Programmation en JavaScript

Pierre Pompidor

Aperçu du langage JavaScript

Langage de programmation créé en 1995 :

- Scripts exécutés par un navigateur pour manipuler les données du **DOM** (Document Object Model) du navigateur
- Accès asynchrone à des données fournies par un serveur et insertion de celles-ci dans le DOM
- Environnement serveur *Node.js* très populaire
- De nombreuses bibliothèques : cartographiques (Google Maps, OpenLayers ...), graphiques (D3.js ...), ...

Aperçu du langage JavaScript

Langage de programmation créé en 1995 :

- Scripts exécutés par un navigateur pour manipuler les données du **DOM** (Document Object Model) du navigateur
- Accès asynchrone à des données fournies par un serveur et insertion de celles-ci dans le DOM
- Environnement serveur *Node.js* très populaire
- De nombreuses bibliothèques : cartographiques (Google Maps, OpenLayers ...), graphiques (D3.js ...), ...

Confluence de deux paradigmes de programmation :

Paradigme de la **programmation fonctionnelle**
Paradigme de la **programmation par objets**

Où coder du code javascript et comment le déboguer ?

Codes JavaScript dans des fichiers d'extension **.js** :

Liés aux codes HTML dans le bloc `head` via la balise `<script>`
Ou plus généralement, n'importe où...

Où coder du code javascript et comment le déboguer ?

Codes JavaScript dans des fichiers d'extension **.js** :

Liés aux codes HTML dans le bloc `head` via la balise `<script>`
Ou plus généralement, n'importe où...

Exemple du contenu d'un fichier JS nommé *bonjour.js* :

```
console.log("Bonjour ␣!");
```

Où coder du code javascript et comment le déboguer ?

Codes JavaScript dans des fichiers d'extension **.js** :

Liés aux codes HTML dans le bloc `head` via la balise `<script>`
Ou plus généralement, n'importe où...

Exemple du contenu d'un fichier JS nommé *bonjour.js* :

```
console.log("Bonjour␣!");
```

Les messages générés par la méthode *log()* de l'objet *console*

sont affichés dans la console (outils du navigateur)

Page HTML *test.html* mettant en œuvre ce script :

Lien sur le fichier JavaScript :

```
<html>
    <head>
        <script src="bonjour.js"
                type="text/javascript"
                language="javascript">
        </script>
    </head>
    <body> JavaScript vous dit bonjour </body>
</html>
```

Page HTML *test.html* mettant en œuvre ce script :

Lien sur le fichier JavaScript :

```
<html>
    <head>
        <script src="bonjour.js"
                type="text/javascript"
                language="javascript">
        </script>
    </head>
    <body> JavaScript vous dit bonjour </body>
</html>
```

Surtout n'utilisez jamais une balise <script> auto-fermante

Mise en place d'actions suite aux actions de l'internaute

Association de **gestionnaires d'événements** aux balises/éléments HTML

- onclick : clic sur un élément
- onchange : sélection/désélection de cases à cocher, de boutons radio
- onselect : sélection d'un item d'une liste déroulante
- onmouseover : entrée du pointeur de la souris sur un élément
- onmouseout : sortie du pointeur de la souris d'un élément
- onload : chargement de la page dans le navigateur
- onunload : déchargement de la page
- ...

Appels de fonctions JavaScript pour zoomer et dézoomer la photo de James dans son CV

```
<!doctype html><html>
  <head>
     <script src="CV_James.js"></script>
  </head>

  <body>
     <header>
        <p> James Bond
              <img id="photo" src="Connery.jpg"
                   onmouseover="zoom(this)"
                   onmouseout="dezoom(this)" />
        </p>
     </header>
      ...
  </body>
</html>
```

Fonctions JavaScript pour zoomer et dézoomer la photo

```
var largeurImage;
var hauteurImage;

function zoom(image) {
    largeurImage = image.style.width;
    hauteurImage = image.style.height;
    image.style.width = "auto";
    image.style.height = "auto";
}

function dezoom(image) {
    image.style.width = largeurImage;
    image.style.height = hauteurImage;
}
```

Commentaires sur les fonctions JavaScript

La fonction *zoom()* est invoquée par le gestionnaire d'événements *onmouseover*

Une fonction JavaScript est déclarée par le mot réservé *function*
Les instructions de la fonction sont circonscrites par ses accolades

Commentaires sur les fonctions JavaScript

La fonction *zoom()* est invoquée par le gestionnaire d'événements *onmouseover*

Une fonction JavaScript est déclarée par le mot réservé *function*
Les instructions de la fonction sont circonscrites par ses accolades

Le paramètre *image* contient l'adresse en mémoire d'un objet cet objet est un élément du DOM

Un objet implémente une liste de duos clefs/valeurs
Ici l'objet contient les différents attributs (ou propriétés) de l'image

Commentaires sur les fonctions JavaScript

La fonction *zoom()* est invoquée par le gestionnaire d'événements *onmouseover*

Une fonction JavaScript est déclarée par le mot réservé *function*
Les instructions de la fonction sont circonscrites par ses accolades

Le paramètre *image* contient l'adresse en mémoire d'un objet cet objet est un élément du DOM

Un objet implémente une liste de duos clefs/valeurs
Ici l'objet contient les différents attributs (ou propriétés) de l'image

Contrairement aux autres langages objets majeurs

La programmation par objets en JavaScript ne nécessite pas la définition de classes même s'il est possible depuis 2015 d'en utiliser (normalisation ECMA6 et extensions comme TypeScript)

Rôle de JavaScript comme accesseur du DOM

Le **DOM** (Document Object Model)

représente toutes les données allouées en mémoire du navigateur (l'allocation d'une balise HTML est appelé un **élément**)

Rôle de JavaScript comme accesseur du DOM

Le **DOM** (Document Object Model)

représente toutes les données allouées en mémoire du navigateur (l'allocation d'une balise HTML est appelé un **élément**)

L'accès au DOM est essentiel

pour modifier le contenu d'une page sans passer par le serveur

Rôle de JavaScript comme accesseur du DOM

Le **DOM** (Document Object Model)

représente toutes les données allouées en mémoire du navigateur (l'allocation d'une balise HTML est appelé un **élément**)

L'accès au DOM est essentiel

pour modifier le contenu d'une page sans passer par le serveur

Des fonctions spécifiques pour sélectionner un ou plusieurs éléments du DOM :

- via le nom d'un type de balises (par exemple toutes les balises <img>)
- via un identifiant (exprimé par un attribut *id*)
- via un nom de classe CSS

Sélection d'éléments dans le DOM

Pour sélectionner des éléments du DOM, il y a deux possibilités :

- utiliser les fonctions JavaScript de l'API DOM "de base"
- utiliser une bibliothèque JavaScript de plus haut niveau

Sélection d'éléments dans le DOM

Pour sélectionner des éléments du DOM, il y a deux possibilités :

- utiliser les fonctions JavaScript de l'API DOM "de base"
- utiliser une bibliothèque JavaScript de plus haut niveau

Exemple avec l'API DOM de la sélection d'un élément par son id (document est l'objet qui désigne la page) :

```
console.log(document.getElementById("photo").src);
```

Sélection d'éléments dans le DOM

Pour sélectionner des éléments du DOM, il y a deux possibilités :

- utiliser les fonctions JavaScript de l'API DOM "de base"
- utiliser une bibliothèque JavaScript de plus haut niveau

Exemple avec l'API DOM de la sélection d'un élément par son id (document est l'objet qui désigne la page) :

```
console.log(document.getElementById("photo").src);
```

Même exemple avec la bibliothèque JQuery :

```
console.log($("#photo").attr("src"));
```

Sélection d'éléments dans le DOM - suite

Un schéma d'introspection (inspecteur DOM) avec l'API DOM :

```
function analyseDOM(noeud) {
  console.log(noeud.nodeName+"␣"+noeud.nodeValue);
  switch (noeud.nodeType) {
    case 1 : ... // noeud de type element (balise)
    case 3 : ... // noeud de type text
    ...
 }
 for (let n=0;n<noeud.attributes.length;n++) {...}
 for (let n=0;n<noeud.childNodes.length;n++) {
     analyseDOM(noeud.childNodes[n])
 }
}
```

Sélection d'éléments dans le DOM - suite

Un schéma d'introspection (inspecteur DOM) avec l'API DOM :

```
function analyseDOM(noeud) {
  console.log(noeud.nodeName+"␣"+noeud.nodeValue);
  switch (noeud.nodeType) {
    case 1 : ... // noeud de type element (balise)
    case 3 : ... // noeud de type text
    ...
 }
 for (let n=0;n<noeud.attributes.length;n++) {...}
 for (let n=0;n<noeud.childNodes.length;n++) {
     analyseDOM(noeud.childNodes[n])
 }
}
```

Appel : analyseDOM(document);

JavaScript / syntaxe de base : déclaration des variables

Variables **typées dynamiquement** et déclarées par :

Le mot réservé **var** :

la portée de la variable est (presque) globale

JavaScript / syntaxe de base : déclaration des variables

Variables **typées dynamiquement** et déclarées par :

Le mot réservé **var** :

la portée de la variable est (presque) globale

Le mot réservé **let** :

la portée de la variable est alors le bloc d'instructions

JavaScript / syntaxe de base : déclaration des variables

Variables **typées dynamiquement** et déclarées par :

Le mot réservé **var** :

la portée de la variable est (presque) globale

Le mot réservé **let** :

la portée de la variable est alors le bloc d'instructions

Le mot réservé **const** :

la variable n'est accessible qu'en lecture

Syntaxe de base : déclaration des variables

Javascript type (en interne) les variables suivant six types primitifs :

- un type **booléen** : **true** et **false** ;
- un type **nul** : **null**
- un type **indéfini** : **undefined**
- un type pour les **nombres** : **number**
- un type pour les **chaînes de caractères** : **string**
- un type pour les **symboles**

Syntaxe de base : déclaration des variables

Javascript type (en interne) les variables suivant six types primitifs :

- un type **booléen** : **true** et **false** ;
- un type **nul** : **null**
- un type **indéfini** : **undefined**
- un type pour les **nombres** : **number**
- un type pour les **chaînes de caractères** : **string**
- un type pour les **symboles**

Type **Object** dans les autres cas de figures

Syntaxe de base : déclaration des variables

Voici quelques exemples de création de variables scalaires :

```
var bizarre;  // sa valeur est : undefined
var i = 0;
let bool = true;
```

Syntaxe de base : déclaration des variables

Voici quelques exemples de création de variables scalaires :

```
var bizarre;  // sa valeur est : undefined
var i = 0;
let bool = true;
```

Exemple de création d'une chaîne de caractères :

```
let nom = "Bond";
let prenom = 'James';
```

Syntaxe de base : les structures de données usuelles

Deux structures de données sont omniprésentes

Syntaxe de base : les structures de données usuelles

Deux structures de données sont omniprésentes

Les **objets** :

Regroupement de données et des traitements qui y sont appliqués : l'utilisation d'objets permet une meilleure structuration du code

Syntaxe de base : les structures de données usuelles

Deux structures de données sont omniprésentes

Les **objets** :

Regroupement de données et des traitements qui y sont appliqués : l'utilisation d'objets permet une meilleure structuration du code

Les **listes** :

- qui correspondent à des tableaux dynamiques
- qui sont par ailleurs des objets

JavaScript / syntaxe de base : les listes

Création d'une liste :

soit en utilisant la fonction constructrice *Array()*
soit directement en employant les crochets (qui ne sont que du sucre syntaxique).

JavaScript / syntaxe de base : les listes

Création d'une liste :

soit en utilisant la fonction constructrice *Array()*
soit directement en employant les crochets (qui ne sont que du sucre syntaxique).

Exemple de création d'une liste :

```
let maListe = new Array(1, 2, 3, "partez");
let maListe = [1, 2, 3, "partez"];
```

JavaScript / syntaxe de base : les listes

Création d'une liste :

soit en utilisant la fonction constructrice *Array()*
soit directement en employant les crochets (qui ne sont que du sucre syntaxique).

Exemple de création d'une liste :

```
let maListe = new Array(1, 2, 3, "partez");
let maListe = [1, 2, 3, "partez"];
```

Nombre d'éléments d'une liste :

```
console.log(maListe.length);
```

JavaScript / syntaxe de base : les objets

Les objets ne sont pas originellement des instances de classes

contrairement à de nombreux autres langages de programmation

JavaScript / syntaxe de base : les objets

Les objets ne sont pas originellement des instances de classes

contrairement à de nombreux autres langages de programmation

Formatage de ces objets lorsque ceux-ci sont sérialisés (c'est à dire sauvegardés sous un format textuel)

appelé **JSON** (JavaScript Object Notation)

JavaScript / syntaxe de base : les objets

Les objets ne sont pas originellement des instances de classes

contrairement à de nombreux autres langages de programmation

Formatage de ces objets lorsque ceux-ci sont sérialisés (c'est à dire sauvegardés sous un format textuel)

appelé **JSON** (JavaScript Object Notation)

Gestion des classes en JavaScript :

- via la norme ECMAScript 6
- des extensions "de plus haut niveau" comme **TypeScript**

Syntaxe de base : les objets (suite)

Exemple de création d'un objet littéral :

```
let monObjet =
  {"nom": "Bond", "prenom": "James",
   "acteurs": ["Sean Connery", "George Lazenby",
               "Roger Moore", "Timothy Dalton",
               "Pierce Brosnan","Daniel Craig"],
   "films": {"1962" : "James Bond 007 contre ...",
             "1963" : "Bons baisers de Russie",
             "1964" : "Goldfinger"}
  };
```

Syntaxe de base : les objets (suite)

Exemple de création d'un objet littéral :

```
let monObjet =
  {"nom": "Bond", "prenom": "James",
   "acteurs": ["Sean Connery", "George Lazenby",
               "Roger Moore", "Timothy Dalton",
               "Pierce Brosnan","Daniel Craig"],
   "films": {"1962" : "James Bond 007 contre ...",
             "1963" : "Bons baisers de Russie",
             "1964" : "Goldfinger"}
  };
```

Quelques exemples de manipulation de cet objet :

```
console.log(monObjet.nom);          // Bond
console.log(monObjet.acteurs[0]);   // Sean Connery
console.log(monObjet.films[1964]);  // Goldfinger
```

Les objets instanciés par une fonction constructrice

```
function personne(nom, prenom) {
    this.nom = nom; this.prenom = prenom;
    this.cv = function() {
        console.log("Je␣suis␣"+this.nom
                                +"␣"+this.prenom); };
}
let bond = new personne("Bond", "James");
bond.cv();  // Je suis Bond James
```

Les objets instanciés par une fonction constructrice

```
function personne(nom, prenom) {
    this.nom = nom; this.prenom = prenom;
    this.cv = function() {
        console.log("Je␣suis␣"+this.nom
                                +"␣"+this.prenom); };
}
let bond = new personne("Bond", "James");
bond.cv();  // Je suis Bond James
```

new est l'opérateur qui alloue la mémoire

mais sans avoir de cadre défini pour cette allocation

this est la référence de l'objet en construction

Cette variable contient l'adresse mémoire de l'objet

Les blocs d'instructions et la structure conditionnelle

Blocs d'instructions circonscrits par des **accolades**

Création de blocs d'instructions par les fonctions, lesstructures conditionnelles, itératives...

Les blocs d'instructions et la structure conditionnelle

Blocs d'instructions circonscrits par des **accolades**

Création de blocs d'instructions par les fonctions, lesstructures conditionnelles, itératives...

La structure conditionnelle présente le schéma suivant :

```
if ( <expression conditionnelle> ) {
    ...
}
else { // ce bloc est facultatif
    ...
}
```

De nombreuses structures itératives

- la structure **while (...) ...**
- la structure **for (...; ...; ...) ...**
- la structure **for (... in ...) ...**
- la structure **for (... of ...) ...**
- la méthode **forEach()** s'appliquant aux listes

De nombreuses structures itératives

- la structure **while (...) ...**
- la structure **for (...; ...; ...) ...**
- la structure **for (... in ...) ...**
- la structure **for (... of ...) ...**
- la méthode **forEach()** s'appliquant aux listes

Structure for (... in ...)

```
let voitures = ["Aston␣Martin", "Ford␣Mustang", ...]
for (let i in voitures) {
     console.log(i +"␣:␣"+voitures[i]); }
```

De nombreuses structures itératives

- la structure **while (...) ...**
- la structure **for (...; ...; ...) ...**
- la structure **for (... in ...) ...**
- la structure **for (... of ...) ...**
- la méthode **forEach()** s'appliquant aux listes

Structure for (... in ...)

```
let voitures = ["Aston␣Martin", "Ford␣Mustang", ...]
for (let i in voitures) {
     console.log(i +"␣:␣"+voitures[i]); }
```

Structure for (... of ...)

```
for (let v of voitures) { console.log(v); }
```

Enrichissement du CV avec un carrousel

Faites défiler un carrousel circulaire de photos en cliquant sur la photo d'identité de James

- associer un gestionnaire d'événements *onclick* à la balise <img> pour appeler une fonction JavaScript *carrousel()*
- récupérer sur internet une photo de chaque James Bond
- créer dans le code JavaScript :
  - une variable *numImage* initialisée à 0
  - une liste contenant les noms des différentes photos (la première étant celle affichée par défaut)
- créer la fonction JavaScript *carrousel()* qui :
  - incrémente *numImage*
  - si *numImage* est égal au nombre de photos, lui redonne 0 comme valeur
  - affiche la photo d'indice *numImage* en modifiant l'attribut *src* (en utilisant la fonction ***getElementById()***)

AJAX (Asynchronous JavaScript And XML)

Chargement par la navigateur de **données** et non d'une page HTML ce qui modifie son DOM

- Ces données proviennent de fichiers (usages de plus en plus rares) ou de serveurs (notamment *Node.js*)
- Fonctionnalité maintenant ancienne de JavaScript
- Mise à jour d'une page en modifiant ou pas sa structure

AJAX (Asynchronous JavaScript And XML)

Chargement par la navigateur de **données** et non d'une page HTML ce qui modifie son DOM

- Ces données proviennent de fichiers (usages de plus en plus rares) ou de serveurs (notamment *Node.js*)
- Fonctionnalité maintenant ancienne de JavaScript
- Mise à jour d'une page en modifiant ou pas sa structure

Formatage des données

- Initialement en **XML** (Extensible Mark-up Language)
- Actuellement en **JSON** (JavaScript Objet Notation) : sérialisation d'objets JavaScript

AJAX (Asynchronous JavaScript And XML)

Chargement par la navigateur de **données** et non d'une page HTML ce qui modifie son DOM

- Ces données proviennent de fichiers (usages de plus en plus rares) ou de serveurs (notamment *Node.js*)
- Fonctionnalité maintenant ancienne de JavaScript
- Mise à jour d'une page en modifiant ou pas sa structure

Formatage des données

- Initialement en **XML** (Extensible Mark-up Language)
- Actuellement en **JSON** (JavaScript Objet Notation) : sérialisation d'objets JavaScript

Utilisation de la bibliothèque JavaScript jQuery

AJAX - exploitation de données formatées en JSON

Sérialisation d'objets JavaScript

Les données sont textualisées

Cette structure est appelée une collection

AJAX - exploitation de données formatées en JSON

Sérialisation d'objets JavaScript

Les données sont textualisées

Pour les formater, la structure la plus usuelle est la liste d'objets (la liste étant elle-même un objet) :

```
[{
   "<nom␣de␣propriete>" : <valeur de propriete>,
   "<nom␣de␣propriete>" : <valeur de propriete>,
   ...
 },
 ...
]
```

Cette structure est appelée une collection

Utilisation de jQuery pour importer des données

- importation d'une liste d'objets JSON (d'un fichier/serveur);
- parcours de la liste pour accéder à chaque objet ;
  les propriétés de l'objet courant sont intégrées dans une chaîne de caractères au format HTML :
- attachement de cette chaîne de caractères à un élément du DOM grâce à la méthode **append()**.

```
function afficherAlcools() {
 $.getJSON("<fichier.json> / <URL>",function(data){
     let html = "";
     for (let objet of data) {
         html += "<creation d'un fragment HTML>"; }
     $("<point d'attachement>").append(html);
 });
}
```

Explications de différents éléments de code

$ désigne l'objet de plus niveau de la bibliothèque JQuery

à cet objet sont attachées des fonctions (appelées méthodes)

Explications de différents éléments de code

$ désigne l'objet de plus niveau de la bibliothèque JQuery

à cet objet sont attachées des fonctions (appelées méthodes)

$.getJSON(...)

Méthode qui permet d'importer des données JSON et de les allouer en mémoire. Ces données sont accessibles par le paramètre (*data*) de la fonction de *rappel* (ou fonction de *callback* asynchrone)

Explications de différents éléments de code

$ désigne l'objet de plus niveau de la bibliothèque JQuery

à cet objet sont attachées des fonctions (appelées méthodes)

$.getJSON(...)

Méthode qui permet d'importer des données JSON et de les allouer en mémoire. Ces données sont accessibles par le paramètre (*data*) de la fonction de *rappel* (ou fonction de *callback* asynchrone)

$.each(...) peut remplacer la boucle for

Méthode qui permet de parcourir une liste et en extraire chaque objet (variable *objet*) (*index* indice chaque objet à partir de 0)

Explications de différents éléments de code

$ désigne l'objet de plus niveau de la bibliothèque JQuery

à cet objet sont attachées des fonctions (appelées méthodes)

$.getJSON(...)

Méthode qui permet d'importer des données JSON et de les allouer en mémoire. Ces données sont accessibles par le paramètre (*data*) de la fonction de *rappel* (ou fonction de *callback* asynchrone)

$.each(...) peut remplacer la boucle for

Méthode qui permet de parcourir une liste et en extraire chaque objet (variable *objet*) (*index* indice chaque objet à partir de 0)

jQuery fait usage de fonctions de rappel qui délivrent les données mises à disposition

fonctions anonymes (lambda fonctions) passées en paramètres

Focus sur les fonctions de rappel (suite)

Création d'une liste

```
let liste = [1, 2, 3, "partez !"];
```

Focus sur les fonctions de rappel (suite)

Création d'une liste

```
let liste = [1, 2, 3, "partez␣!"];
```

Création d'une fonction myForEach() :

```
let myForEach = function(liste, callback) {
    for (let i in liste) callback(i, liste[i]);
}
```

Focus sur les fonctions de rappel (suite)

Création d'une liste

```
let liste = [1, 2, 3, "partez␣!"];
```

Création d'une fonction myForEach() :

```
let myForEach = function(liste, callback) {
    for (let i in liste) callback(i, liste[i]);
}
```

Appel de la fonction myForEach() :

```
myForEach([1, 2, 3, "partez␣!"],
          function(i, element) {
              console.log(i, ":", element);
          });
```

Focus sur les fonctions de rappel (suite)

Création d'une liste

```
let liste = [1, 2, 3, "partez !"];
let liste = new Array(1, 2, 3, "partez !");
```

Focus sur les fonctions de rappel (suite)

Création d'une liste

```
let liste = [1, 2, 3, "partez␣!"];
let liste = new Array(1, 2, 3, "partez␣!");
```

Création d'une fonction myForEach() :

```
Array.prototype.myForEach = function(callback) {
    for (let i in this) callback(i, this[i]);
}
```

Focus sur les fonctions de rappel (suite)

Création d'une liste

```
let liste = [1, 2, 3, "partez␣!"];
let liste = new Array(1, 2, 3, "partez␣!");
```

Création d'une fonction myForEach() :

```
Array.prototype.myForEach = function(callback) {
    for (let i in this) callback(i, this[i]);
}
```

Appel de la méthode myForEach() :

```
liste.myForEach(function(i, element) {
                    console.log(i, ":", element);
                });
```

AJAX - Chargement de données complémentaires au CV

Dans cet exemple, importation de données d'un fichier

En TP, nous importerons les données à partir d'un serveur

Modifions le code HTML du CV de James pour :

- lier la bibliothèque jQuery à la page HTML :
  https://code.jquery.com/
- insérer un lien hypertexte dans la rubrique "hobbies" :
  appel de la fonction JavaScript d'importation des données
- créer une balise <ul> identifiée par un identifiant :
  accroche du code HTML formé à partir des données importées
- importer les données et modifier le DOM

AJAX - Chargement de données complémentaires au CV

```
<head>
    <script src="https://code.jquery.com/jquery-3.
            crossorigin="anonymous"></script>
    <script src="CV_James_AJAX.js"></script>
</head>
<body>
  ...
 <article>
     <p> Hobbies : </p>
     <ul>
       <li> Degustation de
            <a onclick="afficherAlcools()"
               href="#">nombreux alcools</a>
            <ul id="listeAlcools"></ul>
       </li>
       <li> Conversation mondaine ... </li>
     <ul>
```

AJAX - Fichier JSON *alcools.json*

```
[{"nom":"Vodka Martini",
   "amateurs":["Sean Connery", "Roger Moore",
                "Pierce Brosnan", "Daniel Craig"]
 },
 {"nom":"Vesper Martini","amateurs":["Daniel Craig"]
 {"nom":"Rum Collins", "amateurs":["Sean Connery"]}
 {"nom":"Mint Julep", "amateurs":["Sean Connery"]},
 {"nom":"Whisky The Macallan", "experts":["Craig"]}
]
```

Liste (encadrée par des crochets) contenant des objets (encadrés par des accolades)

AJAX : intégration des données dans le DOM

Exemple de fonction JavaScript utilisant du code jQuery :

```
function afficherAlcools() {
    $.getJSON("alcools.json", function(data) {
        let html = "";
        for (let objet of data) {
            html += "<li>"+objet.nom+"</li>";
        }
        $("#listeAlcools").append(html);
    });
}
```

AJAX : intégration des données dans le DOM

Exemple de fonction JavaScript utilisant du code jQuery :

```
function afficherAlcools() {
    $.getJSON("alcools.json", function(data) {
        let html = "";
        for (let objet of data) {
            html += "<li>"+objet.nom+"</li>";
        }
        $("#listeAlcools").append(html);
    });
}
```

La chaîne de caractères *html* de valeur finale :

```
<li>Vodka Martini</li><li>Vesper Martini</li>...
```

AJAX - résultat après sélection du lien

CV de James Bond avec AJAX - Mozilla Firefox

CV de James Bond avec AJA

file:///home/fongus/DUI/WEB/CV_avec_AJAX/CV_James_AJAX.html#

James Bond

Nationalité britannique
Profession : agent secret (dernier employeur : MI6)

Expérience professionnelle :

Année	Nature de la mission
1954	Neutralisation d'un dangereux amnésique (le Chiffre)
1961	Prévention d'une catastrophe nucléaire
1962	Démantèlement d'une organisation criminelle
...	...

Compétences :

- Permis B (avec certificat d'aptitude à la conduite rapide)
- Langues étrangères : russe, japonais, danois (notions)

Hobbies :

- Dégustation de nombreux alcools
  - Vodka Martini
  - Vesper Martini
  - Rum Collins
  - Mint Julep
- Conversation mondaine en présence de public féminin

Attention à l'asynchronisme !

Les fonctions de callback conditionnées par l'accès à des ressources sont **asynchrones**

Attention à l'asynchronisme !

Les fonctions de callback conditionnées par l'accès à des ressources sont **asynchrones**

Qu'affiche console.log() dans cet exemple ?

```
$.getJSON("<appel␣serveur>", function(data) {
  let html = "";
  for (let type of data){
    $.getJSON("<appel␣serveur>"+type,
        function(data2) {
          for (let objet of data2)
              html += objet.nom+"␣";
    });
  }
  console.log(html);
});
```

Serveur Node.js : idéal pour délivrer des données JSON

Node.js est un interpréteur JS côté serveur et orienté réseau

Serveur Node.js : idéal pour délivrer des données JSON

Node.js est un interpréteur JS côté serveur et orienté réseau

Mise en œuvre de services web grâce au framework express

```
var express = require("express");
var fs = require('fs');
var app = express();
...
app.listen(8888);

app.get('/', function(request, response) {
    let text = fs.readFileSync('<fichier␣JSON>');
    response.end(text);
});
```

Appel du serveur : localhost:8888/

Pour exécuter du code JavaScript seulement après que les éléments initiaux aient été instanciés dans le DOM :

La old school :

```
<body onload="fonctionJavaScript()">
```

Pour exécuter du code JavaScript seulement après que les éléments initiaux aient été instanciés dans le DOM :

La old school :

```
<body onload="fonctionJavaScript()">
```

En utilisant jQuery et le sélecteur sur le document $()

```
$().ready(function(){
    ...
});
```

Pour exécuter du code JavaScript seulement après que les éléments initiaux aient été instanciés dans le DOM :

La old school :

```
<body onload="fonctionJavaScript()">
```

En utilisant jQuery et le sélecteur sur le document $()

```
$().ready(function(){
    ...
});
```

En plaçant une balise <script> après le </body> !

Widgets graphiques - bibliothèque jQuery-ui

Une bibliothèque à dézipper dans votre dossier de travail

```
<script src="jquery-ui-1.12.1/jquery-ui.min.js">
</script>
<link rel="stylesheet"
      href="jquery-ui-1.12.1/jquery-ui.min.css">
</link>
```

Widgets graphiques - bibliothèque jQuery-ui

Une bibliothèque à dézipper dans votre dossier de travail

```
<script src="jquery-ui-1.12.1/jquery-ui.min.js">
</script>
<link rel="stylesheet"
      href="jquery-ui-1.12.1/jquery-ui.min.css">
</link>
```

Offre de nombreux composants graphiques

Menus déroulants, onglets, accordéons...

Widgets graphiques - bibliothèque jQuery-ui

Une bibliothèque à dézipper dans votre dossier de travail

```
<script src="jquery-ui-1.12.1/jquery-ui.min.js">
</script>
<link rel="stylesheet"
      href="jquery-ui-1.12.1/jquery-ui.min.css">
</link>
```

Offre de nombreux composants graphiques

Menus déroulants, onglets, accordéons...

De nombreuses autres bibliothèques de composants graphiques

jQuery-ui / mise en œuvre d'un accordéon

Création de code HTML (ou par insertions dans le DOM)

```
<div id="points_interet">
    <h3> bar </h3>
    <div id='bar'>
        <input type='checkbox'> resto1 </input>
        ...
    </div>
    <h3> restaurant </h3>
    <div id='restaurant'>
        <input type='checkbox'> hotel1 </input> ...
    </div>
</div>
```

Abracadabra !

```
$('#points_interet').accordion({ ... });
```

Bibliothèques cartographiques poir l'ffichage de cartes

Google Maps

- Google fournisseur de la bibliothèque et des fonds de cartes
- programmation confortable
- beaucoup de fonctionnalités (par exemple gestion des adresses)
- payant pour une exploitation commerciale
- quota pour une exploitation non commerciale

Bibliothèques cartographiques poir l'ffichage de cartes

Google Maps

- Google fournisseur de la bibliothèque et des fonds de cartes
- programmation confortable
- beaucoup de fonctionnalités (par exemple gestion des adresses)
- payant pour une exploitation commerciale
- quota pour une exploitation non commerciale

OpenLayers

- fournisseur de fonds de cartes libre :
  mais fréquemment associé à *OpenStreetMap*
- possibilité de créer des serveurs cartographiques indépendants

Affichage d'une carte avec OpenLayers/OpenStreetMap

Liens sur un cdn (content delivery network)

```
<script
  src="https://cdn.rawgit.com/openlayers/.../ol.js">
</script>
<link  href="https://cdn.rawgit.com/.../ol.css"  .../>
```

Affichage d'une carte avec OpenLayers/OpenStreetMap

Liens sur un cdn (content delivery network)

```
<script
  src="https://cdn.rawgit.com/openlayers/.../ol.js">
</script>
<link  href="https://cdn.rawgit.com/.../ol.css"  .../>
```

Utilisation de la classe Map

```
var map = new ol.Map({
  target: 'map',
  layers: [new ol.layer.Tile({source:
                              new ol.source.OSM()})],
  view: new ol.View({
        center: ol.proj.fromLonLat([3.87, 43.61]),
        zoom: 12
  })
```

OpenLayers / affichage de marqueurs

OpenLayers / affichage de marqueurs

Clonage d'une image existante

```
let image = $("#markerProto").clone();
image.attr("id", "marker1");
$("body").append(image);
```

OpenLayers / affichage de marqueurs

Clonage d'une image existante

```
let image = $("#markerProto").clone();
image.attr("id", "marker1");
$("body").append(image);
```

Utilisation de la classe Overlay

```
let marker = new ol.Overlay({
    position: ol.proj.fromLonLat([long, lat]),
    positioning: 'center-center',
    element: document.getElementById("marker1")
});
map.addOverlay(markers[numPi]);
```

OpenLayers / affichage d'informations liées au marqueur

Création d'une division identifiée (il y a d'autres possibilités)

OpenLayers / affichage d'informations liées au marqueur

Création d'une division identifiée (il y a d'autres possibilités)

Comme précédemment, utilisation de la classe Overlay

```
let popup = new ol.Overlay({
    position: ol.proj.fromLonLat([long, lat]),
    positioning: 'center-center',
    offset: [<en abscisse>, <en ordonnee>],
    element: document.getElementById("div1")
});
map.addOverlay(popup);
```

OpenLayers / affichage d'informations liées au marqueur

Création d'une division identifiée (il y a d'autres possibilités)

Comme précédemment, utilisation de la classe Overlay

```
let popup = new ol.Overlay({
    position: ol.proj.fromLonLat([long, lat]),
    positioning: 'center-center',
    offset: [<en abscisse>, <en ordonnee>],
    element: document.getElementById("div1")
});
map.addOverlay(popup);
```

Bien entendu les marqueurs comme les popups devraient être créés dans des collections !

Affichage de la popup sur la sélection du marqueur

Affichage de la popup sur la sélection du marqueur

Mise en œuvre d'écouteurs d'événements

```
$('body').on("click", "img", function() {
  console.log('click␣sur␣'+$(this).attr('id'));
  let arr
      = $(this).attr('id').match(/marker(.*)/);
  let popup
      = document.getElementById("div"+arr[1]);
  (popup.style.display == "none" ?
                    popup.style.display = "block" :
                    popup.style.display = "none")
});
```

Affichage de la popup sur la sélection du marqueur

Mise en œuvre d'écouteurs d'événements

```
$('body').on("click", "img", function() {
  console.log('click␣sur␣'+$(this).attr('id'));
  let arr
      = $(this).attr('id').match(/marker(.*)/);
  let popup
      = document.getElementById("div"+arr[1]);
  (popup.style.display == "none" ?
                    popup.style.display = "block" :
                    popup.style.display = "none")
});
```

Vous remarquerez l'utlisation d'une regexp