Programmation web: JavaScript

JP Kasprzyk – 2021

Notes de cours: http://geomatics.ulg.ac.be/download/sigweb/cours/

- **1. Les bases du JavaScript (JS)**
  - **Caractéristiques**
    - » **Langage dynamique**
      - Le contenu de la page web se modifie dynamiquement (sans besoin de modification du code) en fonction de facteurs externes:
        - Chargement de la page web
        - Action de l'utilisateur (ex: clic sur un bouton)
        - Réception de données provenant du serveur
        - Action répétée dans le temps (ex: afficher l'heure)
      - Série d'**instructions** séparées par des « ; », utilisation de **variables**, de **structures de contrôle** (if, while, for, …) et de **fonctions**
    - » **Langage client**
      - Langage exécuté par le navigateur (idem HTML et CSS)
      - Exception: serveur web NodeJS
    - » **Langage interprété**
      - Le code est lu et compris tel quel par le navigateur (idem HTML et CSS)
      - A l'opposé, un langage compilé nécessite une transcription du code en langage machine (compilation)
    - » **Langage orienté objet**
      - Exploitation des notions de classes (attributs et méthodes) et d'héritage

– Où placer le code JS?

» Dans l'attribut « événement » d'un élément (balise) HTML quelconque

- Le code est exécuté si l'événement a lieu sur cet élément

```
<button onclick="alert('Bonjour !')">Cliquez moi</button>
```

» Dans un élément <script> du document HTML

```
<script>

        let bonjour = document.getElementById('b1');
        bonjour.addEventListener('click', alerte);

        function alerte(){
            alert('Bonjour');
        }

</script>
```

» Dans un fichier externe (fichier.js) appelé dans le <head> ou à la fin du <body>

```
<script src='cours.js' async></script>
```

– Attention à l'ordre d'exécution du code!

» Le code JS ne doit pas faire référence à un élément HTML qui ne soit pas déjà lu par le navigateur (sauf dans la définition d'une fonction ou d'une classe)

» Il vaut donc mieux mettre les élements <script> et les appels de fichiers js à la fin du <body>

– Les variables JavaScript

» Déclaration

- var x : variable globale (sauf si déclarée dans une fonction)
- const x : constante
- let x : variable dont la portée se limite à un bloc {…}

» Types de valeurs

- string ou « chaine de caractères » ;
- number ou « nombre » ;
- boolean ou « booléen » ;
- null ou « nul / vide » ;
- undefined ou « indéfini » ;
- symbol ou « symbole » ;
- **object** ou « objet » ;

Valeurs primitives

– Les fonctions JavaScript

» **Fonction** = bloc de code nommé* et réutilisable dont le but est d'effectuer une tâche précise

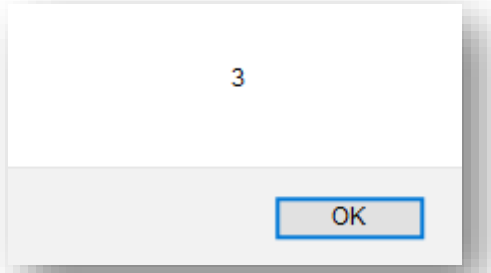
- La fonction porte un nom*: MaFonction()
- La fonction accueille des paramètres: MaFonction(p1, p2)
- La fonction retourne un output: var result = MaFoncton(p1,p2)

```
<script>

  function somme(a, b) {
      var output = a + b;
      return output
  }
  var result=somme(1,2);
  alert(result);

</script>
```

Définition de la fonction « somme »

Appel de la fonction « somme »

Appel de la fonction native « alert » ➔ ouverture d'un popup



» **Intérêt** = une même fonction peut être appelée à plusieurs endroits du code ➔ économie de code

» *Une fonction peut être anonyme si elle est appelée une seule fois dans le code (par exemple: fonction appelée par un événement)

» Attention à ne pas confondre fonction et méthode!

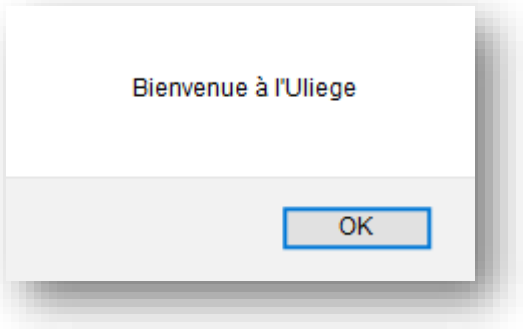
– Les méthodes JavaScript

» **Méthode** = fonction définie au sein d'un objet et s'appliquant sur celui-ci.

- Exemple de méthode native « replace » s'appliquant sur un objet « string »:

```
<script>
  var str = "Bienvenue à l'ULg"
  var res = str.replace("ULg", "Uliege");
  alert(res);
</script>
```

Objet Méthode



- Note: les valeurs primitives String, Number, Boolean et Symbol sont aussi des objets JS natifs avec leurs propres propriétés et méthodes

- ## 2. Javascript orienté objet
  - ### Rappel: la notion de classe et d'objet
    - » En programmation orientée objet, on appel **instance d'une classe** un **objet** avec un comportement et un état définis par la classe
      - Exemple:
        - tous les cyclomoteurs ont un ensemble de **propriétés** (= **attributs** = état)
          - Cylindrée
          - Couleur
          - Vitesse max
        - Tous les cyclomoteurs ont un ensemble de **méthodes** (= **opérations** = comportement)
          - Démarrer()
          - Arrêter()
          - Rouler()
    - » Une classe « fille » peut hériter des propriétés d'une classe « mère »
      - Exemple: tous les cyclomoteurs sont des véhicules
    - » A l'instar des fonctions, les classes permettent une économie considérable du code!

Représentation UML d'une classe

– Objet JavaScript littéral (sans classe)

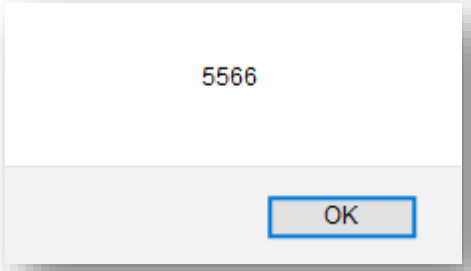
- » L'objet littéral n'est l'instance d'aucune classe
- » Ses propriétés et méthodes sont définies au sein-même de l'objet
- » Note: « this » fait référence à l'objet en cours

```
var jp = {
  firstName: "Jean-Paul",
  lastName : "Kasprzyk",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

alert(jp.id);

alert(jp.fullName());
```

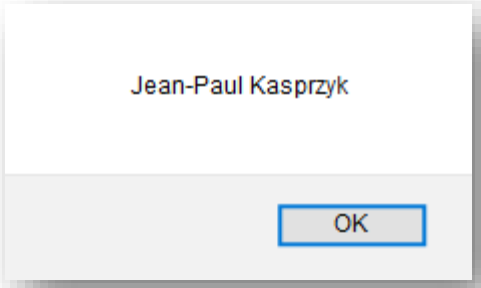
Définition propriétés

Définition méthode

Appel de la propriété « id » de l'objet « jp »



Appel de la méthode « fullName() » de l'objet « jp »



– Objet JavaScript comme instance de classe

» Les propriétés et méthodes sont définies au sein d'une classe

» L'objet est instancié au moyen d'une fonction « constructeur »

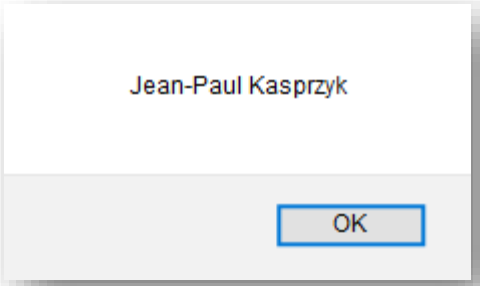
```
class person {
   constructor(nom, prenom, id) {
     this.nom = nom;
     this.prenom = prenom;
     this.id=id;
   }

   fullName() {
       return this.prenom + " " + this.nom;
   }
}

var jp = new person("Kasprzyk", "Jean-Paul", 1);

alert(jp.fullName());
```

Constructeur et propriétés

Méthode

Instanciation

Appel de la méthode « fullName() » de l'objet « jp »



– Classe et méthode statique

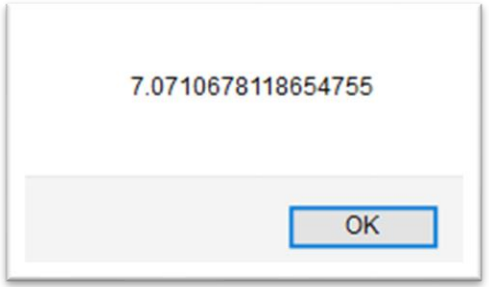
» Une méthode statique porte sur la classe en général et non sur une instance

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  static distance(a, b) {
    var dx = a.x - b.x;
    var dy = a.y - b.y;
    return Math.hypot(dx, dy);
  }
}

var p1 = new Point(5, 5);
var p2 = new Point(10, 10);

alert(Point.distance(p1, p2));
```

Constructeur et propriétés

Méthode statique

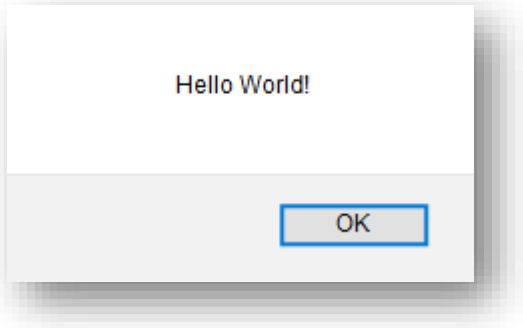
Instanciation de deux objets



Appel de la méthode statique « distance » entre deux instances de classe « point »

- **3. Le DOM et JavaScript**
  - » Tout comme CSS, JavaScript peut interagir avec les éléments du DOM grâce aux méthodes natives de l'objet « **document** »
  - – **Du DOM vers JS**
    - Exemple: récupération du contenu d'une balise HTML dans une variable JS

```
<!DOCTYPE html>
<html>
<body>
<div id="para1">Hello World!</div>
<script>
    var hello=document.getElementById("para1").innerHTML;
    alert(hello);
</script>
</body>
</html>
```



- La méthode **getElementById** retourne un objet « **élément** » (= **nœud** = balise du document HTML)
- Le contenu html de cet objet se trouve dans sa propriété « innerHTML »
- Les objets « **document** » et « **element** » sont appelés « **interfaces** »

– **De JS vers le DOM**

» Exemple: alimentation du contenu d'une balise HTML par JS

```
<!DOCTYPE html>
<html>
<body>
<div id="para1"></div>
<script>
  document.getElementById("para1").innerHTML="<p>Hello World!</p>"
</script>
</body>
</html>
```

Hello World!

- **Quelques méthodes de l'interface « document »**
  - » **getElementById(x)**: retourne l'élément avec l'ID x
  - » **getElementsByClassName(x)** : retourne tous les éléments de classe x sous la forme d'un tableau ou « array » (idem pour les deux méthodes suivantes)
  - » **getElementsByTagName(x)**: retourne tous les éléments du type sémantique « x » (ex: p, div, a, …)
  - » **getElementsByName(x)**: retourne tous les éléments portant le nom x

```
<a id='lien1' name='geo' class='lien' href='www.geomatics.ulg.ac.be'>
    Lien vers le site de l'Unité de Géomatique
</a>
```

- **Quelques propriétés de l'interface « document »**
  - body
  - head
  - title
  - cookie

- **Quelques propriétés de l'interface « element »**
  - » **parentNode**: élément parent dans le DOM
  - » **children**: éléments enfants
  - » **innerHTML**: contenu HTML de l'élément
  - » **style**: style CSS
    - L'objet style contient les propriétés CSS: color, display, margin, background, …

```
<!DOCTYPE html>
<html>
<body>
<div id="content">
  <p>Hello</p>
  <p>World!</p>
</div>
<script>
  var enfants=document.getElementById("content").children;
  for (valeur of enfants){
      valeur.style.color='blue';
  }
</script>
</body>
</html>
```

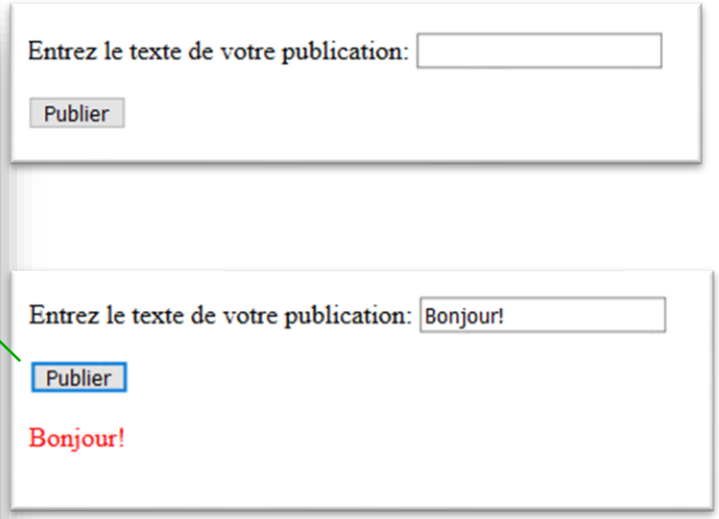
Hello

World!

- **4. Les événements JavaScript**
  - **Généralités**
    - » Les **événements** permettent de déclencher une fonction selon qu'une action se soit produite ou non
    - » Les événements sont généralement associés à un **élément HTML** (ex: <button>, <select>) comme propriétés
    - » Les événements sont généralement **provoqués par l'utilisateur**

```
<!DOCTYPE html>
<html>
<body>
<p>
  Entrez le texte de votre publication:
  <input id="input1" type="text"/>
</p>
<button id="bouton1" onclick="publi(input1.value)">Publier</button>
<div style="color:red" id="para1"></div>

<script>
  function publi(text){
      document.getElementById("para1").innerHTML="<p>"+text+"</p>";
  }
</script>
</body>
</html>
```

Appel de la fonction « publi »
Evénement « onClick »



– Quelques événements courants

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

- **5. JQuery**
  - **Définition**
    - » A l'instar de Bootstrap pour le CSS, Jquery est un **framework** simplifiant la syntaxe JavaScript pour de nombreuses fonctionnalités
      - Parcours et modification du DOM
      - Utilisation des sélecteurs CSS
      - La gestion des événements
      - Animations
      - Les requêtes AJAX
  - **Caractéristiques**
    - » Encore très utilisé aujourd'hui, même si probablement voué à disparaître au profit d'autres frameworks plus récents (ex: React.js)
    - » Beaucoup d'autres librairies populaires en dépendent (ex: DataTable.js, Chart.js)
    - » Très présent sur les « forums de geeks »
    - » Jquery n'empêche pas l'utilisation de la syntaxe classique JS!
  - **Installation: https://jquery.com/**
    - » Fichier .js à inclure comme une librairie dans votre page web

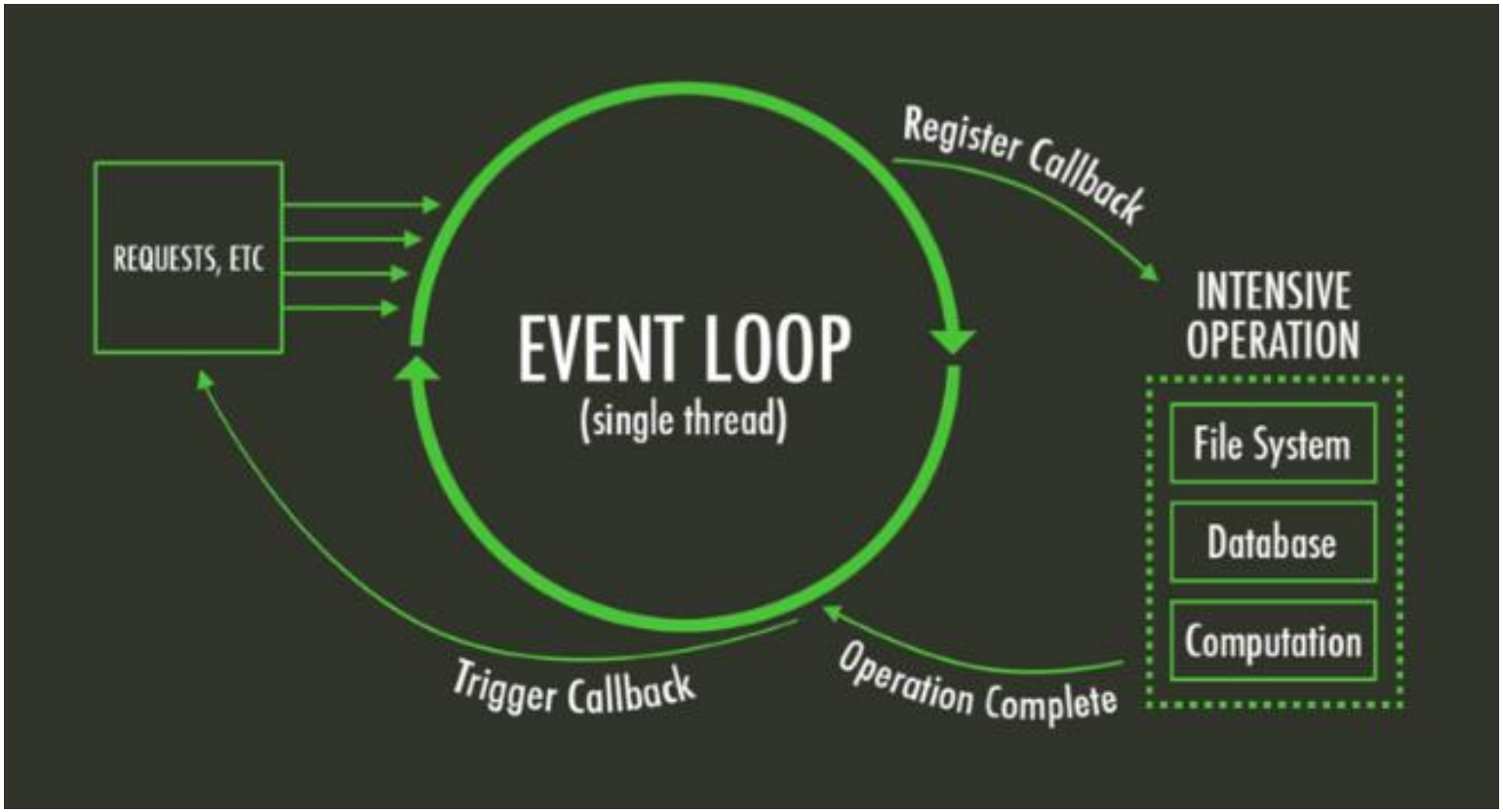
```
<script src="/chemin/vers/jQuery.js"></script>
```

- **Syntaxe**
  - » Toutes les instructions dépendent:
    - Soit de l'objet Jquery, noté « jquery » ou « $ »
    - Soit de la fonction Jquery, notée « jquery() » ou « $() »
- **Exemple de code Jquery**
  - » Affichage d'une bannière en cliquant sur un bouton

```
<script>
  var hiddenBox = $( "#banner-message" );
  $( "#button-container button" ).on( "click", function( event ) {
    hiddenBox.show();
  });
</script>
```

- **6. L'asynchronisme JavaScript**
  - **Définitions informatiques**
    - » Deux instructions sont **synchrones** si la seconde attend la fin de la première avant de démarrer
    - » Deux instructions sont **asynchrones** si la seconde n'a pas besoin d'attendre la fin de la première pour démarrer
  - **Intérêt**
    - » L'asynchronisme est une des forces de JS
      - L'exécution d'une tâche asynchrone longue **ne bloque pas** le reste de l'application
        - Ex: chargement de données provenant du serveur
      - Permet la création d'applications web **fluides** aussi bien côté client que côté serveur (NodeJS)
      - Particulièrement utile en « **web mapping** » pendant le chargement et le traitement des couches cartographiques

– **L'asynchronisme JS, comment ça marche?**



- **Les requêtes AJAX (« Asynchronous JavaScript and XML »)**
  - » **Dans les années 90**, les sites web étaient conçus principalement sur base de HTML
    - Le moindre échange de données avec le serveur nécessitait un **rechargement** complet de la page:
      - Exemple: envoi de formulaire via l'input de type « submit »
  - » **Aujourd'hui**, JS et la technologie AJAX permettent une **communication asynchrone** avec le serveur sur une même page web
    - Indispensable pour le « web mapping »
  - » A la base, AJAX fut conçu pour échanger les données au format **XML**
    - Aujourd'hui, le XML est de plus en plus délaissé au profit du format **JSON**
  - » La syntaxe JS d'AJAX étant très lourde, nous verrons directement celle de **JQuery**

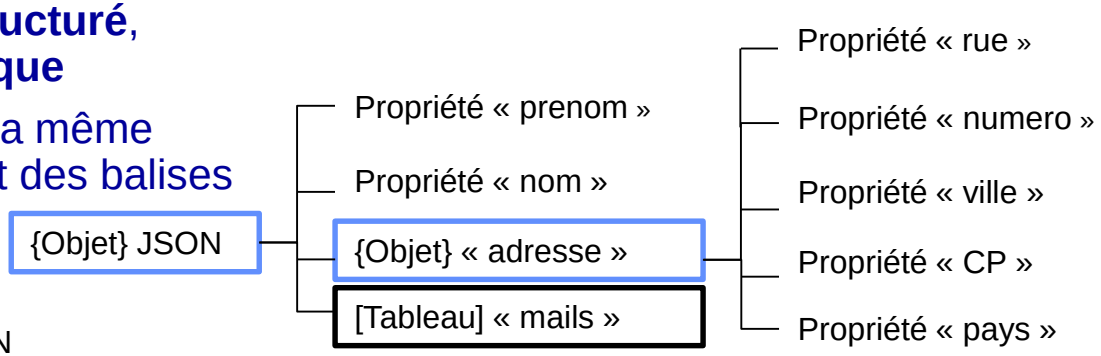
» Exemple Ajax + JQuery

```
<script>

$.ajax({
  type: 'post',
  url: 'ajax/traitement_serveur.php',
  data: 'db='+db+'&sql_select='+sql_select+'&sql_from='+sql_from+'&sql_where='+sql_where,
  dataType: "json",
  async: true,
  success: function (json){
      traitement_client(json);
  },
  error: function() {
      alert('Server Error')
  }
});

</script>
```

- **7. Le format JSON (« JavaScript Objet Notation »)**
  - **Généralités**
    - » **JSON** = format d'échange de données très populaire sur le web
    - » **Sérialisation** d'objets, nombres, chaînes de caractères, booléens, valeurs nulles et tableaux (« arrays »)
    - » Format **compact** et efficace
    - » La **syntaxe** est la même que celle des objets **JavaScript** ➔ communication aisée entre JS et JSON
    - » Formalisme **non-structuré**, flexible et **hiérarchique**
    - » Le format **XML** suit la même logique en exploitant des balises

```
{
    "prenom": "Jean-Paul",
    "nom": "Kasprzyk",
    "adresse":{
        "rue": "rue de la Géomatique",
        "Numero": 1,
        "ville": "Liège",
        "CP": 4000,
        "Pays": "Belgique"
    },
    "mails":[
        "jp.kasprzyk@uliege.be",
        "jp.kasprzyk@ulg.ac.be"
    ]
}
```

Exemple de JSON:

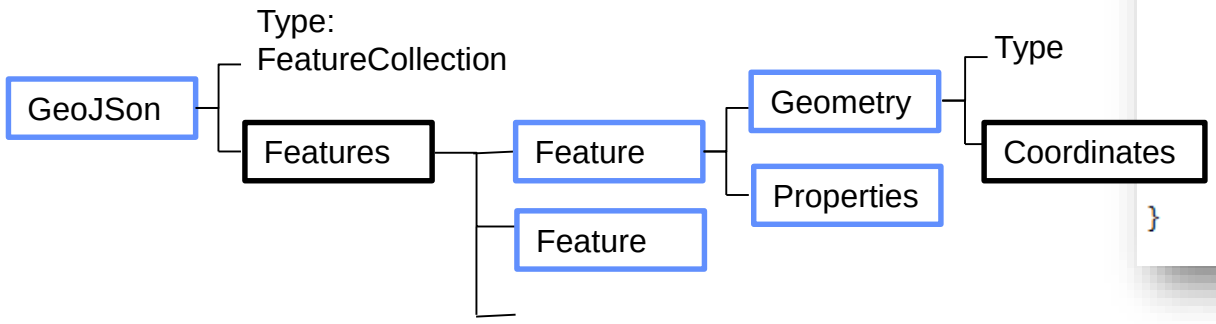


Notes: les méthodes ne sont pas gérées par JSON

– **Import / export**
  » Même si la syntaxe est la même, un **JSON n'est pas un objet JavaScript**
  » JSON est un format de texte indépendant de tout langage
    • Importable vers un objet JS: **JSON.parse()**
    • Exportable depuis un objet JS: **JSON.stringify()**
  » **Note**: parse() et stringify() sont deux méthodes de l'objet natif JSON
  » L'export et l'import JSON sont disponibles dans de **nombreux langages et outils**:
    • Python
    • PHP
    • Java
    • QGIS
    • …

– GeoJSON

- » Extension de JSON pour l'échange d'**entités spatiales vectorielles**
- » Facilement importable/exportable dans une **libraire cartographique JS** (OpenLayers, Leaflet, …)
- » Définition de « **Features** » = Objets ayant une implantation spatiale
- » Système de coordonnées de référence = WGS 84 non-projeté (**EPSG 4326**)




```geojson
{
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "geometry": {
            "type": "Point",
            "coordinates": [102.0, 0.5]
        },
        "properties": {
            "prop0": "value0"
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [102.0, 0.0],
                [103.0, 1.0],
                [104.0, 0.0],
                [105.0, 1.0]
            ]
        },
        "properties": {
            "prop0": "value0",
            "prop1": 0.0
        }
    }
]
}
```

Exemple de GeoJSON

Type	Examples
Point	

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

LineString	

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

Polygon	

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

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

Exemples de géométries simples
et de multi-géométries décrites en GeoJSON

Type	Examples
MultiPoint	

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

MultiLineString	

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

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]]
        ]
    ]
}
```

- **TopoJSON**
  - » Extension de GeoJSON gérant la **topologie** des entités géographiques
    - Exploitation de la **structure arc-nœud** (voir cours SIG) pour modéliser des frontières partagées entre entités
      - Requêtes topologiques peuvent être calculées directement en JS en exploitant notamment des notions d'adjacence entre entités
      - Elimination des redondances géométriques de GeoJSON
    - Toutes les **coordonnées** sont exprimées en **nombres entiers** (integer)
      - Transformation de WGS84 vers coordonnées locales TopoJSON (premières coordonnées du fichier = 0,0) avec conservation des paramètres de translation et de mise à échelle
      - Chaque couple de coordonnées est exprimé relativement par rapport à la coordonnées précédente

    ➔ **Réduction de la quantité d'information** par rapport à GeoJSON de l'ordre de 80%!

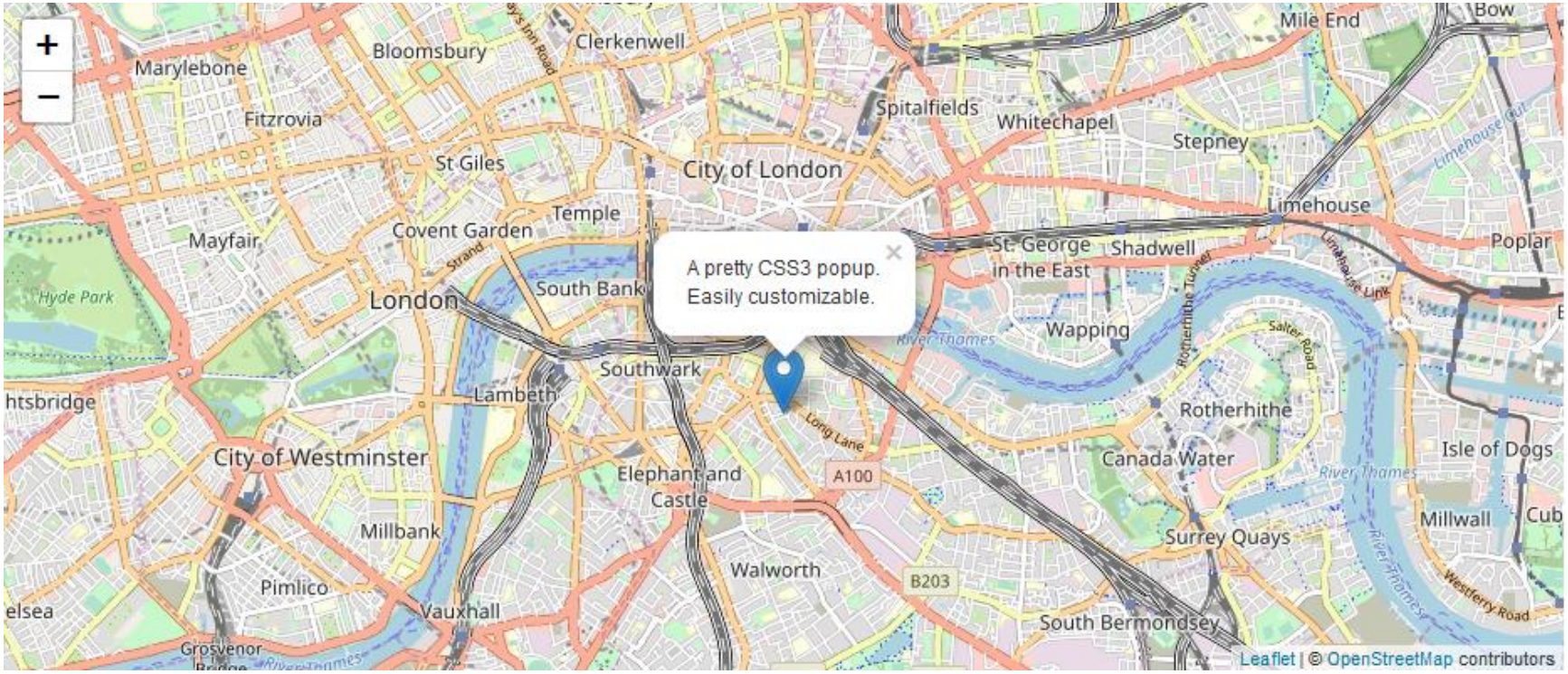
» Exemple: modélisation TopoJSON d'une entité géographique (île d'Aruba)

```
{
  "type": "Topology",
  "transform": {
    "scale": [0.0360036003600360005, 0.01736158967459246],
    "translate": [-180, -89.99892578124998]
  },
  "objects": {
    "aruba": {
      "type": "Polygon",
      "arcs": [[0]],
      "id": 533
    }
  },
  "arcs": [
    [[3058, 5901], [0, -2], [-2, 1], [-1, 3], [-2, 3],
    [0, 3], [1, 1], [1, -3], [2, -5], [1, -1]]
  ]
}
```

- **8. Web mapping**
  - **Principes**
    - » Gestion JS de **services web spatiaux** (WMS, WFS, …) côté client
    - » Gestion JS des **données spatiales** dans un navigateur web (côté client)
      - Récupération des **données brutes** vectorielles (GeoJSON, …)
        - Envoi des critères de sélections de l'utilisateur au serveur = paramètres de construction de requêtes SQL
        - Requêtes AJAX
      - **Représentation cartographique**
        - Gestion symbologie et généralisation
        - Superposition de couches et gestion des systèmes de coordonnées
          - En général, les données sont projetées en « **Pseudo-Mercator sphérique** » (EPSG 3857)
            - Compatibilité avec services web Open Street Map (OSM) et Google
      - **Autres représentations** (graphiques, tableaux, …)
      - Application éventuelle de **traitements spatiaux côté client**, notamment:
        - Création de nouvelles entités par l'utilisateur avec sauvegarde ou non sur le serveur
        - Traitements légers d'analyse spatiale

- **Outils de cartographie**
  - » Deux grande librairies JS **open source:** nécessite certaines connaissances en JS mais customisation de la carte très poussée
    - **Leaflet**: https://leafletjs.com/
      - – Léger et utilisation simple
      - – Particulièrement adapté aux pages web « responsives »
      - – Très populaire: bénéficie de nombreuses extensions
    - **OpenLayers**: https://openlayers.org/
      - – Lourd et plus complexe d'utilisation (programmation « bas niveau »)
      - – Offre plus de fonctionnalités que Leaflet
  - » Outils **propriétaires**: utilisation simple (connaissance de JS souvent optionnelle) mais fonctionnalités limitées à ce que l'outil propose
    - ArcGIS web map: https://doc.arcgis.com/fr/arcgis-online/reference/what-is-web-map.htm
    - MapBox: https://www.mapbox.com/
    - GeoClip: https://www.geoclip.fr/
- **Outil d'analyse spatiale open source**
  - » **Turf.js**: librairie JS pour l'analyse spatiale côté client (ou côté serveur avec Node.js)

– Leaflet – exemple 1: un fond de carte, un marqueur et un popup



```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première carte Leaflet</title>
        <link rel="stylesheet"
        href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css" />
        <style>
          #map{
              height: 400px;
              width: 100%;
          }
        </style>
    </head>
    <body>
        <div class="map" id="map"></div>
        <script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js">
        </script>
        <script>
            var map = L.map('map').setView([51.505, -0.09], 13);
            var background= L.tileLayer(
              'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
                attribution: 'Open Street Map Contributors'
            });
            background.addTo(map);
            var marqueur=L.marker([51.5, -0.09]).addTo(map)
                .bindPopup('A pretty CSS3 popup.<br> Easily customizable.')
                .openPopup();
        </script>
    </body>
</html>
```

Importation du style CSS de Leaflet

Style CSS de l'élément HTML « map »

Définition de l'élément HTML « map » ( )

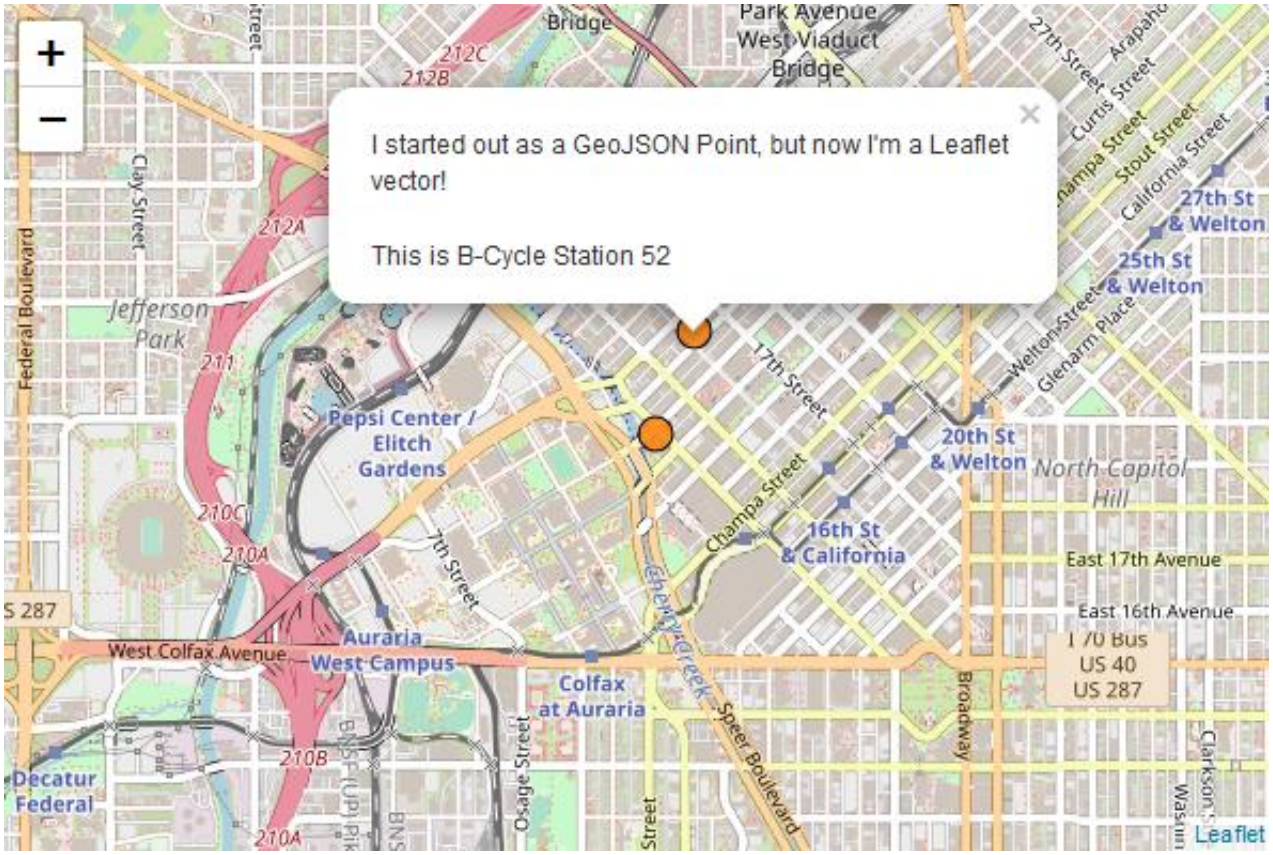
Importation de Leaflet

Construction d'un objet map (méthode de l'objet L « Leaflet ») sur l'élément HTML « map »

Définition d'une couche « fond de carte » en exploitant le WMS OSM (construction d'un objet « tileLayer » avec la méthode de l'objet L)

Définition d'un marqueur et de son popup

– Leaflet - exemple 2: Importation d'un geojson



Contenu d'une variable JSONString (chaîne de caractères) renvoyé par le serveur (résultat requête ajax)

```
{
    'type': 'FeatureCollection',
    'features': [
        {
            'geometry': {
                'type': 'Point',
                'coordinates': [
                    -104.9998241,
                    39.7471494
                ]
            },
            'type': 'Feature',
            'properties': {
                'popupContent': 'This is B-Cycle Station 51'
            },
            'id': 51
        },
        {
            'geometry': {
                'type': 'Point',
                'coordinates': [
                    -104.9983545,
                    39.7502833
                ]
            },
            'type': 'Feature',
            'properties': {
                'popupContent': 'This is B-Cycle Station 52'
            },
            'id': 52
        }
    ]
}
```

Programmation web: JavaScript

JP Kasprzyk – 2021

Code JavaScript

```
var map = L.map('map').setView([39.74739, -105], 13);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 18
}).addTo(map);

var bicyleRental = JSON.parse(JSONString);

L.geoJSON(bicycleRental, {
    onEachFeature: function(feature, layer) {
        var popupContent = "<p>I started out as a GeoJSON " +
                feature.geometry.type + ", but now I'm a Leaflet
                vector!</p>";

        if (feature.properties && feature.properties.popupContent) {
            popupContent += feature.properties.popupContent;
        }

        layer.bindPopup(popupContent);
    },

    pointToLayer: function (feature, latlng) {
        return L.circleMarker(latlng, {
            radius: 8,
            fillColor: "#ff7800",
            color: "#000",
            weight: 1,
            opacity: 1,
            fillOpacity: 0.8
        });
    }
}).addTo(map);
```

Initialisation de la carte

Ajout du fond de carte (couche WMS)

Transformation de la variable JSONString en objet JavaScript

Définition de la couche geoJSON avec :

- Traitement sur chaque entité pour ajouter un popup (propriété « onEachFeature »)
- Symbolisation des points (propriété « pointToLayer »)