Langage et environnement de base pour développement web

D'après le cours de Mr Mickaël Perrier
mickael.perrier@proton.me

Master II GéoNum, Promotion 2025

Université Lyon – Campus Porte des Alpes
03/11/2025

Bastien CAHIER – MCP Data Solutions

Organisation du cours

5 Modules de 4H (B. CAHIER & M. GRADELER)

- Quelques aspects théoriques
- Favoriser la mise en pratique
- Pas d'évaluation en cours
- Un petit projet à rendre (donné ultérieurement)

Programme de la séance #1

- Introduction à Leaflet
- Affichage de données
- Gestion des couches
- Gestion des GeoJSON

LEAFLET : Introduction

Solutions de WebCarto

ArcGis Online

Google Maps

Bing Maps

Propriétaires

OpenLayers

Leaflet

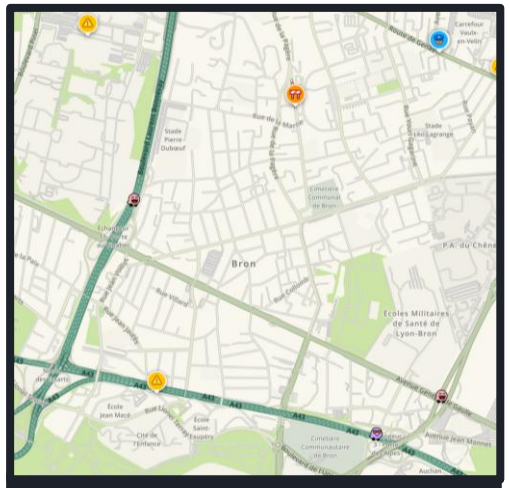
Libres

Leaflet

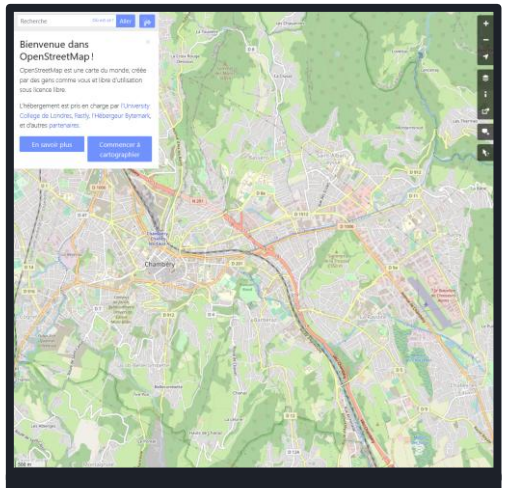
« Leaflet est une bibliothèque JavaScript libre de cartographie en ligne développée par Vladimir Agafonkin de CloudMade et de nombreux contributeurs.
Elle est notamment utilisée par le projet de cartographie libre et ouverte OpenStreetMap. »

- Créée en 2010
- 800 contributeurs sur Github
- Simple & léger
- Coté client
- Comporte de nombreux plugins
- Très utilisée
- Compatible avec de nombreux outils

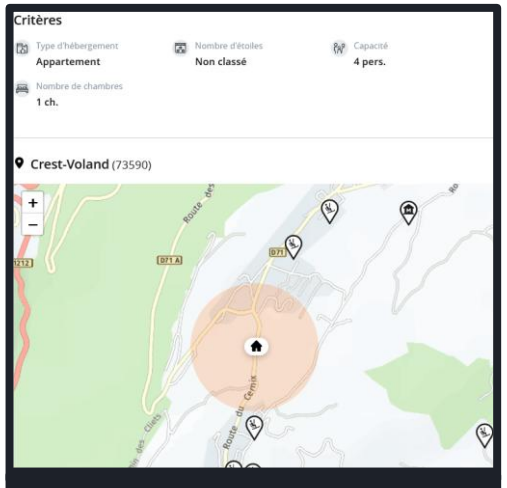
Quelques exemples



WAZE



openstreetmap.org



Annonce LeBonCoin

Et plein d'autres !

Leaflet : Les bases,
À vous de coder!

Premiers pas avec Leaflet

HTML

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Premiers pas avec Leaflet</title>
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
  <link rel="stylesheet" href="css/style.css" />
  <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
</head>
<body>
  <h1>Hello World !</h1>
  <div id="map"></div>
</body>
<script type="text/javascript" src="js/script.js"></script>
</html>
```

Premiers pas avec Leaflet

CSS

```css
body {
  background-color: rgb(25, 25, 27)
}

h1 {
  color:coral;
  font-size: 25pt;
}

#map {
  width: 500px;
  height: 500px;
}
```

Premiers pas avec Leaflet

JS

```
var map = L.map('map');
var osmUrl = 'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png';
var osmAttrib = 'Map data © OpenStreetMap contributors';
var osm = new L.TileLayer(osmUrl, {attribution:
osmAttrib}).addTo(map);
map.setView([45.719, 4.918], 17);
```

Quelques explications

```
var map = L.map('map');
```

↪ Initialisation d'un objet `map` qui sera affiché dans la `<div>` qui a l'id `'map'`

```
var osmUrl = 'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png';
```

↪ La variable `osmUrl` contient l'adresse vers les tuiles du fond de carte

```
var osmAttrib = 'Map data © OpenStreetMap contributors';
```

↪ La variable `osmAttrib` contient un texte (obligatoire?) qui indique la source des données. Il apparaîtra en bas à droite de la carte

```
var osm = new L.TileLayer(osmUrl, {attribution: osmAttrib}).addTo(map);
```

↪ Création d'une objet TileLayer qui va contenir les tuiles à afficher, et ajout à l'objet map.

```
map.setView([45.719, 4.918], 17);
```

↪ Positionnement de la carte (latitude puis longitude) et niveau de zoom (entre 1 et 19)

La doc' = La Bible

La documentation de Leaflet est disponible à cette adresse :

https://leafletjs.com/reference.html

Elle liste les différents éléments (classes, propriétés et fonctions) de Leaflet.

Fournisseurs de tuiles de fond de plan

Une liste de fonds de plan qu'il est possible d'intégrer à Leaflet :

https://leaflet-extras.github.io/leaflet-providers/preview/

⚠ Attention, certains jeux de tuiles demandent une clé d'API, et certains liens ne sont plus à jour ...

Afficher des données

L'affichage de données géographiques dans Leaflet :

`L.TileLayer`	Fond de plan
`L.marker`	Point
`L.circleMarker`	Point avec buffer
`L.circle`	Cercle
`L.polyline`	Ligne
`L.polygon`	Polygone
`L.popup - et bindPopup()`	Popup, affichée au clic
`L.tooltip - et bindTooltip()`	Tooltip, affiché au survol de la souris
`L.icon`	Créer ses propres icônes (pour les points)

Afficher des données : Marker

Ajouter le code suivant à votre script :

JS

```
var marker = L.marker([45.7179362, 4.9196902]);
marker.addTo(map);
```

- L'icône utilisée est celle par défaut
- Les coordonnées en `[latitude, longitude]` : c'est la norme avec Leaflet
- On fait ces opérations en 2 temps :
  1. On crée le point
  2. On l'ajoute à l'objet `map`

Pour supprimer un point:

JS

```
map.removeLayer(marker);
```

Afficher des données : CircleMarker

Ajouter le code suivant à votre script :

JS

```
var circle_marker = L.circleMarker(
    [45.7174886, 4.9190906],
    {radius: 20}
);
circle_marker.addTo(map);
```

↪ On ajoute une taille (en pixels) autour point à afficher (comme un buffer)

Afficher des données : CircleMarker

On peut ajouter plus de propriétés de style à l'élément `circleMarker` :

JS

```
var circle_marker = L.circleMarker(
    [45.7174886, 4.9190906],
    {
        radius: 20,
        color: '#FF0000',
        weight: 10
    }
);
```

↪ On peut modifier de nombreuses propriétés d'un circleMarker :
https://leafletjs.com/reference.html#circlemarker-option

Afficher des données : Circle

Ajouter le code suivant à votre script :

JS

```
var circle = L.circle(
    [45.71793, 4.91968],
    50,
    { color: 'green', fillOpacity: 0.7 }
);
circle.addTo(map);
```

↪ Attention : contrairement au `circleMarker`, le rayon du cercle est ici indiquée en mètres et pas en pixels !

Afficher des données : Polyline

Ajouter le code suivant à votre script :

JS

```
var line = L.polyline([[45.71964002, 4.9182032], [45.7221640,
4.9153070], [45.7275100, 4.9156503], [45.7315674, 4.9147154]])
line.addTo(map);
```

Afficher des données : Polygon

Ajouter le code suivant à votre script :

JS

```
var polygon = L.polygon([[45.7174322, 4.9187703], [45.7183342, 4.9199559], [45.7181565, 4.9202223], [45.7172580, 4.9190334]], {color: 'purple', fillOpacity: 0.5});
polygon.addTo(map);
```

↪ Comme une polyligne, mais le dernier point est automatiquement relié au premier

Afficher des données : Popup & tooltip

Ajouter le code suivant à votre script :

JS

```
circle.bindPopup("Coucou !");
polygon.bindTooltip("Les tooltips <br/> c'est aussi du <span
style='color: red'>HTML ! </span>")
```

↪ Le contenu des tooltips sont en HTML. Elles peuvent donc contenir des informations dynamiques, des boutons, etc.

↪ Il existe plusieurs propriétés pour personnaliser ces éléments :

- https://leafletjs.com/reference.html#popup
- https://leafletjs.com/reference.html#tooltip

Afficher des données : Popup & tooltip

Il est aussi possible de leur attribuer des styles CSS :

JS

```
circle_marker.bindTooltip("From CSS", {className: 'my_popup', permanent:
true})
```

CSS

```
.my_popup {
  background-color: darkslategrey;
  color:lavenderblush;
  border: 2px solid #FF24AA;
}
```

Afficher des données : Icône personnalisée

Ajouter le code suivant à votre script :

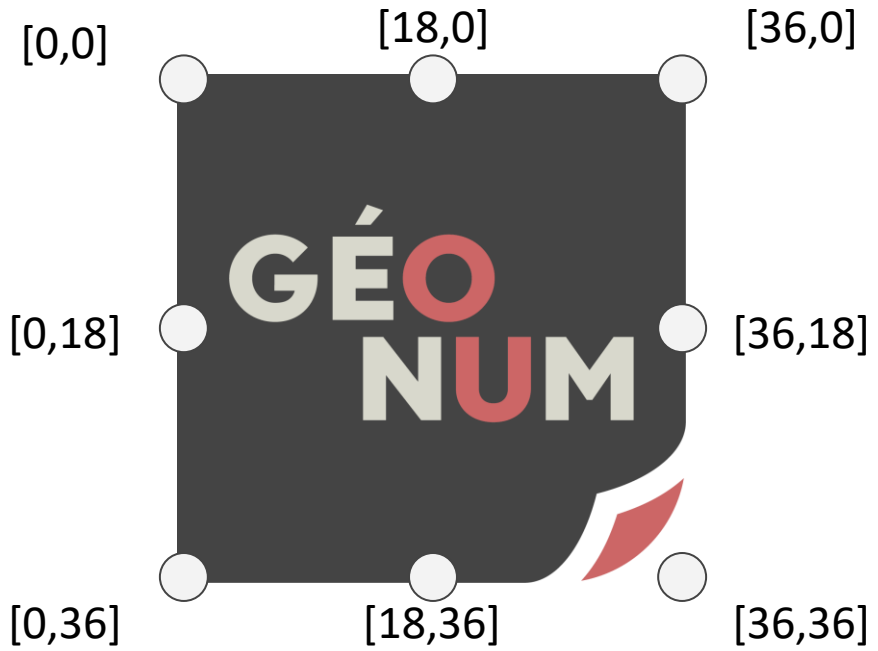
JS

```
var geonumIcon = L.icon ({
  iconUrl: 'img/logo.png',
  iconSize: [36, 36],
  iconAnchor: [18,36]
});
var geonum_marker = L.marker([45.7186024, 4.9192939], {icon: geonumIcon})
geonum_marker.addTo(map);
```

↪ Comme pour tous les éléments Leaflet, la liste des propriétés de l'élément `icon` est disponible sur la documentation de Leaflet : https://leafletjs.com/reference.html#icon

Afficher des données : Icône personnalisée

La propriété **`iconAnchor`** définit comment placer l'icône par rapport au point géographique sur la carte. On lui indique des coordonnées `[x,y]`, en fonction de la taille de l'icône :



Afficher des données : rappel

Cours :
exo_0.js
exo_0.html

L'affichage de données géographiques dans Leaflet :

`L.TileLayer`	Fond de plan
`L.marker`	Point
`L.circleMarker`	Point avec buffer
`L.circle`	Cercle
`L.polyline`	Ligne
`L.polygon`	Polygone
`L.popup - et bindPopup()`	Popup, affichée au clic
`L.tooltip - et bindTooltip()`	Tooltip, affiché au survol de la souris
`L.icon`	Créer ses propres icônes (pour les points)

Pause !

Mise en pratique

Leaflet : Les bases – Mise en pratique

```
var t2 = [[45.71964002, 4.9182032], [45.7221640, 4.9153070], [45.7275100, 4.9156503], [45.7315674, 4.9147154]];
var batiment_v = [[45.7174322, 4.9187703], [45.7183342, 4.9199559], [45.7181565, 4.9202223], [45.7172580, 4.9190334]];
```

JS

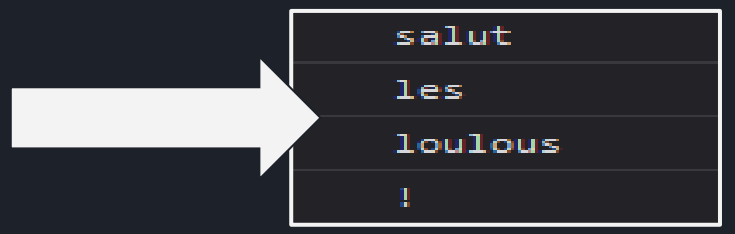
À partir des listes de coordonnées ci-dessus, cartographier avec Leaflet :

- Les stations de tram (variable `t2`)
- La ligne de tram (variable `t2`)
- Le bâtiment V (variable `batiment_v`)
- Des zones tampons de 300m autour de chaque station
- Afficher les stations avec une icône personnalisée

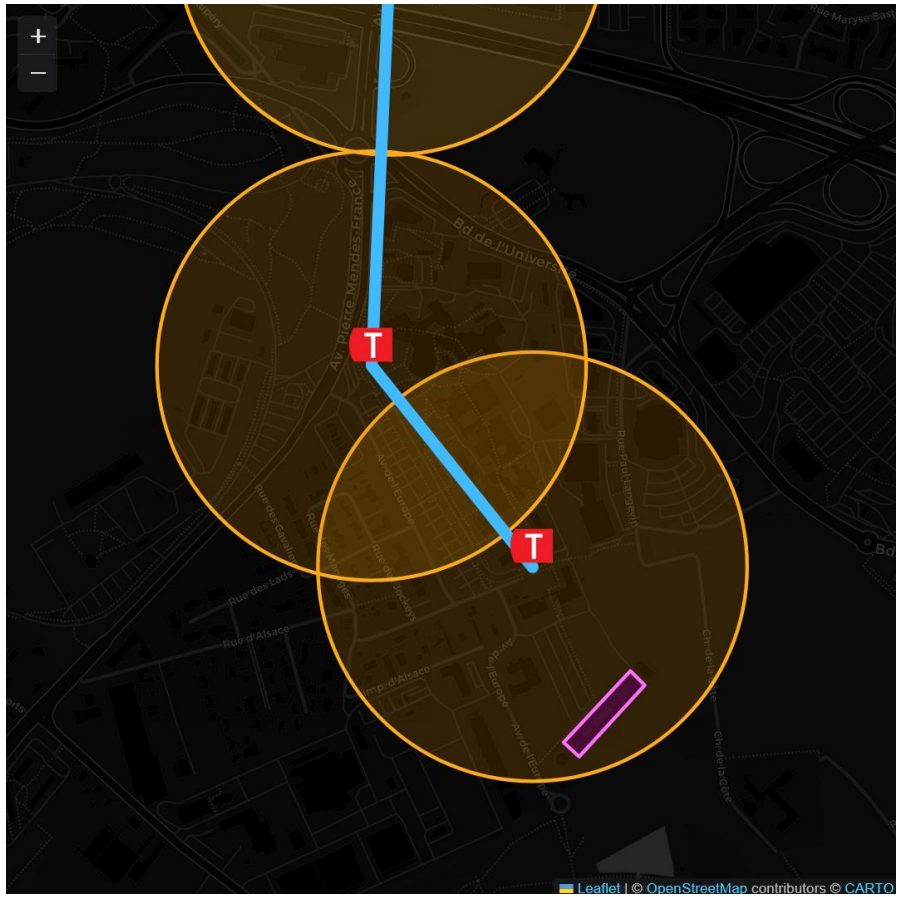
Cours :
exo_1.js
exo_1.html

↪ Rappel : le fonctionnement d'une boucle for en JavaScript

```
var tableau = ["salut", "les", "loulous", "!"];
for (var i = 0; i < tableau.length; i++) {
  console.log(tableau[i]);
};
```



Exercice n°1 : résultat

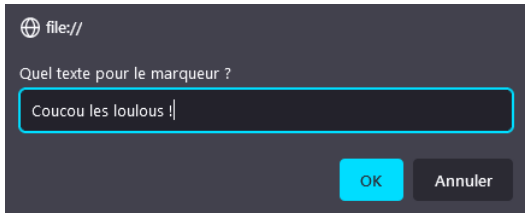


Cours :
exo_1.js
exo_1.html

Exercice n°2

Ajouter une étiquette (popup ou `tooltip`) à tous les arrêts de tram. Le contenu de cette étiquette doit être demandé à l'utilisateur à l'ouverture de la page

Cours :
exo_2.js
exo_2.html



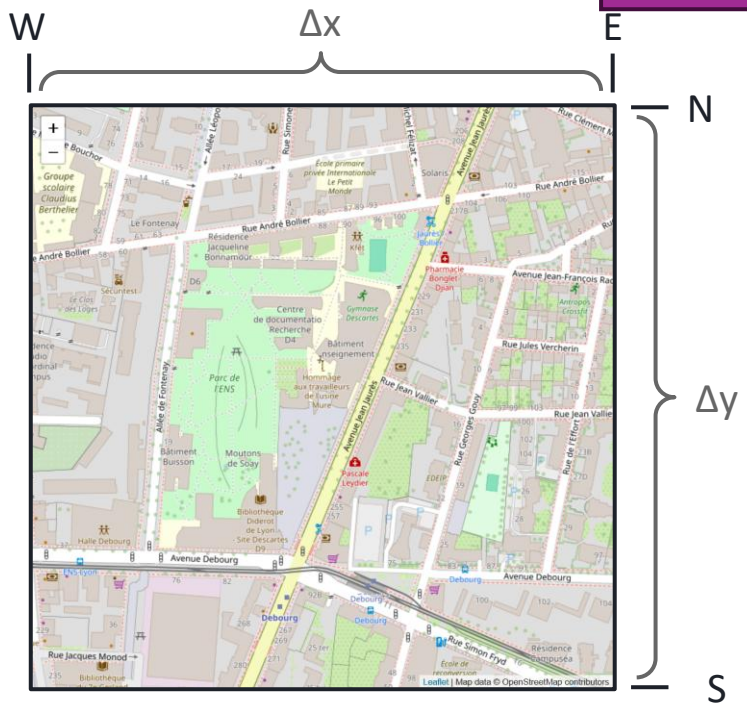
Exercice n°3

Dessiner l'emprise de la carte au chargement et placer un marqueur de coordonnées aléatoires dans la carte au lancement de la page et

Cours :
exo_3.js
exo_3.html

↪ Récupérer les coordonnées min/max de la carte

```
var bounds = map.getBounds();
var north_bound = bounds.getNorth();
...
```



Gestion des couches

Leaflet : Gestion des couches

Ajouter le code suivant à votre script :

JS

```
var baseLayers = {
  "OpenStreetMap": osm
};
var overlays = {
  "Ligne tram": tram_line,
  "Bâtiment V": batiment_v_poly
};
L.control.layers(baseLayers, overlays).addTo(map);
```

- `baselayer` = fond de plan, 1 seul à la fois
- `overlays` : couches qui viennent se superposer au fond de plan

Leaflet : Gestion des couches

Regrouper des couches dans un groupe :

JS

```
var lyon = L.marker([x,y]);
var cities = L.layerGroup([lyon, paris, marseille, saint_etienne]);
cities.addTo(map);
```

Ou

JS

```
var cities = L.layerGroup();
var lyon = L.marker([x,y]).addTo(cities);
cities.addTo(map);
```

On pourra contrôler toutes les couches du groupe simultanément!

Leaflet : Gestion des couches

Cours :
exo_4.js
exo_4.html

Exercice n°4

1. Ajouter un fond de carte satellite depuis https://leaflet-extras.github.io/leaflet-providers/preview/

2. Ajouter tous les arrêts de tram à un groupe de couche

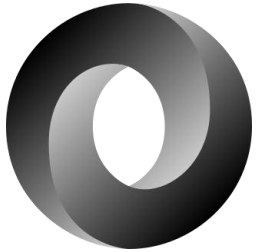
3. Paramétrer le contrôle d'affichage des couches :
   - 2 fonds de carte
   - 3 couches : bâtiment, tram, arrêts de tram

```
var couches_groupees = L.layerGroup();
couches_groupees.addTo(map);
ma_couche.addTo(couches_groupees)
```

```
var baseLayers = {
  "OpenStreetMap": osm,
  « Satellite": fond_sat,
};
var overlays = {
  "Ligne tram": tram_line,
  "Bâtiment V": batiment_v_poly,…
};
L.control.layers(baseLayers,
overlays).addTo(map);
```

Gestion des GeoJSON

Le format JSON



JSON

« JavaScript Object Notation (JSON) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. »

```json
{
  "menu": {
    "id": 001,
    "value": "File",
    "popup": {
      "menuitem": [
        {
          "value": "New",
          "onclick": "CreateNewDoc()"
        },
        {
          "value": "Open",
          "onclick": "OpenDoc()"
        },
        {
          "value": "Close",
          "onclick": "CloseDoc()"
        }
      ]
    }
  }
}
```

Le format GeoJSON

GeoJSON

« GeoJSON (Geographic JSON) est un format ouvert d'encodage d'ensemble de données géospatiales simples utilisant la norme JSON (JavaScript Object Notation). Il permet de décrire des données de type point, ligne, chaîne de caractères, polygone, ainsi que des ensembles et sous-ensembles de ces types de données et d'y ajouter des attributs d'information qui ne sont pas spatiales. »


```
{
  "type": "FeatureCollection",
  "name": "station_autopartage_WGS84",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
    }
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "nom": "LPA Croix-Rousse",
        "commune": "Lyon 4ème",
        "typeautopa": "Citiz LPA",
        "nbemplacem": "3",
        "anneereali": "2011",
        "gid": "1"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          4.833139860859754,
          45.781409359390885
        ]
      }
    },
```

Le format GeoJSON

Le GeoJSON supporte plusieurs SCR, **Mais leaflet travaille principelement en WGS84!**

Télécharger un GeoJSON des stations de vélo'v :

https://data.grandlyon.com/jeux-de-donnees/stations-velo-v-metropole-lyon/info

⚠ Attention à bien récupérer les données en WGS84 ! ⚠

Charger un GeoJSON en JavaScript

Il y a 2 manières de charger un GeoJSON en JavaScript :

Utiliser la fonction `fetch()` de JavaScript sur un fichier `.geojson`

Copier-coller le contenu d'un fichier `.geojson` dans une variable JavaScript

Charger un GeoJSON en JavaScript

Le moyen le plus simple consiste à copier le contenu d'un fichier `.geojson` dans la variable d'un fichier `.js` :

GeoJSON

```
{
  "type": "FeatureCollection",
  "name": "nrj_energie.nrjinstallphotovolt_1",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "commune": "Albigny-sur-Saône",
        "insee": "69003",
        "gid": "68"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              4.817725199912649,
              45.855968715527567
            ],
            [
              4.817662459036137,
              45.855995906349136
            ],
            [
              4.817716382244889,
              45.856057596707501
            ]...
```

JS

```
var toto = {
  "type": "FeatureCollection",
  "name": "nrj_energie.nrjinstallphotovolt_1",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "commune": "Albigny-sur-Saône",
        "insee": "69003",
        "gid": "68"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              4.817725199912649,
              45.855968715527567
            ],
            [
              4.817662459036137,
              45.855995906349136
            ],
            [
              4.817716382244889,
              45.856057596707501
            ]...
```

Charger un GeoJSON en JavaScript

Cette méthode est certes un peu bancale, mais pour ne pas s'y perdre, on va quand même appliquer une règle simple :

1 fichier `.geojson` = **1** fichier JavaScript = **1** variable

Les fichiers contenant des données `geojson` seront à déposer dans le dossier `data`. Il faut ici bien différencier les fichiers `.js` de traitements des fichiers `.js` de données.

Charger un GeoJSON en JavaScript

Il faut ensuite charger ce fichier JS dans le HTML :

```
<head>
  ...
  <script src="data/base.js"></script>
  ...
</head>
```

Charger un GeoJSON dans Leaflet

Puis le charger la *variable* qui contient le geojson dans Leaflet :

JS

```
L.geoJSON(toto).addTo(map);
```

⚠ Ceci est la méthode de chargement d'un GeoJSON par défaut : elle est très limitée. Les points sont des `L.marker`, les lignes des `L.polyline` et les polygones des `L.polygon`. On a aucun contrôle sur le style, les tooltips, etc.

💡 Pour la liste de toutes les options disponibles pour `L.geoJSON`, ne pas oublier la doc : https://leafletjs.com/reference.html#geojson

Styliser un GeoJSON

Pour styliser nos éléments :

JS

```
var toto_data = L.geoJSON(toto, {
  pointToLayer: function (feature, latlng) {
    return L.circleMarker(latlng);
  },
  style: my_style
}).addTo(map);
```

Définir le style dans un objet JavaScript, et possiblement dans un fonction :

JS

```
function my_style(feature) {
  return {
    color: 'red',
    weight: 2,
    opacity: 0.5
  }
};
```

Une petite fonction TRÈS utile …

Pour adapter l'emprise de la carte à différents éléments :

JS

```
map.fitBounds(toto_data.getBounds())
```

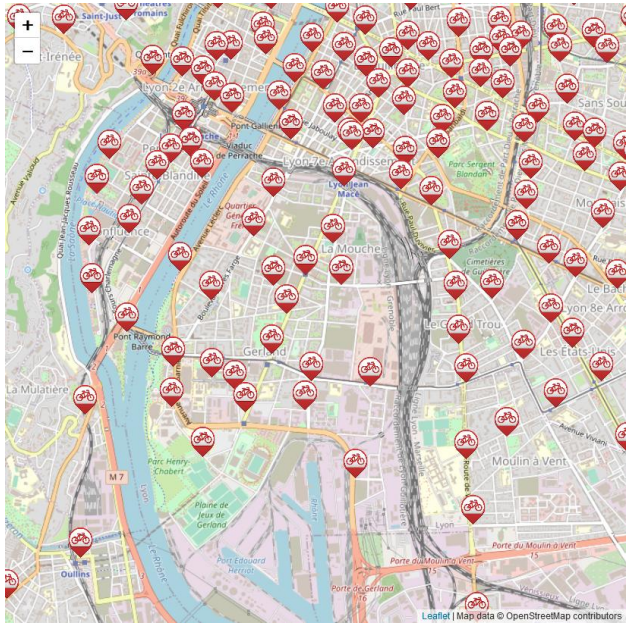
Note 1 : la variable à utiliser (ici `toto_data`) doit être un **objet** Leaflet
(un objet Leaflet = `L.qq_chose`)

Note 2 : À placer après avoir chargé vos éléments `L.marker`, `L.polygon`, `L.geoJSON`, etc.

Exercice n°5

Cours :
exo_5.js
exo_5.html

1. Récupérer un GeoJSON des stations de vélo'v⚠ Attention à bien changer le SCR en WGS84 !
1. Copier le contenu dans la variable d'un fichier JS, et charger ce fichier dans votre HTML
1. Utiliser cette donnée pour afficher les stations avec une icône personnalisée
1. Définir l'emprise de la carte par rapport aux données



https://data.grandlyon.com/jeux-de-donnees/stations-velo-v-metropole-lyon/info

Pause !

Réutiliser les données attributaires

Rappel : un objet JavaScript est un ensemble de clés-valeurs.
On peut ainsi accéder aux données attributaires contenues dans un GeoJSON avec Leaflet :

GeoJSON

```
"features": [
    {
      "type": "Feature",
      "properties": {
        "nom": "Bancel \/ Chevreul"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          4.840524571672513,
          45.74852028673056
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "nom": "Perrin \/ Jean Jaurès"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          4.885609434424513,
```

JS

```
var stations_velov_layer =
L.geoJSON(stations_velov_wgs84, {
  pointToLayer: function (feature, latlng) {
    return L.marker(latlng, {icon: velovIcon});
  }
}).addTo(map);
map.fitBounds(stations_velov_layer.getBounds())
```

Réutiliser les données attributaires

Une propriété intéressante de l'objet L.geoJSON est `onEachFeature`, qui permet d'appliquer divers traitements sur chaque feature (pas seulement les points) :

JS

```
var stations_velov_layer = L.geoJSON(stations_velov_wgs84, {
  onEachFeature: function (feature, layer) {
    ...
  }
}).addTo(map);
```

- `feature` : l'objet GeoJSON
- `layer` : l'objet Leaflet (oui, ça s'appelle un « layer », c'est assez mal nommé ...)

Exercice n°6

Cours :
exo_6.js
exo_6.html

1. Afficher les stations de vélo'v d'une couleur différente en fonction de la capacité d'accueil

Utiliser :
- feature.properties.nbbornettes
- feature.properties.nom

2. Créer un popup avec le nom de la station et la capacité d'accueil.

```
var stations_velov_layer = L.geoJSON(stations_velov_wgs84, {
  {
  pointToLayer: function (feature, latlng){
  return L.circleMarker(latlng,{color:feature.properties.color});
  },
   onEachFeature: function (feature, layer) {...}
}).addTo(map);
```

Merci de votre attention

Questions ?