Chapitre 2
Architecture client/serveur
Bases du JavaScript

Plan

- Architecture générale
- Division client/serveur
- JavaScript : types de base et structures de données
- JavaScript : opérations de base
- JavaScript de présentation
- JSON

Distinction importante!

Site web / Page web :

«Ensemble de page(s) pouvant être consulté(es) en suivant des hyperliens.»

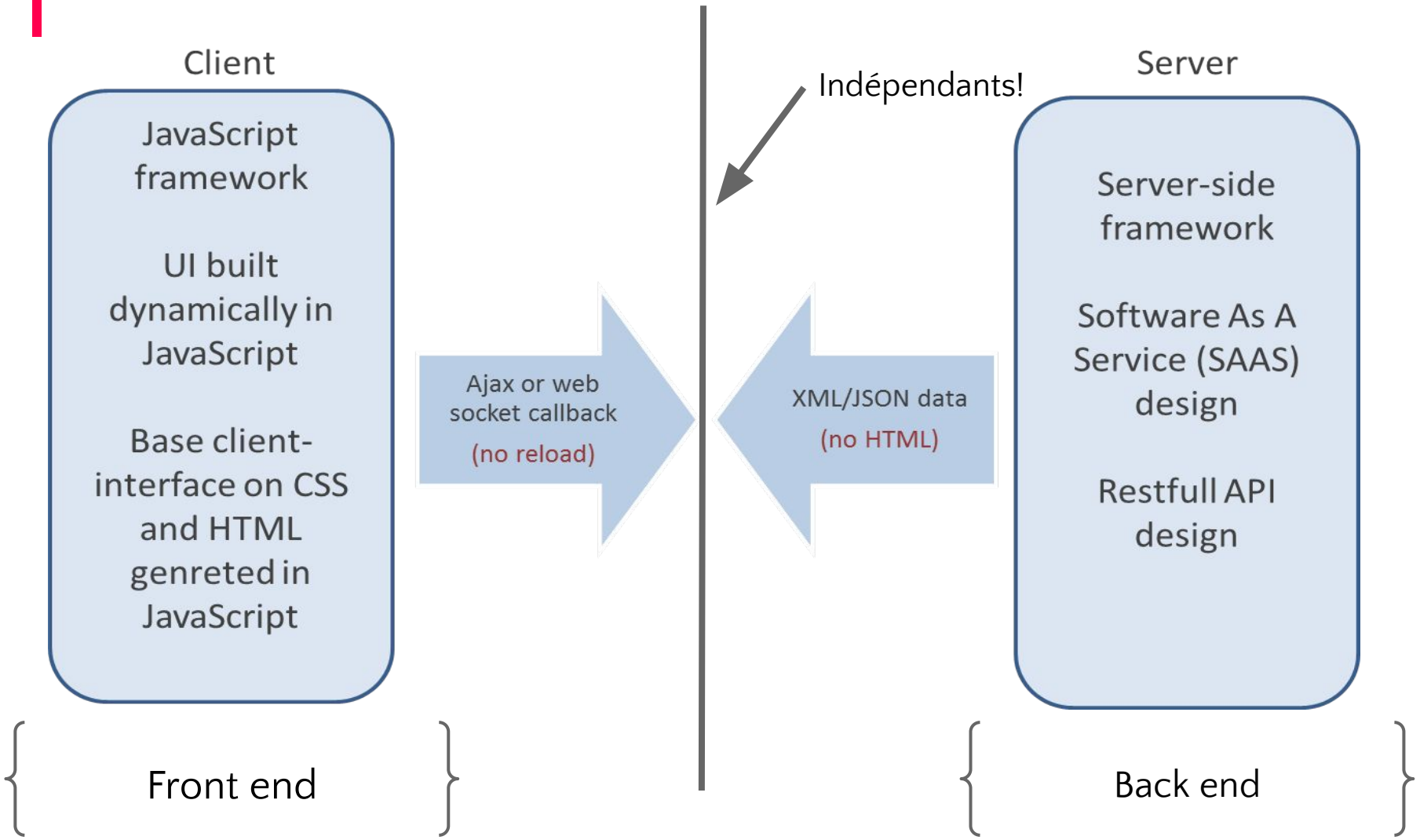
- Construit en HTML, CSS etc.

Application web :

«Logiciel applicatif manipulé grâce à un navigateur Web»

-> C'est ce sur quoi nous allons focuser.

Architecture générale



Architecture générale

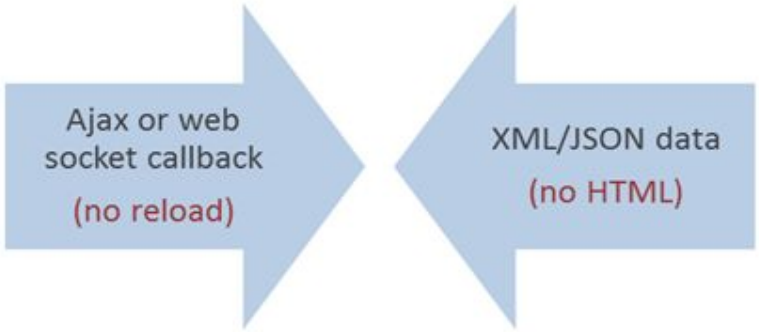
- Client léger/lourd, selon le type d'application
- Pas de logique DE DOMAINE côté client : logique d'affichage/traitement des données seulement!
- Division **client** : Logique de présentation séparée du traitement des données ou des appels à l'API.

Architecture générale

- Répond aux demandes du client
- A un domaine bien séparé du client, jamais celui-ci ne doit accéder à des objets du serveur!
- Design permettant de supporter n'importe quel client : navigateur ou autre application
- Peut contenir une base de données ou pas : ce n'est pas sensé paraître !

Architecture générale

- Requêtes asynchrones
- Manipule les données fournies par le serveur et lui retourne



- Transforme les objets client en objets du domaine et vice-versa

{ Front end } { Back end }

Client

Voyons le client en profondeur...

Langages interprétés

Langages qui seront interprétés par le navigateur :

Pas exécuté directement par la machine (comparativement aux langages compilés), mais par un autre programme nommé interprète.

Interprète = navigateur dans le cas du Web.

JavaScript: Nous commencerons par les rudiments du language, pour ensuite exclusivement focuser sur l'**ES6**.

ES6

JavaScript

- Inventé en 1995
- Implémentation de l'ECMAScript (Langage standardisé par l'ECMA, implémenté par le JavaScript, ActionScript et même C++, langage orienté objet)
- Langage destiné à être exécuté par le client
  - (Depuis 2009, langage serveur avec **node.js**, à voir dans le chapitre 7)
  - Alternatives :
    - Google Dart
    - CoffeeScript
    - Microsoft TypeScript
    - Google Web Toolkit (GWT)
    - Autres

JavaScript

Pourquoi voir le **JavaScript**?

- C’est le ‘code’ de votre application web. Ajoute tout le dynamisme dans vos pages.
- Permet la communication client/serveur, notamment pour l’envoi/réception de données.
- Est devenu un standard incontesté dans le domaine du web. S’assurer que tout le monde est sur la même longueur d’onde...
- Applications isomorphiques en vogue, JavaScript autant client que serveur.

JavaScript

Note importante!

Les exemples suivants sont réalisés en **ES5** - bien qu'étant encore le standard universel, il est de plus en plus *deprecated* au profit de l'**ES6** et de ses successeurs.

L'important ici est de comprendre les concepts et de ne pas s'attarder à la syntaxe.

JavaScript : types

Le JavaScript est un langage très faiblement **typé.**

Toutes les variables utilisent le mot clé '**var**'.

```
var toto1 = 1;
var toto2 = "thisIsAString";
var toto3 = new Array();
var toto4 = {};
```

⇒ Ne jamais oublier que le langage est **interprété**!

À vous de savoir ce que vous manipulez!

JavaScript : types

Le JavaScript est un langage très faiblement **typé.**

Comment connaître le type? Utilisez l'opérateur **typeof.**

```
var toto1 = 1;


alert(typeof toto1); // Affiche "number".
```

Il existe les types ***null*** et ***undefined*** en JavaScript.

Utilisation :

```
if (typeof toto1 !== 'undefined') {
    // Pourquoi le faire ainsi?
}
```

JavaScript : opérateurs

Les **opérateurs** sont pratiquement les mêmes que les autres langages.

Petites particularités :

`== vs ===`

`!= vs !==`

```
1 == '1' // Retourne true

1 === '1' // Retourne false
```

La triple égalité prend en compte le **type** des variables

⇒ Utilisée dans la grande majorité des cas!

JavaScript : types

Les **types de base** offrent plusieurs méthodes. Nous verrons les plus courants, à vous de lire pour découvrir les autres!

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux

1. String

```
var text = “Bonjour!”;

text.charAt(1); // Renvoie ‘o’.

text.toLowerCase(); // Renvoie la chaîne “bonjour!”.

text.substring(1,4); // Renvoie la chaîne “onj”.

text.trim(); // Enlèverait les espaces au début/à la fin.

etc… voir
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/String
```

JavaScript : types

2. Date

```
var date = new Date(); // Équivalent à Date.now().

var anniversary = new Date(1995, 11, 17);

anniversary.setYear(1996); // Changeons l'annnée...

anniversary.getDate(); // Renvoie le jour du mois (1-31), donc 17.

anniversary.getDay(); // Renvoie le jour de la semaine (0-6), donc 0.
```

Plusieurs méthodes de disponibles, à vous de les expérimenter…
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date

JavaScript : types

3. Array

⇒ Seule structure de donnée de base en JavaScript, le tableau!

```
var array = []; // Peut également s'écrire new Array();

var years = [1950, 1960, 1970];

years.pop(); // Renvoie 1970 (dernière valeur) et l'enlève du tableau.

years.push(1970); // Ajoute la valeur 1970 à la fin du tableau.

var test = years[0]; // Renvoie la valeur à l'indice 0, donc 1950.

var test2 = years.length; // Renvoie le # d'éléments, donc 3.

years.push("une string"); // Acceptable? Pourquoi?
```

JavaScript : types

3. Array : plusieurs méthodes

```
var years = [1950, 1960, 1970];

years.reverse(); // Donne [1970, 1960, 1950].

years.sort(); // Donne [1950, 1960, 1970].

years.splice(0,1); // Donne [1950].

years.forEach(function(year) {
    console.log(year);
});
```

Plusieurs méthodes de disponibles, à vous de les expérimenter…

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array

JavaScript : fonctions

Le JavaScript permet évidemment de déclarer des fonctions.

```
function toto1() {
    return “hello”;
}

var toto1 = function() {
    ...
}
```

Remarquez qu’il n’y pas de **type de retour**. Il est donc impossible de savoir si une fonction retourne quelque chose ou non, il faut regarder le code plus attentivement…

JavaScript : fonctions

Exemple 2 : Remarquez qu'il est possible d'obtenir une variable qui consiste en **un pointeur** sur **la fonction**.

⇒ Encore plus important de connaître ce que vous manipulez !

En effet, la seule variable

```
var toto1;
```

pourrait être une string, un array etc.

ou une fonction qui s'appelle ainsi :

```
toto1();
```

JavaScript : *debugging*

TRÈS TRÈS IMPORTANT!

Apprenez à utiliser les fonctionnalités de ***debugging*** offertes par le JavaScript!

ENCORE PLUS IMPORTANT :

La touche **F12** ouvre les outils de développement de votre navigateur. Utilisez-les! Vous pourrez facilement expérimenter et débugger!

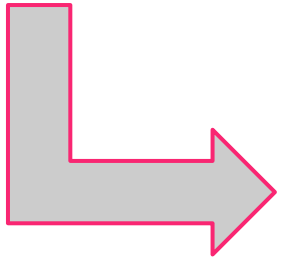
```
alert(“test!”);
```

JavaScript : *debugging*

```
console.log('test!');
```

```
console.log("test!");
test!
```

```
debugger;
```



Lorsque la console de votre navigateur est ouverte, permet de lancer un ***breakpoint***.

Autrement dit, le navigateur va pauser l'exécution du code JavaScript sur cette ligne précise, vous permettant de tester ou de voir les valeurs des variables locales par exemple.

JavaScript : opérations de base

Le JavaScript permet évidemment d'utiliser des opérations telles que les **for** et **while**.

```
for (var i = 0; i < myArray.length; i++) {
    console.log(myArray[i]);
}
```

Qu'en est-il du **for in**?

"Cette instruction effectue, dans un ordre arbitraire, une boucle sur les propriétés énumérables d'un objet".

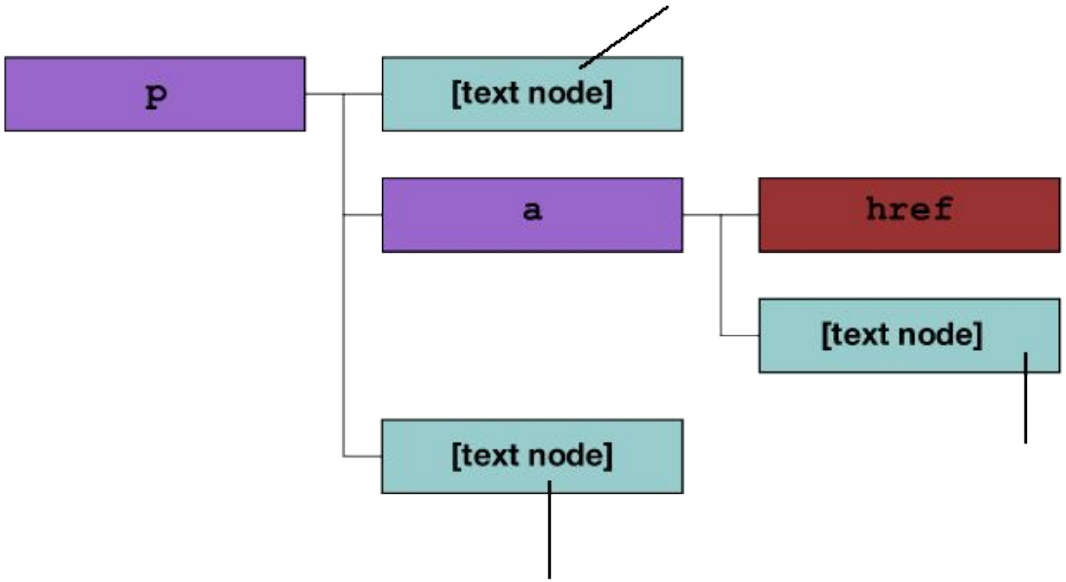
Nous verrons les objets très bientôt… et également une autre sorte de *loop* ES6 beaucoup plus pratique…

JavaScript de présentation

Utilise le DOM : Document Object Model

- Représentation objet du document.

Ex : Hi! click



JavaScript de présentation

Le DOM permet de :

- Accéder à n'importe quel élément dans la page et manipuler son style, son contenu et ses attributs
- Créer des nouveaux éléments, leur donner du contenu et les insérer dans la page seulement lorsqu'ils sont nécessaires.

Le JavaScript de présentation devrait être séparé du reste, il n’a pas le même but!

JavaScript de présentation

Les deux principaux éléments du DOM à savoir manipuler sont :

window : Représente la fenêtre du navigateur et tout son contenu. Tout agit sur cet élément. Par exemple, **alert()** est en réalité **window.alert();**.

document (ou window.document) : Le document représente le contenu HTML de la page courante. Il permet de le lire ainsi que de le modifier.

JavaScript de présentation

Méthodes intéressantes de **document** :

```
// Retourne l'élément html identifié par myId, ou undefined.
document.getElementById('myId');

// Retourne un array des éléments ayant l'attribut name défini //
à myName.
document.getElementsByName('myName');

// Retourne un array des éléments ayant la classe myClassName.
document.getElementsByClassName('myClassName');
```

JavaScript de présentation

Les éléments HTML sont de type **HTMLElement**.

https://developer.mozilla.org/fr/docs/Web/API/HTMLElement

⇒ Un élément HTML possède énormément de méthodes et de propriétés, permettant notamment de :

- Lire/Modifier ses **attributs**.
- Lire/Modifier son id/ses classes CSS.
- Lire/Modifier le texte ou son *inner* HTML.
- Lui attacher des **événements**.

JavaScript de présentation

Exemple très simple :

```
> myElement.textContent = "Accepter ce message"
< "Accepter ce message"
> myElement
< <button id="dismiss-shortcuts-btn" type="button" tabindex="-1">Accepter ce message</button>
```

Exemple un peu plus complexe :

```
> myElement.onclick = function() {
      alert("You clicked me!");
  }
< function () {
      alert("You clicked me!");
  }
```

Le Javascript possède une logique **événementielle**. Ici par exemple, lorsque l'événement *onclick* est lancé par le navigateur, la fonction définie ci-haut sera alors **appelée.**

JavaScript de présentation

```
element.click();
```

```
element.onclick = function() {
    // Do something
};
```

L'événement est lancé, par exemple lors d'une action de l'utilisateur.

L'événement est attrapé par le *event handler*, et la fonction y correspondant est exécutée.

JavaScript : divers

Outre les opérations de base, le langage est fait pour être utilisé dans une logique majoritairement asynchrone/événementielle.

"*On doit embrasser l'asynchrone, et non pas lutter contre*'

- Les appels au serveur ne seront jamais bloquants…
- La majorité de votre logique réagira à des événements, le flot est très loin d'être séquentiel…

JavaScript : divers

Il est important de savoir que les requêtes Web ne répondent pas immédiatement : on doit donc prévoir un mécanisme de callbacks!

Callback : Fonction de retour pouvant être appelée lorsqu'une opération est terminée.

Change donc drastiquement le fil d'exécution du code.

Votre application ne doit jamais ***geler***.

JavaScript : divers

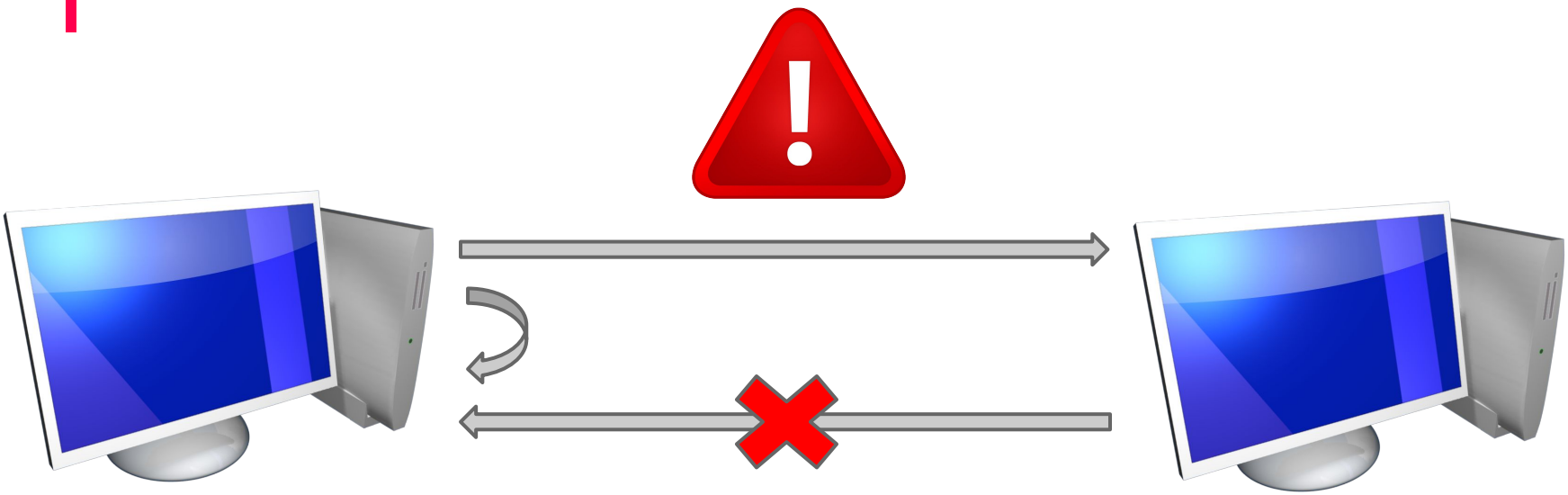
Exemple simple:

```
function doSomething(callback) {
    // ...
    // Call the callback
    callback(‘hello!');
}

function foo(message) {
    // I'm the callback
    alert (message);
}

doSomething(foo);
```

Un petit mot sur l’asynchrone…

Si tout se passe bien, avec la rapidité d'aujourd'hui, on ne verra 'presque' aucune différence.

Un petit mot sur l’asynchrone…



Si le serveur ne réponds pas, que se passe-t-il ?

Et l'interface utilisateur, elle?

Dans des cas limites, jouez sur les animations ou le texte afin d’inviter l’utilisateur à patienter. (quand le contenu complet de la page dépend de la requête par exemple)

JavaScript : divers

Portée des variables

Attention aux variables **globales** : TOUT est partagé entre les différents fichiers, seulement l'ordre importe!

Toute variable déclarée dans une fonction, peu importe où, est accessible partout dans la fonction.

JavaScript : divers

Portée des variables : exemple

```
function test() {

     var a;

     for (var i = 0; i < 10; i++) {
          var b = i * 10;
          c = 12;
     }

     return [a, b, c];
}
```

Est-ce que ça fonctionne?

JavaScript : divers

Portée des variables : exemple

```
function test() {

     var a;

     for (var i = 0; i < 10; i++) {
          var b = i * 10;
          c = 12;
     }

     return [a, b, c];
}
```

Sans déclaration, on réfère au 'global object', soit (window) ou **this**.

JavaScript : divers

Closures

Fonctionnalité méconnue mais puissante : possibilité, pour une fonction, d'accéder à des variables qui ne sont plus à sa portée :

```
function adder(number) {
    function add(value) {
        return number + value;
    }
    return add;
}

var add10 = adder(10);
add10(1); // ---> retourne quoi?
```

JavaScript : divers

Pratique

setInterval/setTimeout : Timer intégré dans JavaScript pour répéter des événements.

```
setInterval(function() { interval() }, 1000 );

function interval() {
    var date = new Date();
    var dateString = date.toLocaleTimeString();
    document.getElementById("time").value = dateString;
}
```

JSON: JavaScript Object Notation

Le JSON est une façon simple de représenter des objets en JavaScript. Les objets JSON sont simplement des paires de clés/valeurs, utilisées autant du côté client que sur le fil.

Un objet JSON peut se déclarer ainsi :

```
var myObject = {}; // Objet vide. Correspond à new Object();

// Déclare un objet ayant une clé name dont la valeur est John.
// Remarquez les doubles guillemets.
var myObject2 = { “name” : “John” };


// Pas exactement la même syntaxe qu’un objet JavaScript pur!
var myJavascriptObject = { name : ‘John’ }
```

JSON: JavaScript Object Notation

JSON un peu plus complexe :

```
var contact = {
     "firstName": "John",
     "lastName": "Smith",
     "address": {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": 10021
     },
     "phoneNumbers": [
         "212 555-1234",
         "646 555-4567"
     ]
 }
```

JSON: JavaScript Object Notation

```
myObject2.name; // Que renvoie ce code?

myObject2[“name”]; // Et celui-ci?
```

L’accès aux propriétés des éléments JSON peut se faire directement, soit en utilisant leur représentation en chaîne de caractères.

Si la propriété n’existe pas, elle est automatiquement créée.

```
var myObject = {
    name : “John”
}
myObject.lastName; // Retourne undefined
myObject.lastName = “Smith” // Ajoute la propriété lastName
```

JSON: JavaScript Object Notation

JSON : Méthodes utiles

JSON.parse()
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/JSON/parse

⇒ Transforme une **string** JSON en un objet. Très utile lors du **retour** d'une **requête** vers le serveur.

JSON.stringify()
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/JSON/stringify

⇒ Transforme un objet JSON en sa représentation **string**. Très utile lors de l'**envoi** d'une **requête** vers le serveur.