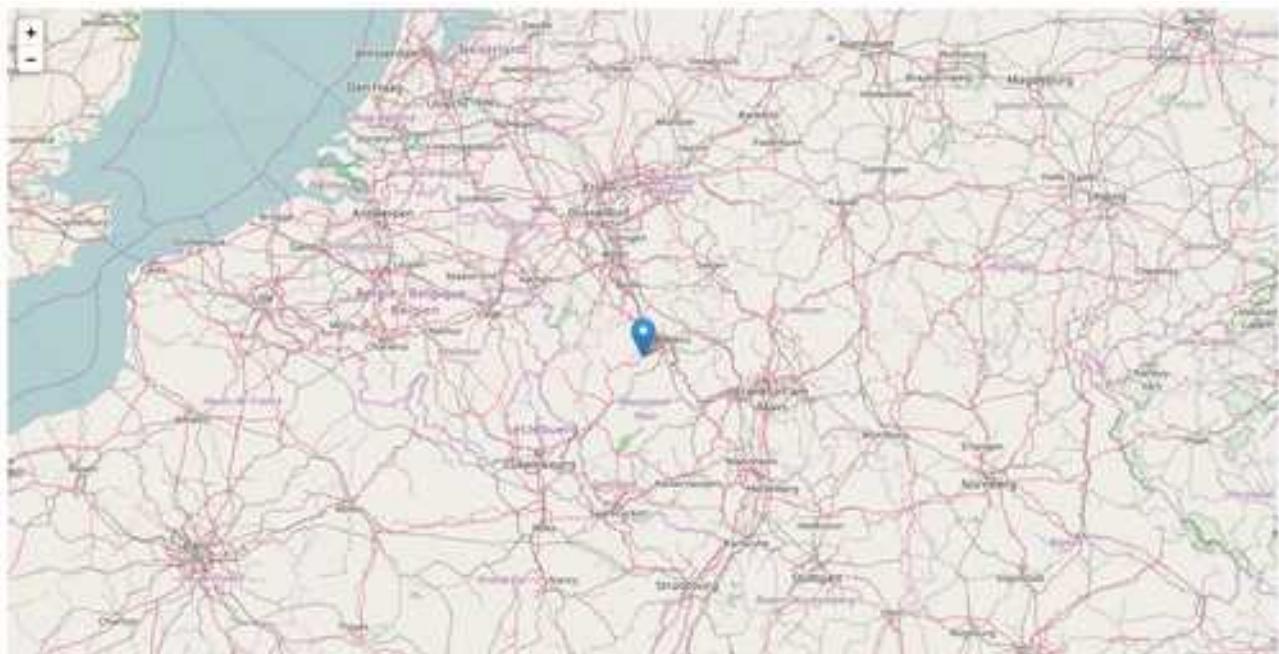
Was macht der Code. Das nachfolgende Beispiel erstellt eine geoJSONLAYER Varialbe. Diese Variabel ist eine Instanz der Klasse L.geoJSON(). Dieser kann ein Objekt vom Typ GeoJSON als Parameter übergeben und zur Karte hinzugefügt werden. (todo verkettet hinzufügen oder allein erklären? Oder habe ich schon erklärt?).

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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>
```

973.html

Als Ergebnis sehen Sie eine Karte, auf der ein Marker abgebildet ist.



981.png

Der Marker enthält noch kein Pop-up. Todo link zu erklärung dazu.

todo vielleicht ein kleines Beispiel mit allen geometrien.

Multiple Geometrien in Leaflet

Das Beispiel im letzten Kapitel enthielt ausschließlich einen Punkt. Geodaten bestehen aber aus ganz unterschiedlichen Formen und Leaflet kann GeoJSON Daten verarbeiten, die aus unterschiedlichen Geometrien zusammen bestehen. Sie haben im vorhergehenden Kapitel schon alle Typen, die GeoJSON zur Verfügung stellt, kennengelernt. Und Sie möchten ja nun sicher

- Punkte (beispielsweise Adressen und Point of Interest),
- Linien (beispielsweise Straßen und Flüsse),
- Polygone (beispielsweise Ländergrenzen und Landnutzungen) sowie
- Typensammlungen dieser Geometrien

auf Ihrer Karte anzeigen.

Das nächste Beispiel zeigt Ihnen, wie Sie einen Punkt, ein Polygon und eine Linie mithilfe einer Variablen im GeoJson Format in einem Schritt auf Ihrer Karte anzeigen könnten.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
```

```
<title>Eine OSM 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"}`}
  ]
}
```

```

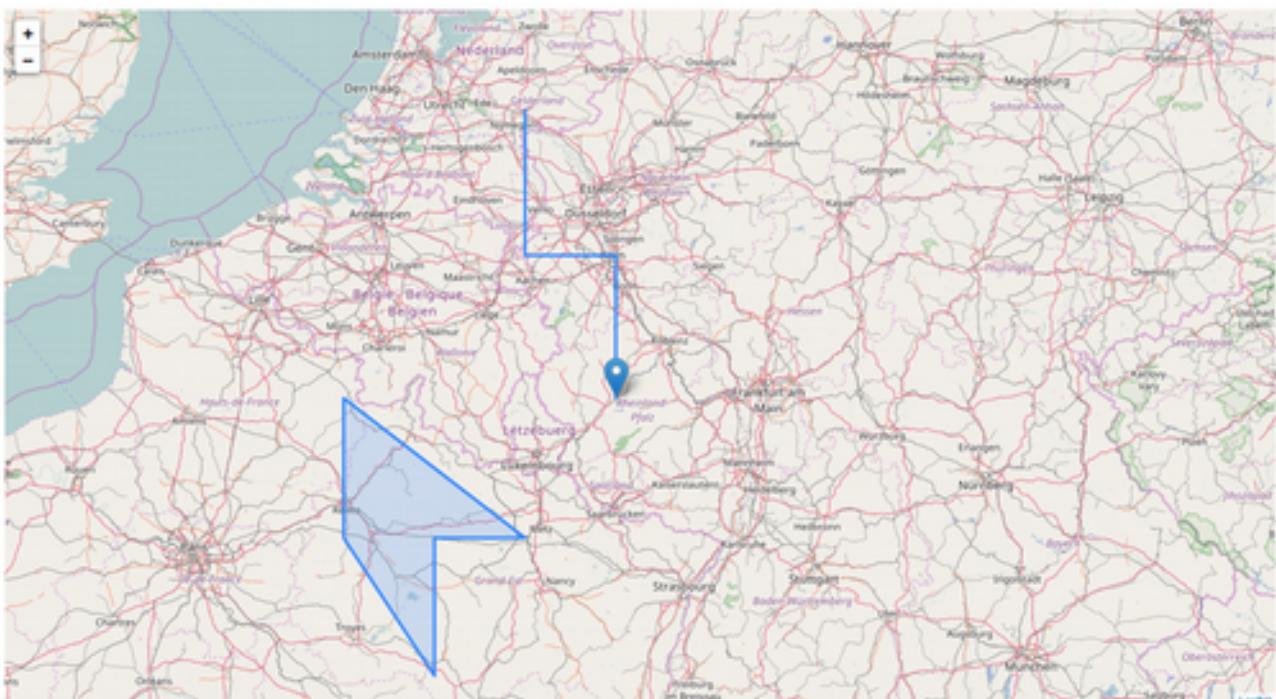
        }
L.geoJSON(geojsonFeatureCollection).addTo(mymap);
</script>
</body>
</html>

```

972.html

todo auch mit donut :) auch auf Featurecollection und geometriecollection eingehen. Mit Donat würde klick auf Fläche in der Mitte kein popup öffnen.

Todo noch ändern, dass GeoJson Polygon schließen muss , leaflet fordert dies aber nicht.



961.png

Donuts

GeoJSON aus leaflet exportieren

So, und nun machen wir genau das Gegenteil des letzten Kapitels. Eine jede der Geometrien, die wir uns im Kapitel angesehen haben, hat eine Methode mit dem Namen `toGeoJson()`. Und diese Methode tut genau das, was der Name schon vermuten lässt. Die Geometrie wird in ein GeoJSON Objekt umgewandelt.

Das folgende Beispiel zeigt Ihnen, wie Sie einen Marker in einen GeoJSON Layer konvertieren können. Zunächst erstellen Sie, genau wie wir es im ersten Kapitel (todo link?) gemacht haben, einen Marker.

Todo Codeschnipsel einfügen.

Danach wird die Methode `toGeoJSON()` des Markers aufgerufen und der Rückgabewert der Methode wird in der Variablen `todo` gespeichert. Bei der Variablen `todo` handelt es sich nun um ein GeoJSON Objekt.

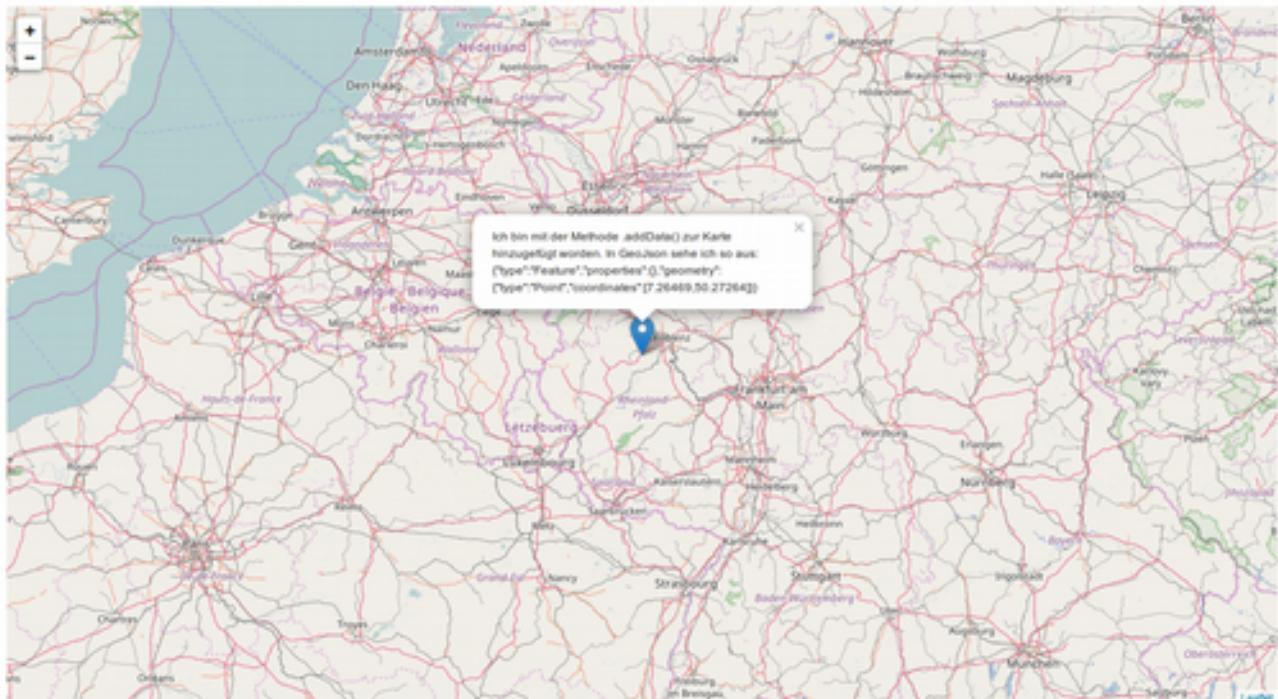
Zu guter Letzt fügen wir das GeoJSON Objekt `todo` zur Karte hinzu.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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>
```

971.html

Mit der `JSON.stringify()` Methode können Sie einen JavaScript-Wert in einen JSON-String konvertieren. Optional werden Werte ersetzt, wenn eine Ersetzungsfunktion angegeben ist, oder, wenn ein Array angegeben ist, dann werden nur die angegebenen Eigenschaften einbezogen. `todo` stringify (https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify)



960.png

Sie werden nun sicher mein Beispiel als umständlich ansehen. Ich habe etwas, was ich ohne zusätzliche Schritte auf der Karten anzeigen könnte, vorher umgewandelt. Dies Vorgehensweise ist nicht der Sinn und Zweck der Leaflet Funktion toGeoJSON. In der Praxis können Sie Website Besuchern mithilfe dieser Funktion ermöglichen, Kartendaten zu exportieren.

Stilen

Ein GeoJSON Layer biete 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 Stiloptionen zugreifen. Die vollständige Liste finden Sie in der Dokumentation von Leaflet. Sehen Sie sich dazu die Optionen zum Klasse Path an: <http://leafletjs.com/reference-1.1.0.html#path>.

Todo Optionen auflisten?

Ein einfaches Beispiel – style:

Der nachfolgende Programmcode zeigt Ihnen an einem Beispiel, wie Sie Styles zusammen mit GeoJSON Layern verwenden können. Im nachfolgenden Programmcode finden Sie ein Funktion, die je nach GeoJSON Objekt eine andere Farbe zurück gibt. Wenn es sich um einen LineString handelt, gibt die Funktion die Farbe Rot zurück, falls ein Polygon vorliegt, antwortet die Funktion mit Violett.

Todo was habe ich genau gemacht?

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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": "Point", "coordinates": [7, 50]},
      "properties": {"prop0": "value0"}
    }
  ]
}
```

```

"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"} }
}
]
}
}

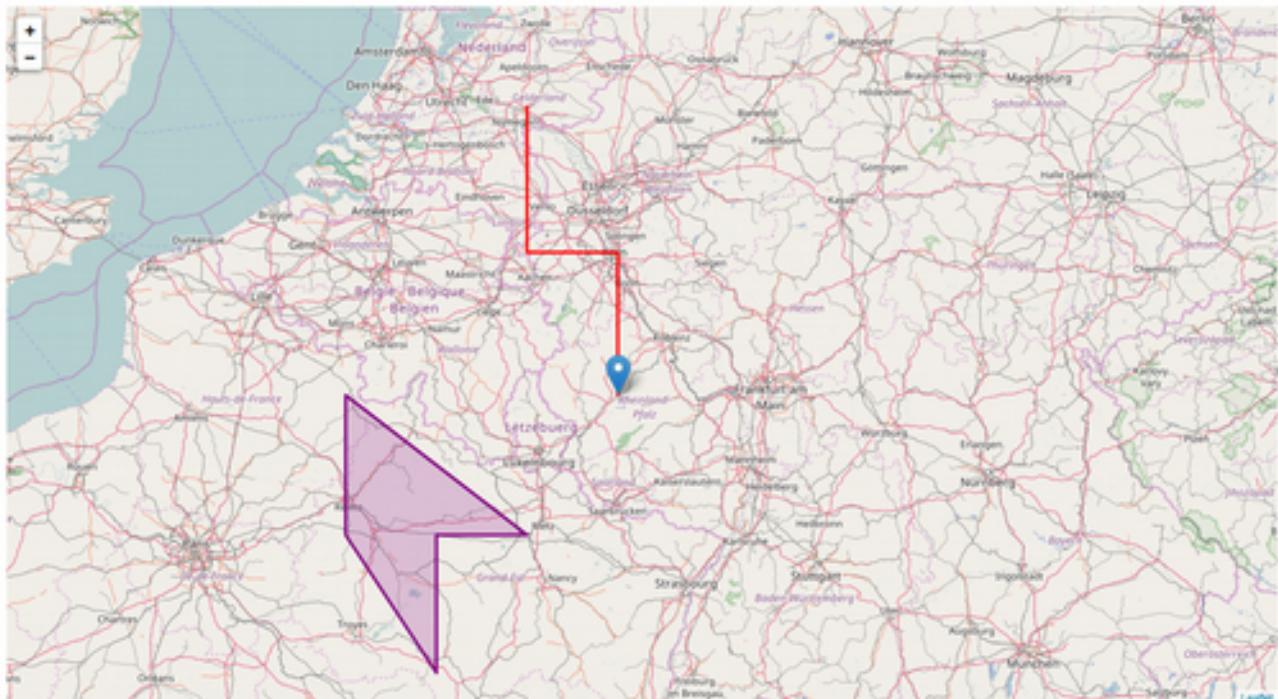
var geoJsonLayer = L.geoJson(geojsonFeatureCollection,
{style:styleFunction}).addTo(mymap);
</script>
</body>
</html>

```

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.



959.png

Ein komplexeres Beispiel mit Ereignissen

Im vorherigen Kapitel haben Sie gelernt, wie Sie mit der Methode `setStyle()` 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, kennzeichnen will. Oder Sie möchten das man das Objekt, über dem die Maus sich gerade befindet, hervorheben. Genau diese beiden Anwendungsfälle sind Thema im nachfolgenden Programmcode.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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);
});

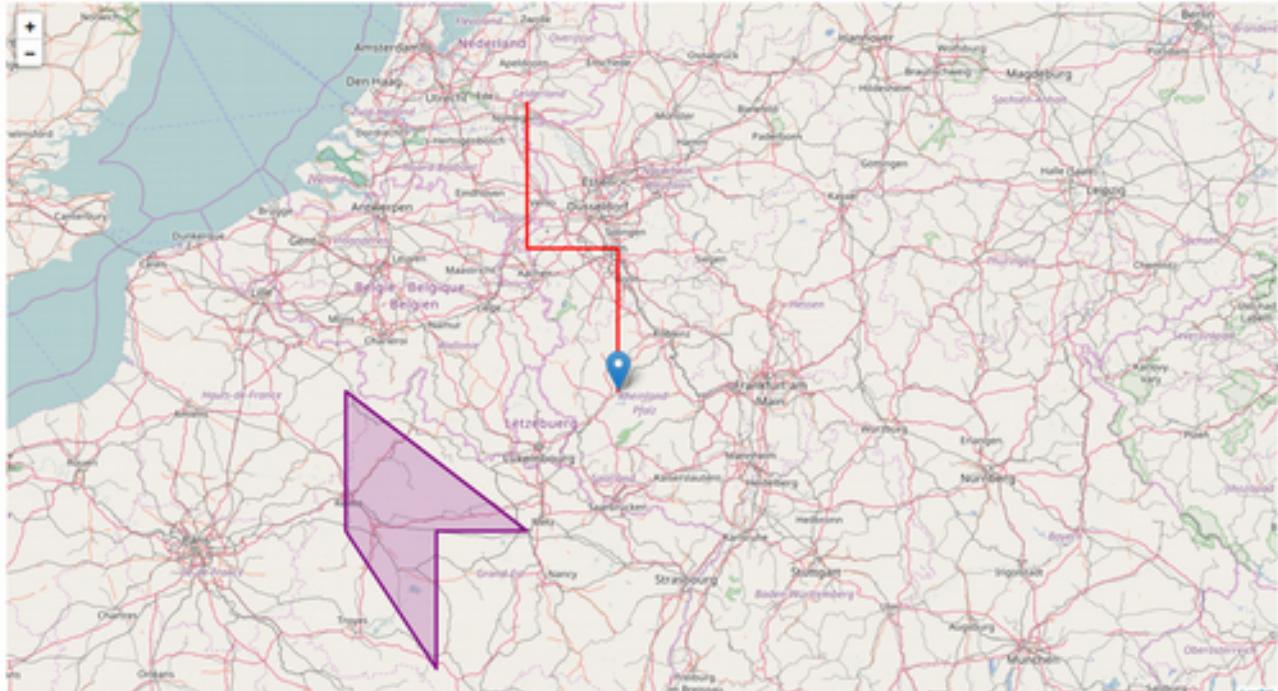
}

</script>
</body>
</html>

```

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.



958.png

todo marker ändert Farbe nicht.

Achtung

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

todo Nun wird alles grün, wenn sich Maus über den Layer bewegt.

Todo Alles Grau nach Hover

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

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()`. Todo ist das richtig ausgedrückt?

OneEachFeature()

`onEachFeature()` ist eine Methode die einmal für jedes im Layer vorhandene GeoJSON Feature aufgerufen wird. Nützlich ist diese Methode zum Anhängen von Ereignissen und Popups an jedes Features. Die Voreinstellung ist, mit den neu erstellten Ebenen nichts zu tun:

968.html

todo: <http://leafletjs.com/reference-1.1.0.html#geojson-oneachfeature>

todo was mache ich m code

```
<!DOCTYPE HTML>  
  
<html lang="de">  
  
<head>  
  
<meta charset="utf-8"/>  
  
<title>Eine OSM 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: true,
  title: "Ein Dorf in der Nähe von Gering"
};

L.geoJson(geojsonFeatureCollection, {
  onEachFeature: function(feature, layer) {
    layer.bindPopup(feature.properties.name);
  }
});
```

```

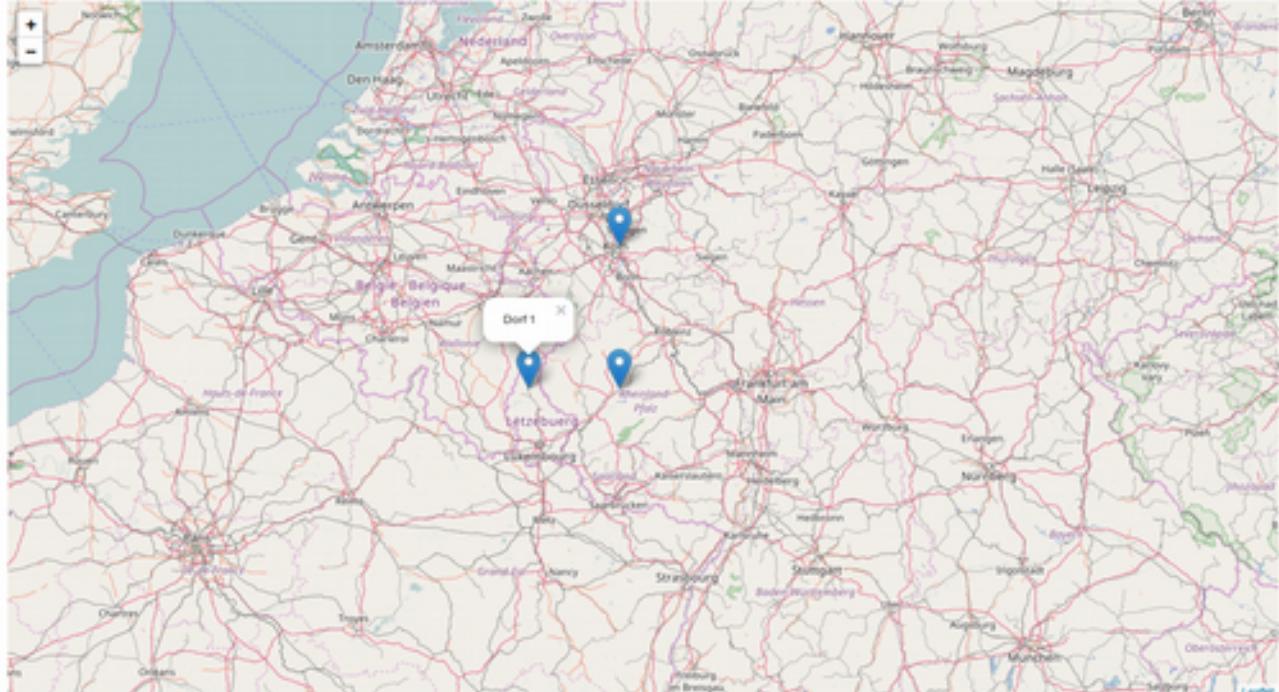
        }
    }).addTo(mymap);

```

</body>

</html>

Das nachfolgende Bild zeigt die drei erstellten Marker.



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. Leaflet bietet Ihnen aber einen anderen Funktion für diesen Zweck. Die Funktion heißt pointToLayer() und ein Beispiel mit dieser Methode finden Sie im nächsten Kapitel.

Punkt zu Ebene

Die Methode pointToLayer(), die wir uns in diesem Kapitel ansehen, arbeitet mit GeoJSON Objekten vom Typ Point. Diese Objekte werden anders verarbeitet als Polygone oder Linien. Im nachfolgenden 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 von der Eigenschaft name abhängig. Zusätzlich werden aber auch noch Optionen vom Namen abhängig gemacht. Nur der Marker von Dorf 1 kann auf der Karte verschoben werden.

Eine Funktion, die festlegt, wie GeoJSON neue Ebenen schichtet. Diese Funktion wird automatisch intern aufgerufen, wenn Daten hinzugefügt werden. Übergeben werden die Variablen GeoJSON Punkt-Funktion und seine LatLng. Die Voreinstellung ist, einen Standard-Marker zu erzeugen:

Todo bei einem LineString oder Polygon passiert nichts.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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"
};

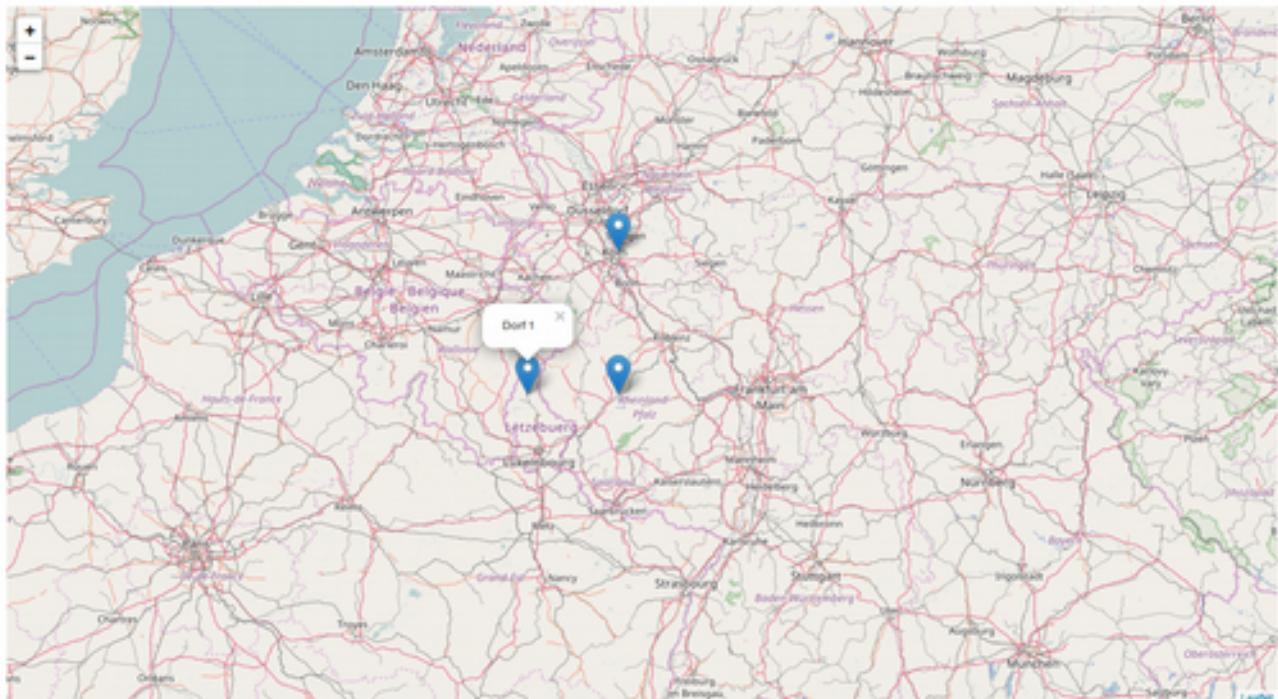
var options_notdraggable = {
draggable: false,
title: "Ein Ort in der Nähe von Gering"
};

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>

```

967.html

todo: <http://leafletjs.com/reference-1.1.0.html#geojson-pointtolayer>



956.png

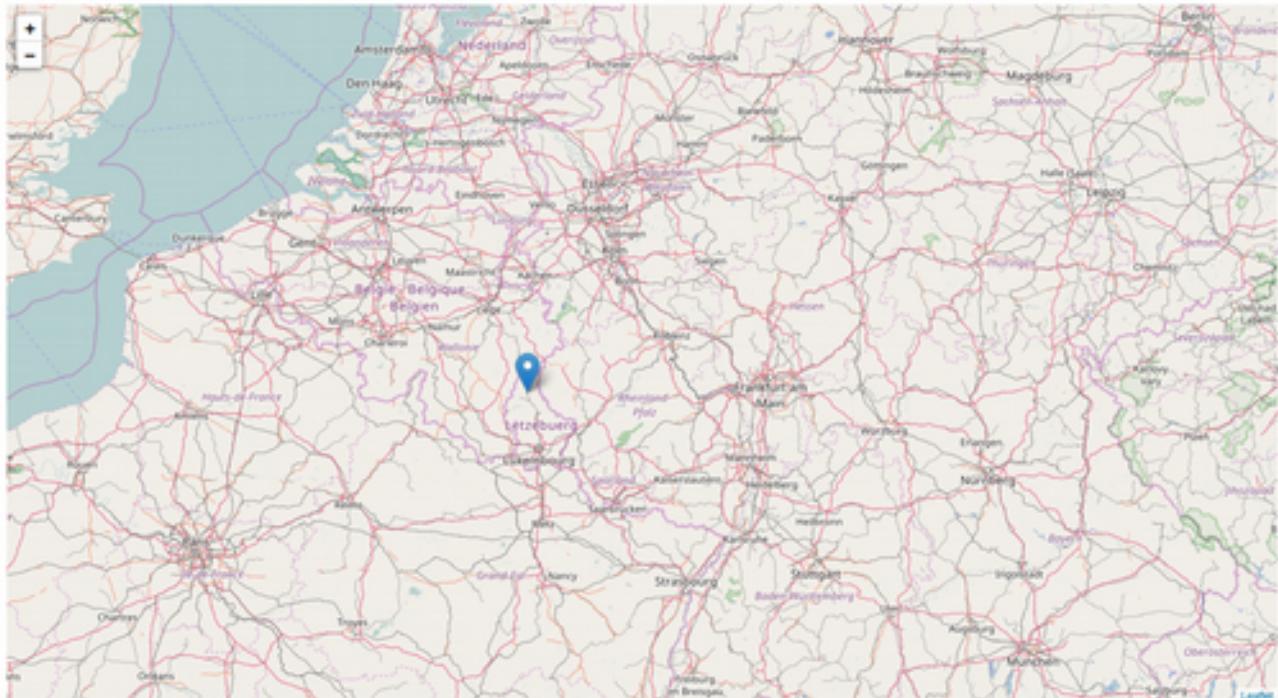
Auf den ersten Blick hat sich nichts zu dem Beispiel in vorherigen Kapitel geändert. Auf den zweiten Blick werden Sie feststellen, dass Sie nur den Marker von Dorf 1 auf der Karte verschieben können.

Filter

Todo Rückgabewert ist auschlaggebend.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM 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: 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>
```



955.png

Nur Dorf 1 wird angezeigt. Die beiden anderen Orte, die sich in denGeoJSON Daten befinden, werden heraus gefiltert.

todo vielleicht noch tipp gpsies [https://www.gpsies.com/map.do?
fileId=rvvclfejgrlqfrom](https://www.gpsies.com/map.do?fileId=rvvclfejgrlqfrom)

Heatmaps und Choroplethenkarten

In den ersten beiden Kapiteln haben wir uns angesehen, wie Sie Elemente entweder direkt oder als GeoJSON formatierte Daten zu Ihrer Karte hinzugefügt können. Jetzt geht es darum, dieses Wissen in die Tat umzusetzen. In diesem Kapitel werden wir nicht nur einfach auf der Karte malen. Wir werden mit der Karte Informationen weitergeben. Und geben Sie zu, insbesondere statistische Daten können auf Karten viel besser veranschaulicht werden als in einer trockenen Tabelle und nebenbei macht es sogar Spaß die Karten zu erkunden.

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

Heatmaps

Was ist eine Heatmap?

todo <https://de.wikipedia.org/wiki/Heatmap>

Heatmaps kennen wir im Deutschen auch unter dem Namen Wärmebild. Eine Heatmap ist im Grunde genommen ein Diagramm, mit dem Daten visualisiert werden. Meist liegt eine mathematische Funktion zugrunde. Diese Funktion bildet eine zweidimensionale Definitionsmenge – beispielsweise Punkte auf einer Landkarte – ab. Je nach Verteilung werden bestimmte Bereiche farblich hervorgehoben. 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.

Todo Dichte und Intensität, geht es hier nur um Dichte? Wir müssen Intensität über Umwege machen.

todo https://en.wikipedia.org/wiki/Multivariate_kernel_density_estimation

Meist werden bei geringer Intensität oder geringer Dichte kalte Farben verwendet und bei einem hohen Aufkommen wird der Bereich mit warmen Farben eingefärbt. Dies erklärt auch den Namen Heatmap – der englische Begriff für Hitze ist *heat*. Diese Art der Hervorhebung ist nicht zwingend.

Blau gilt als kalte Farbe, Rot, Orange und Gelb gelten als warme Farben.

Heatmaps in Leaflet – Dichte

Todo <http://leafletjs.com/plugins.html#heatmaps>

<https://github.com/Leaflet/Leaflet.heat>

todo leaflet heat ist alternative

todo was haben wir gemacht:

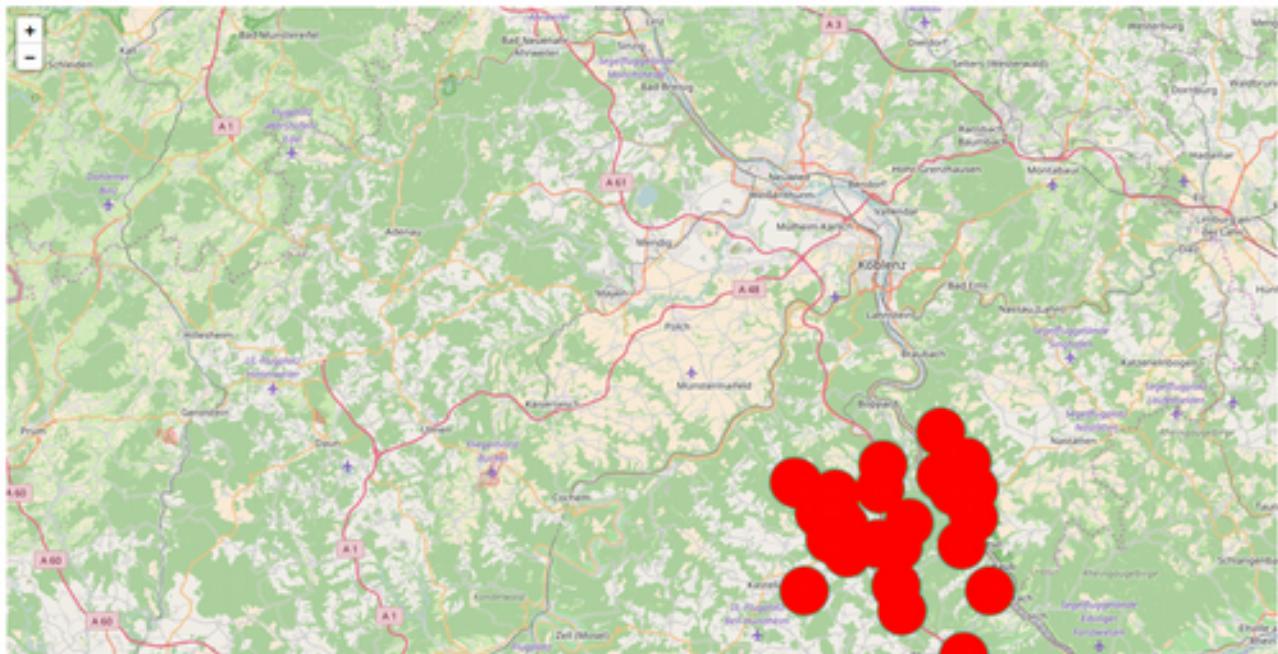
965.html

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

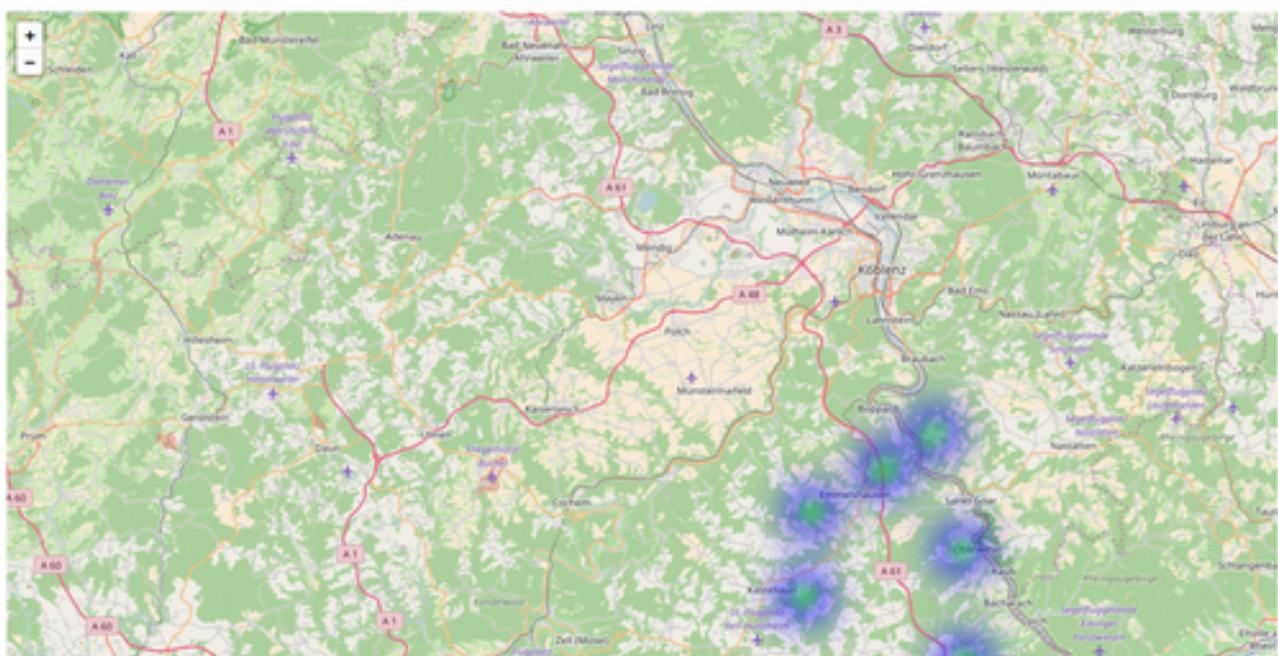
Stile-Optionen

Das [Plugin Leaflet.heat](#) erlaubt es Ihnen Parameter zu übergeben. Dabei haben Sie folgende Möglichkeiten:

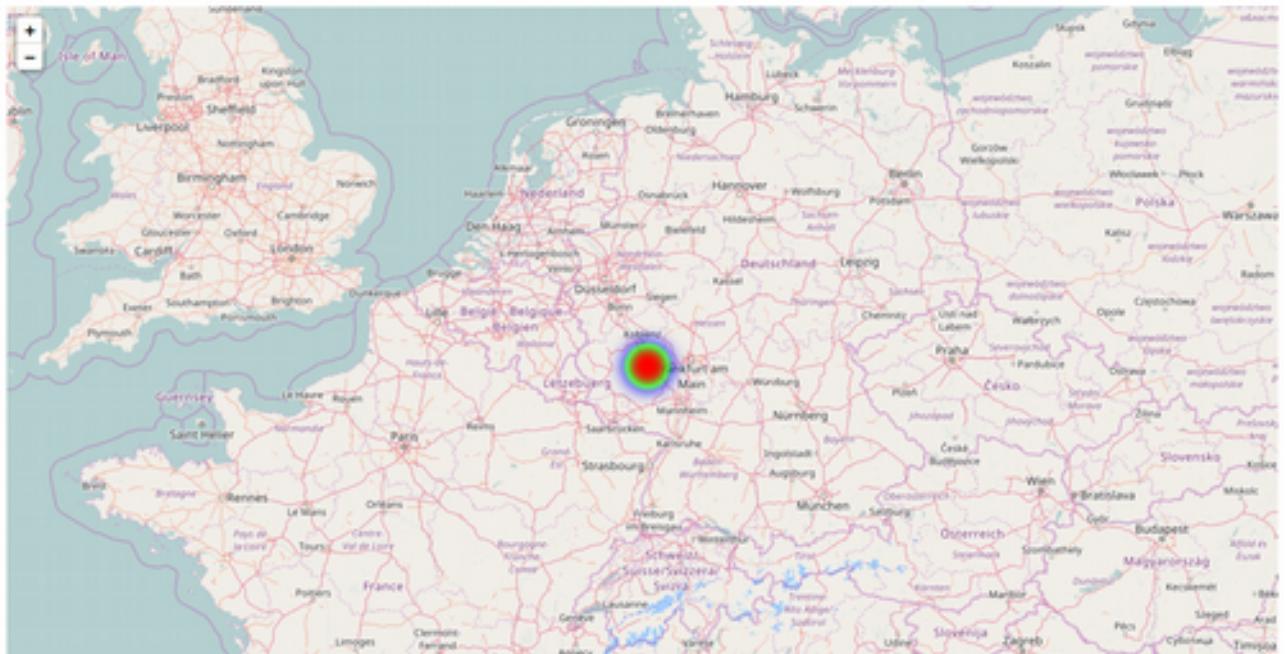
- `minOpacity`:
`minOpacity` gibt die minimale Dichte an, ab der die Anzeige beginnt.
- `maxZoom`:
`maxZoom` sollte auf die Zoomstufe gesetzt werden, bei der die Punkte die maximale Intensität erreichen. Als Standard wird der Wert, der für `maxZoom` in der Karte gesetzt ist, verwendet. Auf die Darstellung des Heatmap-Layers 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 Wert zu niedrig an, kann man die Daten vielleicht nicht mehr zusammenhängend im Überblick sehen.
- `max`:
Maximale Punktintensität – 1.0 ist der Standardwert. Diese Option arbeitet momentan nicht [korrekt](#).
- `radius`:
Dieser Wert 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 dem Wert `blur` können Sie die Schärfe beeinflussen. Der Wert bestimmt, wie viele Punkte zusammen gefasst werden. Ein niedriger `blur` Wert erzeugt einzelnen Punkte, wohingegen ein hoher Wert mehrere Punkte zusammenfasst. Standardmäßig ist der Wert mit 15 festgesetzt.
- `Gradient`:
Der Wert `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 angeben. Die Farbe die am Rand, also bei geringer Dichte angezeigt werden soll, sollten Sie mit dem niedrigen Wert angeben. Die Farbe, die im Zentrum zu sehen sein soll, geben Sie idealerweise beim Wert 1 an.



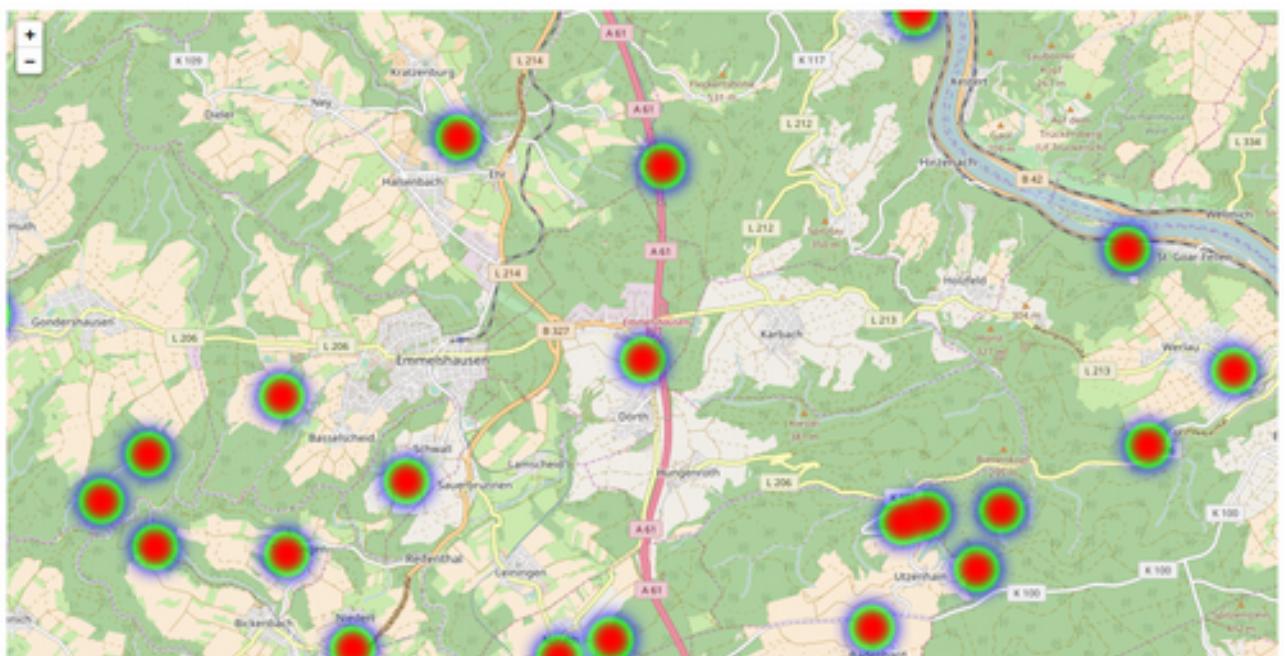
Blur 1 980a.png



blur 40 (ausgewaschen) 980b.png



maxzoom 6 980c.png



maxzoom 13 980d.png

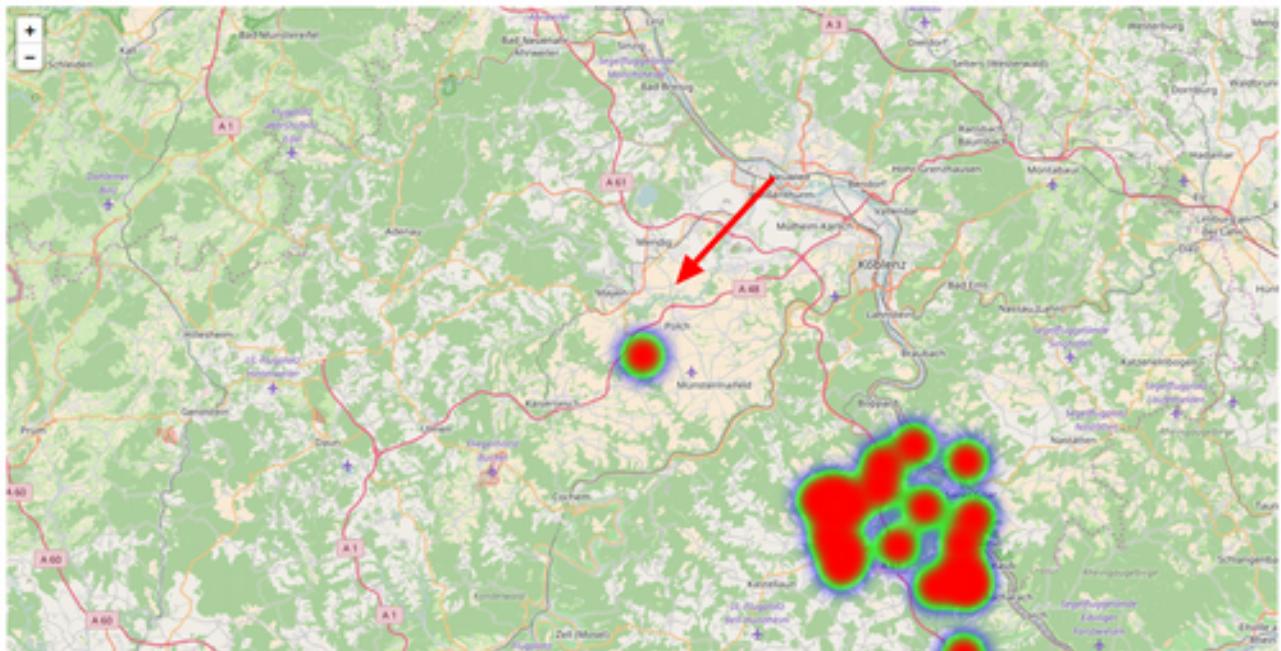
gradien {0.1: '#edf8fb', 0.2: '#ccece6', 0.3: '#99d8c9', 0.5: '#66c2a4', 0.7: '#2ca25f', 1: '#006d2c'} 980e

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 Stil Optionen bietet Ihnen Leaflet.heat vier Methoden:

- `setOptions(options)`: Sets new heatmap options and redraws it.
- `addLatLng(latlng)`: Adds a new point to the heatmap and redraws it.
- `setLatLngs(latlngs)`: Resets heatmap data and redraws it.
- `redraw()`: Redraws the heatmap. Diese Methode beim Ausführen der drei anderen Methoden automatisch ausgeführt, so das Sie dies nicht selbst veranlassen müssen.



979.png Todo Beispiel mit add Lat 964.html und 979.png

todo schönes Beispiel mit mous event:

<http://leaflet.github.io/Leaflet.heat/demo/draw.html>

<https://github.com/Leaflet/Leaflet.heat/blob/gh-pages/demo/draw.html>

todo Beispiel mit Methoden 963.html

Marker

962.html



980.png

Heatmaps mit Leaflet – heatmap.js – Intensität

<https://github.com/pa7/heatmap.js>

<https://www.patrick-wied.at/static/heatmapjs/plugin-leaflet-layer.html>

todo ich muss baselayer mit variabelen versehen weil ich nutez.

Todo doku <https://www.patrick-wied.at/static/heatmapjs/docs.html>

odo optionen

961.html

960.html → add data

Interaktive Heatmaps

959.html

Anmimierte Heatmaps

958.html

Choroplethenkarte

Im vorigen Kapitel haben wir mit Heatmaps die Dichte von Punkten oder eine bestimmte Eigenschaft farblich abgebildet. Eine Choroplethenkarte macht erst einmal nichts anderes. Sie visualisiert die Dichte oder die Intensität

bestimmter Objekte. Allerdings tut sie dies relativ zu einem Polygon. Eine bekannte Choroplethenkarte zeigt die Bevölkerungsdichte eines Landes.

Das Schöne ist, dass wir mit Leaflet keine zusätzlichen Plugins benötigen. Leaflet ist wie dafür gemacht, GeoJSON Daten als Choroplethenkarte anzuzeigen.

Was ist eine Choroplethenkarte?

Todo <https://de.wikipedia.org/wiki/Choroplethenkarte>

Choroplethenkarten werden in Deutschland auch Flächenkartogramme oder Flächenwertstufenkarten genannt. Diese Karten haben ein bestimmtes Thema. Verschiedene Gebiete auf der Karte werden im Verhältnis zur Verteilungsdichte des thematischen Objektes unterschiedlich gezeichnet. Choroplethenkarten können beispielsweise optisch darstellen, wie die Bevölkerungsentwicklung in einem bestimmten Gebiet ist. Oder, man kann Gebiete mit hohen Mietpreisen stärker hervorheben, als Gebiete mit niedrigen Mietpreisen.

Choroplethenkarten in Leaflet

<http://opendatalab.de/projects/geojson-utilities/> todo hier habe ich daten heruntergeladen und es ist ein gutes Beispiel für eine Kartenanwendung.

GeoJson

Die GeoJSON-Datei, die wir 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 JavaSript Datei ablegen, können Sie diese gleichzeitig als Variabel deklarieren.

Todo Beispielcode hier.

Das Beispiel zeigt nur einen Codeausschnitt. Die Vollständige Datei wäre todo Zielen lang und würde ein eigenen Buch füllen.

Einbinden können Sie diese in Ihre HTML-Datei mit der Zeile.

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

Farben

Stile

Normalisierte Choroplethenkarten

Todo überall anfang und schluss

todo Fläche selbst ausrechnen:

<https://stackoverflow.com/questions/18484619/checking-size-of-polygon-drawn-with-leaflet-l-draw-polyline>

Custom Markers

Sie wissen nun wie Sie Daten auf Ihrer Karte in Form von Elementen anzeigen können und wie Sie mit den Daten ein Thema visualisieren können. Im nächsten Schritt geht es darum, der Karte eine individuelle Wirkung zu geben. Wir sehen uns an, wie Sie Marker beliebig gestalten können. Außerdem sehen wir uns ein Plugin an, dass Ihnen diese Arbeit erleichtert.

Einen individuellen Marker erstellen

Wenn Leaflet einen Marker auf einer Karte anzeigt, werden gleich zwei Bilddateien angezeigt: Das eigentliche Bild und ein Schatten. Mit einem passenden Schatten fallend die Marker eher ins Auge. Der Schatten verleiht dem Marker ein Tiefe. So heben Sie sich besser von der Karte ab.

Wir haben Leaflet bisher in unseren Beispieldateien als CND genutzt. Ich hatte Ihnen aber in Kapitel todo auch erklärt, wie Sie Leaflet downloaden können. Wenn Sie Leaflet auf Ihren Rechner herunterladen sehen Sie – sofort in der ersten Ebene – einen Unterordner, der `images` heißt. 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 bekannt vor.



978.png

Todo <https://www.gimp.org/>

todo link zu nächstem Kapitel mit Plugin

Wie Sie ein eigenes Image erstellen, gehört nicht zum Thema dieses Buches. Hier möchte ich Ihnen nur ein paar Punkte aufzählen, die Sie beachten sollten. Wenn Sie das

todo

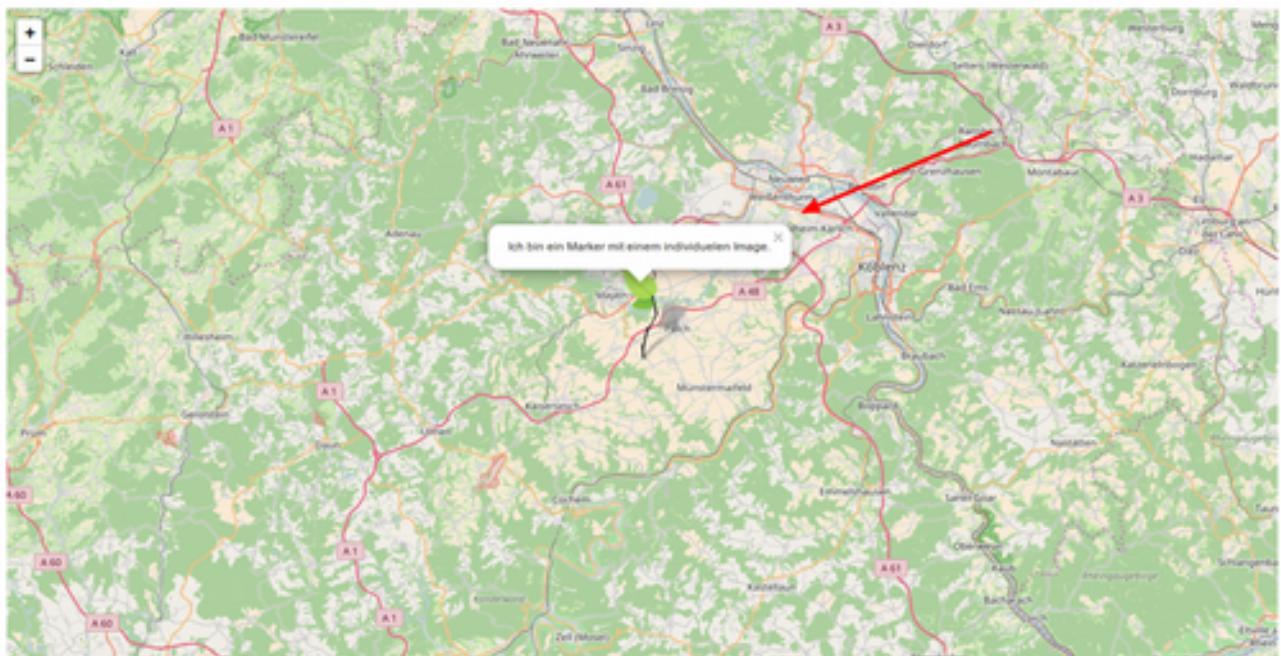
Wenn zwei Bilder haben, also ein Bild, das Ihren Marker selbst darstellt und eines, das den Schatten zeigt, dann können wir dieses in Ihre Karte einbinden. Alternativ können Sie zum Ausprobieren auch die Dateien im Tutorial verwenden

<http://leafletjs.com/examples/custom-icons/leaf-green.png>

<http://leafletjs.com/examples/custom-icons/leaf-red.png>

<http://leafletjs.com/examples/custom-icons/leaf-orange.png>

<http://leafletjs.com/examples/custom-icons/leaf-shadow.png>



977.png

955.html

Eigenschaften eines individuellen Marker

Einen eigenes Marker Bild erstellen Sie in Leaflet mithilfe der Klasse [L.icon](#). Diese Klasse bietet Ihnen zehn Optionen.

- `IconUrl`:
Die `IconUrl` müssen Sie auf alle Fälle angeben. Die URL zum Icon-Bild (absolut oder relativ zu Ihrem Skriptpfad).
- `IconRetinaUrl`:
Die URL zu einer Retina Größe Version des Symbolbildes (absolut oder relativ zu deinem Skriptpfad). Verwendet für Retina Bildschirmgeräte.
Todo und was ist Retina
- `iconSize`:
Größe des Symbolbildes in Pixeln.
- `IconAnchor`:
Die Koordinaten der "Spitze" des Symbols (relativ zu der oberen linken Ecke). Das Symbol wird so ausgerichtet, dass dieser Punkt an der geographischen Lage des Markers liegt. Zentriert, wenn die Größe angegeben ist, kann auch in CSS mit negativen Rändern (`margin`) gesetzt werden.
- `PopupAnchor`:
Die Koordinaten des Punktes, von dem Popups "öffnen", relativ zum Symbol Anker.
- `ShadowUrl`:
Die URL zum Icon Schattenbild. Wenn nicht angegeben, wird kein Schattenbild erstellt.
- `ShadowRetinaUrl`:
- `shadowSize`:
Größe des Schattenbildes in Pixeln.
- `ShadowAnchor`:
Die Koordinaten der "Spitze" des Schattens (relativ zu der oberen linken Ecke) (das gleiche wie `iconAnchor` wenn nicht angegeben).
- `className`:
Ein benutzerdefinierter Klassenname, der beiden Symbol- und Schattenbildern zugewiesen werden soll. Falls diese Option nicht genutzt wird, wird kein Klassenname hinzugefügt.

Die Klasse `L.Icon` erweitern

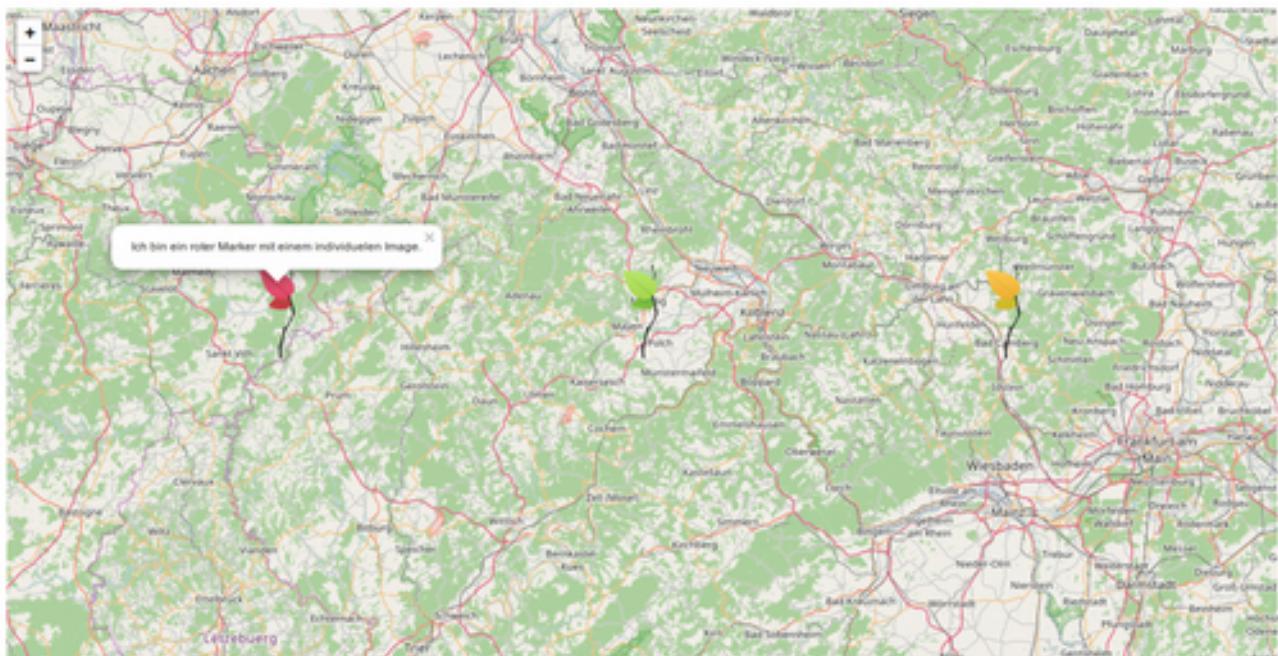
todo <http://leafletjs.com/examples/custom-icons/>

So, nun haben Sie einen benutzerdefinierten Marker erstellt und möchten weitere Marker kreieren. Schön, dass Leaflet objektorientiert programmiert ist.

So können Sie sich das Erstellen von neuen Markern sehr einfach machen. Erweitern Sie einfach die Klasse L.Icon und geben alle gemeinsamen Eigenschaften nur einmal an. Nur die besonderen Eigenschaften des Markers müssen Sie separat angeben.

You may have noticed that we used the new keyword for creating LeafIcon instances. So why do all Leaflet classes get created without it? The answer is simple: the real Leaflet classes are named with a capital letter (e.g. L.Icon), and they also need to be created with new, but there are also shortcuts with lowercase names (L.icon), created for convenience like this:

```
L.icon = function (options) {  
    return new L.Icon(options);  
}; todo mit link zu github?
```



976.png

954.html

Ein Marker Plugin

Sie wissen nun, wie Sie einen Marker mit einem standardisierten Aussehen einfügen und können einen Marker mit einem eigenen Image belegen. Nun haben Sie aber kein eigenes Image und haben auch nicht viel Erfahrung mit einem Grafikprogramm um ein professionell aussehendes individuelles Image zu erstellen. Trotzdem möchten Sie Ihrer Karte ein besonderes Aussehen geben. In diesem Fall bietet Ihnen diese Kapitel eine Lösung. Ich stelle Ihnen

zwei Plugins vor, die Sie bei der Erstellung von individuellen Markern unterstützen.

Beautiful markers

<https://github.com/marslan390/BeautifyMarker>

975.png

974.png

953.html

todo how to make bigger imageSize und vielleicht noch einmal zeigen wie man optionen einbindet. <https://github.com/marslan390/BeautifyMarker/issues/10>

todo other frameworks

<https://github.com/jseppi/Leaflet.MakiMarkers> (hier gibt es auch size)

<https://github.com/lvoogdt/Leaflet.awesome-markers>

Cluster

Je nachdem welche Informationen Sie mit Ihrer Karte weitergeben möchten, kann es vorkommen, dass Sie sehr viele Marker benötigen. Wenn Sie mit vielen Markern 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 doch, 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 der Kartenazeige.

Die erste Cluster Karte

Todo <https://github.com/Leaflet/Leaflet.markercluster>

952.html

973.png

Methoden und Events

todo

Standardmethoden

todo

Andere Methoden

todo

Marker animieren

Hüpfende Marker

<https://github.com/maximeh/leaflet.bouncemarker>

951.html

Animierte Marker

<https://github.com/openplans/Leaflet.AnimatedMarker>

950.html

Weitere Beispiel

todo mit stop start 949.html

(<https://github.com/openplans/Leaflet.AnimatedMarker/issues/50>)

todo 948 bounce out Schlußfunktion

Mit Markern Daten visualisieren

Todo <https://github.com/humangeo/leaflet-dvf> und

<https://github.com/humangeo/leaflet-dvf/wiki/1.0-Compatibility>

Das Plugin Data Visualization Framework

947.html

todo <https://github.com/humangeo/leaflet-dvf/wiki/6.-Markers>

Diagrammen als Marker

<https://github.com/humangeo/leaflet-dvf/wiki/6.-Markers> welche Marker es gibt mit bild.

946.html

972.png

todo Schluss

So, nun wissen Sie todo und können Daten visualisieren. Nun kann es sein, dass Sie selbst keine Datenbestände zum Anzeigen haben. Vielleicht bietet das ESRI Inc. (Environmental Systems Research Institute) Ihnen ja Daten, die zum Thema Ihrer Website passen? Im nächsten Kapitel erfahren Sie, wie Sie Daten dieses Institutes verwenden können.

ESRI

<https://de.wikipedia.org/wiki/ESRI>

Wenn Sie mit Karten arbeiten Daten eines Geoinformationssystems anzeigen möchten, werden Sie früher oder später über den Begriff Shapefile stolpern.

Todo <https://de.wikipedia.org/wiki/Shapefile>

gdb Geodatabase

<https://de.wikipedia.org/wiki/ArcGIS>

ESRI Basis Layer

ESRI bietet

<http://www.esri.com/data/basemaps>

todo <https://github.com/Esri/esri-leaflet>

945.html (todo bild mit und ohne label)

todo was muss ich tun

todo hier alle basemap layer und label layer: <https://esri.github.io/esri-leaflet/api-reference/layers/basemap-layer.html>

Switch Basemaps

<https://esri.github.io/esri-leaflet/examples/switching-basemaps.html>

944.html

Shapefiles

Das Dateiformat Shapefile (oft Shapedaten oder Shape genannt) ist ein ursprünglich für die Software ArcView der Firma ESRI entwickeltes Format für Geodaten.

Ein Shapefile ist keine einzelne Datei, es besteht aus mindestens drei Dateien:

- .shp dient zur Speicherung der Geometriedaten
- .dbf Sachdaten im dBASE-Format
- .shx dient als Index der Geometrie zur Verknüpfung der Sachdaten (auch Attributdaten genannt)

todo

todo Shapefiles findet man, wenn man nach open data und einer Stadt sucht.
Oder <http://wiki.openstreetmap.org/wiki/Shapefiles>

Mein Beispiel <https://www.arcgis.com/home/item.html?id=ae25571c60d94ce5b7fcbf74e27c00e0>
970.png

todo <https://wambachers-osm.website/boundaries/idx42o.jsp> todo bild
971.png

Datei mit Namen exported_boundaries_Germany.shp.zip ist im
Downloadverzeichnis.

Das Archiv enthält die Dateien

-

todo leaflet shape file plugin :
<https://github.com/calvinmetcalf/leaflet.shapefile> 2 dateien

todo testen 969.png <http://leaflet.calvinmetcalf.com/#6/51.358/11.964>

943.html

967.png

mit styles

942.html

968.png

todo vielleicht shape files ansehen

todo optionen des plugins leaflet shape ansehen.

Todo wie popup

941.html 966.png

tip 940 falls du features nicht kennst. 965.png

ESRI Services

Sie wissen nun wie Sie eine ESRI Basiskarte laden, wie Sie mit dem Plugin ESRI-leaflet umgehen und wie Sie Dateien im Format shapefile nutzen können.

ESRI bietet aber noch mehr. Sie können Webservices nutzen. Das ESRI Plugin unterstützt Sie auch bei der Verbindung mit einem Geografischen Webservice.

Ganz nebenbei kann man noch weitere Basemaps nutzen.

- L.esri.BasemapLayer
- L.esri.DynamicMapLayer
- L.esri.ImageMapLayer
- L.esri.RasterLayer
- L.esri.TiledMapLayer
- L.esri.FeatureLayer
- L.esri.Cluster.FeatureLayer
- L.esri.Heat.FeatureLayer

<http://esri.github.io/esri-leaflet/api-reference/>

L.esri.DynamicMapLayer

Todo? <https://github.com/Esri/esri-leaflet>

<http://esri.github.io/esri-leaflet/api-reference/layers/dynamic-map-layer.html>

<https://geoportal.stadt-koeln.de/arcgis/rest/services/Behindertenparkplaetze/MapServer/0>

<https://geoportal.stadt-koeln.de/arcgis/rest/services/>

939.html

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“, die Sie zum Platzieren von Markern auf einer Karte oder zum Positionieren der Karte verwenden können (<https://developers.google.com/maps/documentation/geocoding/intro?hl=de>).

<https://de.wikipedia.org/wiki/Georeferenzierung>

todo es bit auch

<http://wiki.openstreetmap.org/wiki/Nominatim> → nicht so sicher, man kann aber daten eintragen!

Und <https://developers.google.com/maps/documentation/geocoding/start>

<https://github.com/Esri/esri-leaflet-geocoder>

Nach Eingabe einer Adresse den passenden Ort auf der Karte finden
Umgekehrtes Geocoding bezeichnet die Umwandlung geografischer Koordinaten in lesbare Adressen.

938.html

Achtung: adresse nur im Bereich wird gefunden

Mithilfe eines Parameters in der URL den passenden Ort finden

937.html

<http://esri.github.io/esri-leaflet/api-reference/tasks/geocode.html>

<file:///media/astrid/AstridsAktuelleDatenPlatte/Daten/arbeit/Schreiben/bookbon/softwaretesting/BuchLeaflet/code/5/URLgeocode.html?a=56751%20Gering>

Nach Klick auf die Karte die passende Adresse ausgeben

<http://esri.github.io/esri-leaflet/api-reference/tasks/reverse-geocode.html>

query layers

Todo Punkte im ausgewählten Bereich werden gefunden. Deshalb Bereich klein halten, falls es viele Punkte gibt.

Wir haben im letzten Kapitel Daten über einen Webservice geladen. Dabei haben immer alle Daten auf der Karte angezeigt. Das kann sehr unübersichtlich werden. Meist bietet ein Webservice ja große Datenmengen an. Sie möchten sicherlich nur die für Sie und Ihre Anwender relevanten Daten auf Ihrer Karte anzeigen. Eine Lösung für die Aufgabe erarbeiten wir in diesem Kapitel.

Todo unterschied feature server map server:

<https://gis.stackexchange.com/questions/126373/difference-between-feature-server-and-map-server>

doku feature layer: <http://esri.github.io/esri-leaflet/api-reference/layers/feature-layer.html>

todo noch gucken : <https://gis-iq.esri.de/openstreetmap-daten-aus-deutschland-in-arcgis-online/>

todo was gibt es in GIS: <https://gis-iq.esri.de/content-auf-der-arcgis-plattform-viel-mehr-als-nur-basemaps/>

Attribut

Konfigurieren einer Abfrage zum Filtern von Funktionen auf einem [L.esri.FeatureLayer](#).

<http://esri.github.io/esri-leaflet/examples/feature-layer-popups.html>

sicherheitseinrichtungen: <http://www.arcgis.com/home/item.html?id=59711f4ee839438299c90164115f129b>

935.html

abstand

Ausführen von erweiterten Abfragen auf einer Feature-Ebene. Klicken Sie auf die Karte, um die Funktionen innerhalb von 500 Metern abzufragen. Weitere Informationen über Feature Layer finden Sie in der Dokumentation:

<http://esri.github.io/esri-leaflet/api-reference/layers/feature-layer.html>.
<http://esri.github.io/esri-leaflet/examples/querying-feature-layers-2.html>)

934.html

Leaflet benutzerfreundlich als Joomla! Modul

Todo

Leaflet versucht nicht, jedem alles recht zu machen. Stattdessen konzentriert sich das JavaScript Framework darauf, das was Sie zum Erstellung einer Karte benötigen, perfekt anzubieten. Ganz unabhängig davon, wie Sie die Karte präsentieren. Das hat Vorteile. Das Framework bietet eine flexible Schnittstelle und Sie haben viele Möglichkeiten. Nachteilig ist, dass Sie sich selbst um alles darum herum kümmern müssen. Heute werden Karten meist online angezeigt. Ein beliebtes Content Management System, mit dem viele Websites gepflegt werden, ist [Joomla!](#) Warum nehmen Sie nicht einfach Joomla! als Drumherum.

Beispielhaft zeige ich Ihnen hier, wie Sie Ihre mit Leaflet erstellte Karte mithilfe eines Moduls in Joomla! benutzerfreundlich einsetzen können und wie Sie es anderen bedienerfreundlich ermöglichen, benutzerdefinierte Marker in die Karte einzufügen.

Ein Modul in Joomla!

Module sind einfache und flexible Joomla! Erweiterungen. Sie werden für kleine Bereiche der Seite verwendet, die im Allgemeinen weniger komplex sind und über verschiedene Menüpunkte hinweg gesehen werden können. Module verfügen in der Regel über keine eigene Programmlogik. Vielmehr greifen Sie auf die Algorithmen von Controllern zu. Die Ergebnisse dieser Controller

präsentieren Sie dann – im Idealfall schön gestaltet – auf Ihrer Website. Aber Sie merken es schön: Ein Joomla!-Modul kann man nicht ganz klar einem Teil des [Entwurfsmusters Model-View-Controller](#) zuordnen.

Sie müssen nicht immer das Rad neu erfinden. Ich habe die Erfahrung gemacht, dass viele meiner Fragestellungen schon einmal von einem anderen behandelt wurden und ich aus dessen Lösung viel lernen kann. Um das Programmieren einer Joomla! Erweiterung zu lernen ist der beste Ort wohl Joomla! selbst. Sehen Sie selbst nach: Sie finden viele [Beispiele für Module](#) im Kernbereich von Joomla!

Ich erkläre Ihnen hier in diesem Kapitel, wie Sie ein einfaches Modul zur Anzeige einer Karte mit Leaflet erstellen können. In diesem Kapitel lernen Sie die grundlegende Dateistruktur eines Moduls kennen. Diese Grundstruktur können Sie später erweitern, um aufwendigere Module zu erstellen.

Auch in der Dokumentation von Joomla! finden Sie ein [Tutorial](#), das die Erstellung eines Modules erklärt. Todo hier lernst du auch Skript und ...

Grundlegende Dateien

Es gibt vier Dateien, die ich zu einem Grundgerüst eines Joomla! Modules zähle. Das sind die Dateien

- `mod_MODULNAME.php`:
Diese Datei heißt in meinem Beispiel `mod_jdosm.php`. Diese Datei ist der Haupteingangspunkt in das Modul. Mit ihr werden alle notwendigen Initialisierungs Routinen durchführen. Die Helper-Routinen werden aufgerufen, um alle notwendigen Daten zu sammeln und die Vorlage aufzunehmen, die die Modulausgabe anzeigt.
- `helper.php`:
Diese Datei enthält die Helper-Klasse, die verwendet wird, um die eigentliche Arbeit beim Abrufen der Informationen, die im Modul angezeigt werden sollen (in der Regel aus der Datenbank oder einer anderen Quelle).
- `mod_MODULNAME.xml`:
Diese Datei enthält Informationen zum Modul. Sie definiert die Dateien, die von während der Installation von Joomla! Bearbeitet werden müssen. Außerdem legt sie die Konfigurationsparameter für das Modul fest.

- tmpl/default.php:

In der Datei default.php ist die Vorlage oder das Template für das Modul. Diese Datei nimmt die von mod_MODULNAME.php gesammelten Daten auf und erzeugt das HTML, das auf der Seite angezeigt werden soll.

Todo es gibt Backend und Frontend Module wir behandeln nur ein Frondendmodul.

Creating mod_MODULNAME.php

Die Datei mod_MODULNAME.php hat drei wichtige Aufgaben:

- Zuerst bindet sie die Datei helper.php ein. Diese Datei enthält die Klasse, die verwendet werden soll, um die notwendigen Daten zu sammeln.
- Dann ruft sie die entsprechende Methode der Datei helper.php auf, um die passenden Daten zur Anzeige zurückzuerhalten.
- Als letztes bindet Sie Vorlage – also die Datei – ein, um die Ausgabe zu erzeugen.

Die Helper Klasse definieren wir später in der Datei helper.php. Diese Datei ist mit einer require_once-Anweisung enthalten:

Require_once wird verwendet, weil unsere Helperfunktionen innerhalb einer Klasse definiert sind und wir nur die Klasse einmal definieren wollen.

Unsere Helper-Klasse ist noch nicht definiert, aber wenn sie fertig ist, wird es eine Methode enthalten die getHello() heißt. In unserm kleinen Beispiel bringt es keine Vorteile die Ausgabe Hallo, I bin ein Leaflet Karte! in einer separaten Hilfsdatei zu erzeugen. Der einfache Satz könnte einfach in die Datei default.php eingetragen werden. Trotzdem bliebe alles übersichtlich. Ich verwende hier aber trotzdem die Hilfsdatei, weil ich Ihnen die Grundtechnik demonstrieren möchte.

Todo \$ Hello = modHelloWorldHelper :: getHello (\$ params);

Nun bleibt noch eine Zeile, die ich Ihnen nicht erklärt haben – nämlich die allererste. Diese Zeile überprüft, um sicherzustellen, dass diese Datei aus dem Joomla! Anwendung. Dies ist notwendig, um eine variable Injektion und andere potenzielle Sicherheitsbedenken zu verhindern.

Die komplette Datei "mod_jdosm.php" sieht nun wie folgt aus:

```
mod_jdosm.php
<?php
defined('_JEXEC') or die;
require_once __DIR__ . '/helper.php';
$moduleclass_sfx = htmlspecialchars($params-
>get('moduleclass_sfx'));
require JModuleHelper::getLayoutPath('mod_jdosm', $params-
>get('layout', 'default'));
```

Creating helper.php

Die helper.php-Datei enthält die Helper-Klasse, die verwendet wird, um die Daten, die in der Modulausgabe angezeigt werden sollen, abzurufen. Wie bereits erwähnt, hat unsere Helferklasse eine Methode: getHallo(). Diese Methode gibt die Nachricht "Hallo, Welt" zurück.

Here is the code for the helper.php file:

```
<?php
defined('_JEXEC') or die('');
class ModJdosmHelper
{
public static function getHallo()
{
    return "Hallo, I bin ein Leaflet Karte!";
}
```

todo Lizenz erklären

Es gibt keine Regel, dass wir unsere Helfer-Klasse benennen müssen, wie wir haben, aber es ist hilfreich, dies zu tun, damit es leicht identifizierbar und lokalisierbar ist.

Creating tmpl/default.php

Die default.php Datei ist die Vorlage, die die Modulausgabe anzeigt.

Der Code für die default.php-Datei lautet wie folgt:

```
<?php  
defined('_JEXEC') or die;  
echo ModJdosmHelper::getHello();
```

todo erklärung zum Aufruf helperkasse von oben nach hier nehmen.

Ein wichtiger Punkt ist, dass die Vorlagendatei denselben Gültigkeitsbereich hat wie die Datei "mod_helloworld.php". Was bedeutet dies, dass die Variable \$ hello in der Datei "mod_helloworld.php" definiert und dann in der Vorlagendatei ohne zusätzliche Deklarationen oder Funktionsaufrufe verwendet werden kann.

Creating mod_helloworld.xml

Mit der Datei mod_helloworld.xml können Sie festlegen, welche Dateien das Installationsprogramm kopieren muss und wird vom Modulmanager verwendet, um festzustellen, welche Parameter für die Konfiguration des Moduls verwendet werden. Weitere Informationen zum Modul finden Sie auch in dieser Datei.

The code for mod_helloworld.xml is as follows:

```
<?xml version="1.0" encoding="utf-8"?>  
<extension type="module" version="3.6" client="site"  
method="upgrade">  
  <name>mod_jdosm</name>  
  <author>Astrid Günther</author>  
  <creationDate>2017-06-21</creationDate>  
  <copyright>(C) 2005 - 2017 Open Source Matters. All rights  
reserved.</copyright>  
  <license>GNU General Public License version 2 or later;</license>  
  <authorEmail>info@astrid-guenther.de</authorEmail>  
  <authorUrl>www.astrid-guenther.de</authorUrl>  
  <version>1.0.0</version>
```

```
<description>MOD_JDOSM_XML_DESCRIPTION</description>
<files>
<folder>language</folder>
<folder>tmpl</folder>
<folder>models</folder>
<filename module="mod_jdosm">mod_jdosm.php</filename>
<filename>helper.php</filename>
</files>
<config>
</config>
</extension>
```

Da unser Modul keine Formularfelder verwendet, ist der Konfigurationsabschnitt leer.

Modulentwicklung für Joomla! Ist ein ziemlich einfacher, einfacher Prozess. Mit den in diesem Tutorial beschriebenen Techniken kann eine unendliche Vielfalt von Modulen mit wenig Aufwand entwickelt werden.

Im modul die karte anstelle von ahillo anzeigen

Xml

Parameter aufnehmen

```
...
<filename module="mod_jdosm">mod_jdosm.php</filename>
<filename>helper.php</filename>
</files>
<config>
<fields name="params">
<fieldset
name="mapConfig"
label="MOD_JDOSM_H_MAP"
>
<field
type="integer"
name="height"
label="MOD_JDOSM_HEIGHT"
```

```
description="MOD_JDOSM_HEIGHT_DESC"
default="400"
first="10"
last="1000"
step="10"
/>
<field
type="list"
name="zoom"
default="7"
label="MOD_JDOSM_ZOOM"
description="MOD_JDOSM_ZOOM_DESC"
>
<option value="1">1</option>
<option value="2">2</option>
<option value="3">3</option>
<option value="4">4</option>
<option value="5">5</option>
<option value="6">6</option>
<option value="7">7</option>
<option value="8">8</option>
<option value="9">9</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
</field>
<field
type="list"
```

```
name="baselayer"
default="mapbox"
label="MOD_JDOSM_BASELAYER"
description="MOD_JDOSM_BASELAYER_DESC"
>
<option value="mapnik">OpenStreetMap</option>
<option value="stamenwater">Stamen Watercolor</option>
</field>
<field
type="spacer"
name="spacerPos0"
hr="true"
/>
<field
type="spacer"
name="spacerPos"
label="MOD_JDOSM_POSITION"
class="text"
/>
<field
type="text"
name="lon"
label="MOD_JDOSM_LON"
description="MOD_JDOSM_LON_DESC"
default="7.43"
/>
<field
type="text"
name="lat"
label="MOD_JDOSM_LAT"
description="MOD_JDOSM_LAT_DESC"
default="46.93"
/>
</fieldset>
</fields>
</config>
```

```
</extension>
```

tmpl/default.php

```
<?php
defined('_JEXEC') or die;
echo ModJdosmHelper::getHello();
$document = JFactory::getDocument();
$document-
>addStyleSheet('https://unpkg.com/leaflet@1.2.0/dist/leaflet.css')
;
$document-
>addScript('https://unpkg.com/leaflet@1.2.0/dist/leaflet.js');
?>
<div style="
    width: auto;
    height:<?php echo $params->get('height', '400'); ?>px;" 
    id="map<?php echo $module->id; ?>">
</div>
<?php
$script = array();
$script[] = "<script>";

$script[] = ModJdosmHelper::getMap($module, $params);
$script[] = ModJdosmHelper::getTileLayer($module, $params);
$script[] = "mymap" . $module->id . ".setView([" .
    $params->get('lat', '46.93') . ", " .
    $params->get('lon', '7.43') . "], " .
    $params->get('zoom', 'false') . ")";
$script[] = "</script>";
echo implode("\n", $script);
```

helper

```
<?php
defined('_JEXEC') or die('');
class ModJdosmHelper
{
    public static function getHello()
    {
        return "Hallo, Ich bin eine Leaflet Karte!";
    }
    public static function getMap($module, $params)
    {
        $script = "";
        $script .= "var mymap" . $module->id . " = L.map('map" . $module-
>id . "');";
        return $script;
    }
    public static function getTileLayer($module, $params)
    {
        $baselayerSettings = "";
        $baselayerURL = "";
        if ($params->get('baselayer', 'mapbox') == 'mapnik')
        {
            $baselayerURL = 'https://{s}.tile.openstreetmap.org/{z}/{x}/
{y}.png';
            $baselayerSettings = "maxZoom: 19 ";
        }
        elseif ($params->get('baselayer', 'mapbox') == 'stamenwater')
        {
            $baselayerURL = 'https://stamen-tiles-
{s}.a.ssl.fastly.net/watercolor/{z}/{x}/{y}.png';
            $baselayerSettings = "subdomains: 'abcd', minZoom: 1, maxZoom: 16
";
        }
    }
}
```

```

else
{
$baselayerURL = 'https://{s}.tile.openstreetmap.org/{z}/{x}/
{y}.png';
$baselayerSettings = "maxZoom: 19 ";
}
$script = "";
$script .= " L.tileLayer('' . $baselayerURL . '', {";
$script .= $baselayerSettings;
$script .= " }) . addTo(mymap" . $module->id . ");";
return $script;
}
}

```

CUSOM marker

Xml Datei

hauptdatei

```

...
<field
type="list"
name="showpin"
default="0"
label="MOD_JDOSM_CONFIG_CONTROL_SHOWPIN_LABEL"
description="MOD_JDOSM_CONFIG_CONTROL_SHOWPIN_DESC"
>
<option value="1">JYES</option>
<option value="0">JNO</option>
</field>
<field
name="specialpins"
type="subform"
formsource="modules/mod_jdosm/models/forms/custombuttons.xml"
multiple="true"

```

```

label="MOD_JDOSM_CONFIG_CONTROL_SUBFORM_LABEL"
min= "1"
max= "20"
description="MOD_JDOSM_CONFIG_CONTROL_SUBFORM_DESC"
layout="joomla.form.field.subform.repeatable"
showon="showpin:1"
/>
</fieldset>
</fields>
</config>
</extension>

```

Ergänzung models/forms/custombutton.xms

```

<?xml version="1.0" encoding="UTF-8"?>
<form>
<field
type="text"
name="lonpin"
label="MOD_JDOSM_LON"
description="MOD_JDOSM_LON_DESC"
default="7.43"
/>
<field
type="text"
name="latpin"
label="MOD_JDOSM_LAT"
description="MOD_JDOSM_LAT_DESC"
default="46.93"
/>
<field
type="spacer"
name="spacerPosMedia0"

```

```
hr="true"
/>
<field
type="radio"
name="pin"
default="1"
label="MOD_JDOSM_PIN"
description="MOD_JDOSM_PIN_DESC"
>
<option value="1">&lt;img
src="https://unpkg.com/leaflet@1.2.0/dist/images/marker-
icon.png" /&gt;</option>
<option value="2">MOD_JDOSM_OWNPIN</option>
</field>
<field
type="media"
name="customPinPath"
label="MOD_JDOSM_PINPATH"
description="MOD_JDOSM_PINPATH_DESC"
default="http://leafletjs.com/examples/custom-icons/leaf-
green.png"
showon="pin:2"
/>
<field
type="text"
name="customPinSize"
label="MOD_JDOSM_PINSIZE"
description="MOD_JDOSM_PINSIZE_DESC"
default="38, 95"
showon="pin:2"
/>
<field
type="media"
name="customPinShadowPath"
label="MOD_JDOSM_PINSHADOWPATH"
description="MOD_JDOSM_PINSHADOWPATH_DESC"
```

```
default="http://leafletjs.com/examples/custom-icons/leaf-
shadow.png"
showon="pin:2"
/>
<field
type="text"
name="customPinShadowSize"
label="MOD_JDOSM_PINSHADOWSIZE"
description="MOD_JDOSM_PINSHADOWSIZE_DESC"
default="50, 64"
showon="pin:2"
/>
<field
type="text"
name="customPinOffset"
label="MOD_JDOSM_PINOFFSET"
description="MOD_JDOSM_PINOFFSET_DESC"
default="22, 94"
showon="pin:2"
/>
<field
type="text"
name="customPinPopupOffset"
label="MOD_JDOSM_PINPOPUPOFFSET"
description="MOD_JDOSM_PINPOPUPOFFSET_DESC"
default="-3, -76"
showon="pin:2"
/>
<field
type="spacer"
name="spacerPosMedial"
hr="true"
showon="pin:2"
/>
<field
type="list"
```

```

name="popup"
default="0"
label="MOD_JDOSM_POPUP"
description="MOD_JDOSM_POPUP_DESC"
>
<option value="1">MOD_JDOSM_KLICK</option>
<option value="2">MOD_JDOSM_ALWAYS</option>
<option value="0">MOD_JDOSM_NEVER</option>
</field>
<field
type="editor"
name="popuptext"
default=""
label="MOD_JDOSM_POPUPTEXT"
description="MOD_JDOSM_POPUPTEXT_DESC"
filter="safehtml"
showon="popup! : 0"
/>
</form>

```

tmpl/default.php

```

<?php
defined('_JEXEC') or die;
echo ModJdosmHelper::getHallo();
$document = JFactory::getDocument();
$document-
>addStyleSheet('https://unpkg.com/leaflet@1.2.0/dist/leaflet.css')
;
$document-
>addScript('https://unpkg.com/leaflet@1.2.0/dist/leaflet.js');
?>

```

```

<div style="
    width: auto;
    height:<?php echo $params->get('height', '400'); ?>px;" 
    id="map<?php echo $module->id; ?>">
</div>
<?php
$script = array();
$script[] = "<script>";

$script[] = ModJdosmHelper::getMap($module, $params);
$script[] = ModJdosmHelper::getTileLayer($module, $params);
$script[] = "mymap" . $module->id . ".setView([" .
    $params->get('lat', '46.93') . ", " .
    $params->get('lon', '7.43') . "], " .
    $params->get('zoom', 'false') . ")";
if($params->get('showpin', '1') == 1)
{
    $script[] = ModJdosmHelper::getpin($module, $params);
}
$script[] = "</script>";
echo implode("\n", $script);

```

helper

```

<?php
defined('_JEXEC') or die('');
class ModJdosmHelper
{
...
public static function getpin($module, $params)
{
// Standard Icon
$return = "";

```

```

foreach ($params->get('specialpins', null) as $specialpin)
{
if ($specialpin->pin == 1)
{
$return .= "L.marker([" . $specialpin->latpin . ", " .
$specialpin->lonpin . "])."
    . "addTo(mymap" . $module->id . ")";
}

// Custom Icon

if ($specialpin->pin == 2)
{
$iconimage = $specialpin->customPinPath;
$shadowimage = $specialpin->customPinShadowPath;
if (!(substr($specialpin->customPinPath, 0, 4) === "http"))
{
$iconimage = JURI::base() . $specialpin->customPinPath;
}

if (!(substr($specialpin->customPinShadowPath, 0, 4) === "http"))
{
$shadowimage = JURI::base() . $specialpin->customPinShadowPath;
}

$return .= 'var LeafIcon = L.Icon.extend({'
. 'options: {'
. 'iconUrl: \'' . $iconimage . '\','
. 'shadowUrl: \'' . $shadowimage . '\','
. 'iconSize: [' . $specialpin->customPinSize . '],'
. 'shadowSize: [' . $specialpin->customPinShadowSize . '],'
. 'iconAnchor: [' . $specialpin->customPinOffset . '],'
. 'popupAnchor: [' . $specialpin->customPinPopupOffset . ']''
. '}'
. '});';

$return .= 'var myIcon = new LeafIcon();'

$return .= "L.marker([" . $specialpin->latpin . ", " .
$specialpin->lonpin . "], {icon: myIcon})."
    . "addTo(mymap" . $module->id . ")";
}

```

```
if ($specialpin->popup == '1')
{
$return .= ".bindPopup('" . $specialpin->popuptext . "')";
}

if ($specialpin->popup == '2')
{
$return .= ".bindPopup('" . $specialpin->popuptext . "')"
. ".openPopup()";
}

$return .= ";" ;
}

return $return;
}

}
```

Sprachdateien

Todo

todos

todo GeoJson immer gleich geschrieben?

Methode und Funktion immer richtig?

Vor und hinter Quelltext eine leerzeile?

Bildunterschriften

Quotation Mache bei diesen auch etwas einheitlich?