Client-side Scripting

Enseignante: Samia Saidi Ouerghi
samiasaidi@gmail.com

Javascript?

- **JavaScript** (**JS**) is a lightweight interpreted programming language with first-class functions (functions are treated as variables: they can be passed as arguments to other functions, returned as the values from other functions, and assigned to variables etc.)

- While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, e.g., Adobe Acrobat.

- JavaScript supports object-oriented, and procedural programming.
- JavaScript® is a registered trademark of Oracle in the U.S. and other countries.

Caractéristiques de JS

- C'est un langage de scripts qui incorporé aux balises Html, permet d'améliorer la présentation et l'interactivité des pages Web en les rendant dynamiques.
- Pour faire simple, un script est, par opposition à un langage compilé, **un langage qui s'interprète**. Ici, l'interprète du JavaScript, c'est le navigateur du visiteur (le client).
- Ces scripts vont être **gérés et exécutés par le browser** lui-même SANS devoir faire appel aux ressources du serveur.
- JS est un l**angage script orienté objets**, ce qui signifie que vous pouvez créer des classes et instancier des objets.
- JS propose en standard un certains nombres de classes que vous pourrez utiliser à votre aise pour créer vos programmes.

Histoire de JavaScript

- Javascript a été initialement inventé par Brendan Eich en 1995 et développé par Netscape et s'appelait alors **LiveScript**.
- Il est inspiré de nombreux langages, notamment de Java, C et C++.
- Les versions récentes du langage JavaScript ont pour origine les spécifications de la norme ECMA-262 définissant ECMAScript :
  - Version 1.0 (March 1996) supporté par IE et Netscape.
  - Puis les Versions 1.1 (August 1996), 1.2, 1.3, et 1.4 .
  - Version 1.5 (November 2000) supportée par IE, Netscape, Firefox, Opera, Safari et chrome.
  - Version 1.8.5 (2010) (fully supported by all modern browsers)
  - ECMAScript 2018 (Draft)

Les implémentations de Javascript

- Les implémentations actuelles de Javascript sont multiples (exemples) :
  - Safari utilise SquirrelFish JavaScript engine
  - Chrome utilise V8 JavaScript engine (Google's open source JavaScript engine écrit en C++)
  - Mozilla utilise SpiderMonkey JavaScript engine C /C++
  - Etc.
- Par soucis de compatibilité, et pour garantir que votre code satisfait la propriété « cross-browsing », il y a un test qui a été mis en place:
- Le Test ACID3 (http://acid3.acidtests.org/) a été défini par Web standards c'est un score sur 100 indiquant si votre navigateur permet de développer des applications web 2.0 dynamique principalement tester le support de JS,

ACID3 Test

- DOM2 Core
- DOM2 Events
- DOM2 HTML
- DOM2 Range
- DOM2 Style (getComputedStyle, …)
- DOM2 Traversal (NodeIterator, TreeWalker)
- DOM2 Views (defaultView)
- ECMAScript
- HTML4 (<object>, <iframe>, …)
- HTTP (Content-Type, 404, …)
- Media Queries
- Selectors (:lang, :nth-child(), combinators, dynamic changes, …)
- XHTML 1.0
- CSS2 (@font-face)
- CSS2.1 (‘inline-block’, ‘pre-wrap’, parsing…)
- CSS3 Color (rgba(), hsla(), …)
- CSS3 UI (‘cursor’)
- data: URIs
- SVG (SVG Animation, SVG Fonts, …)

Remarque : il y a aussi ACID1 (HTML4+CSS1) et ACID2 (CSS1+CSS2)

What is JavaScript Framework?

- En écrivant du code Javascript, ils faut prendre en considération cette diversité dans les moteurs d'exécution .
- Très complexe à faire
- Solution : une diversité de librairie (Framework) ont vu le jour
  - Un Framework JavaScript est un ensemble d'applications et de fonctionnalités regroupés Interface de Programmation Applicative (API) : boite à outils qui contient toutes les tâches répétitives et courantes.
  - Un Framework permet de gagner beaucoup de temps dans vos développements, facilite la manipulation de DOM, faire des requêtes AJAX, écrire des applications plus robustes et plus riches en fonctionnalités.
  - Un Framework permet d'alléger le développeur des taches récurrentes pour se concentrer sur des taches plus complexes : Il n'est plus nécessaire de faire le développement de zéro.

What is JavaScript Framework?

- Principal apport des Frameworks JS : *Cross-browsing* (garantir la compatibilité du code JS avec divers navigateurs)
- La diversité des plateformes Web fait qu'il est difficile de garantir que votre code est compatible avec tout plateforme (cross-browsing property)
  - Environnement PC : Windows, MacOS, Linux
  - Environnements mobile : iPhone, Android, Blackberry, S60, and JavaME/JavaFX
  - Les navigateurs : le premier s'appelait worldwideweb en 1990 (l'ancien Nexus), suivi par Mosaic en 1993 puis IE en 1995
  - Chrome, Safari, Opera, IE, Maxthon, Konqueror, Opera, etc..

- À ce jour, 500+ Frameworks ont vu le jour !!!!

Exemples de Framework JS

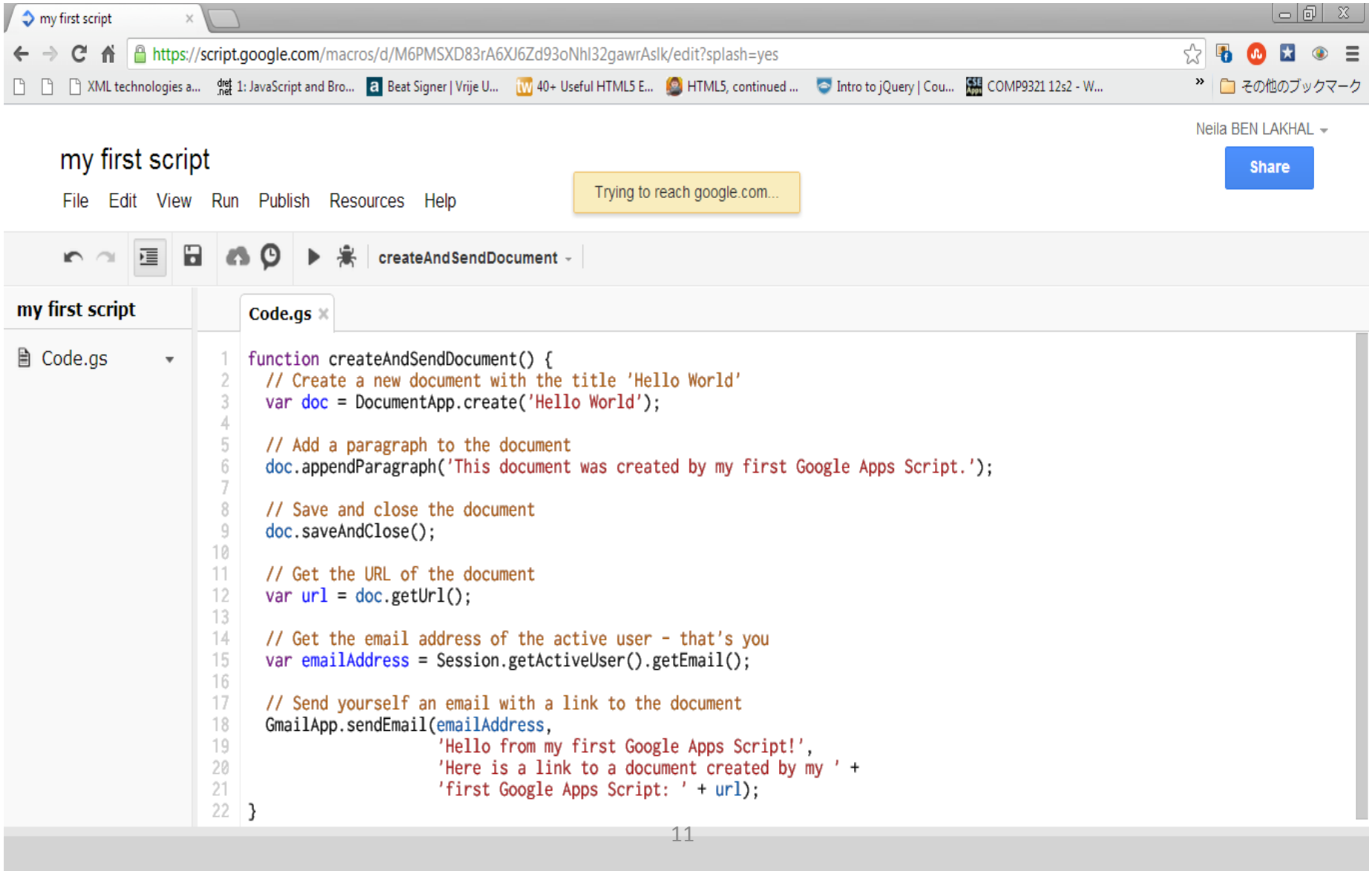
- **Prototype**: (facilite de développement d'applications Web dynamiques) http://www.prototypejs.org/
- **Script.aculo.us**: ("dédié interface riche") http://script.aculo.us/
- **Mootools**: (pour des fonctionnalités JS avancées)
- **extJS**: permet de construire des applications web interactives (initialement une extension à la bibliothèque Javascript YUI de Yahoo)
- **Angular.js** (old version), **angular** (By Google)
- **React.js** (By Facebook)
- **Node.js (**built on Chrome's V8 JavaScript engine)
- **Etc.**

Une liste exhaustive : https://www.javascripting.com/

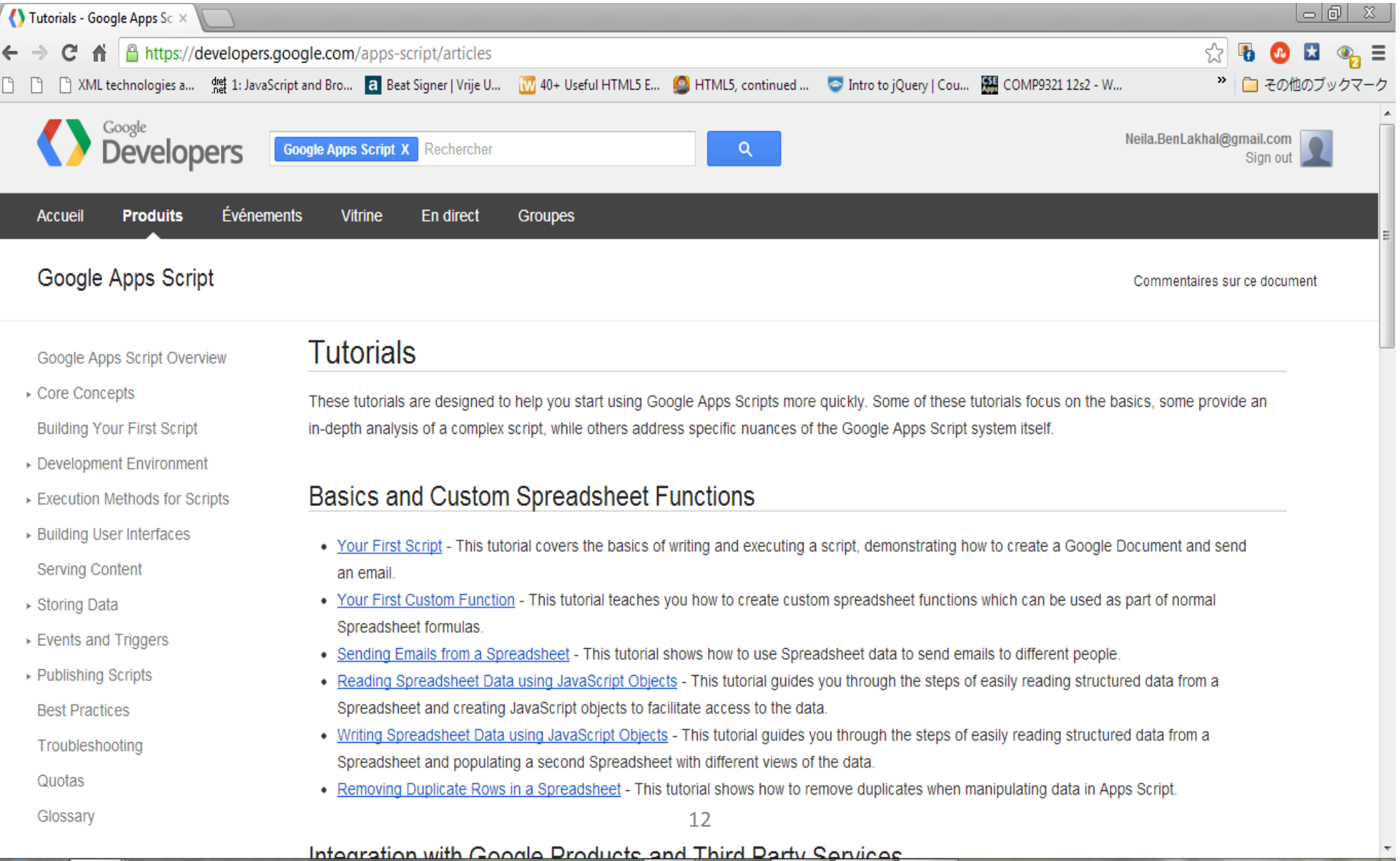
Des variantes de Javascript

- Il y en a ceux qui ont décidé même de définir leurs propres langages de script pour développer des applications Web, par exemple :
  - Google Apps Script is a JavaScript cloud scripting language that lets you extend Google Apps and build web applications. Scripts are developed in Google Apps Script's browser-based script editor, and they are stored in and run from Google's servers. (https://script.google.com)
  - Processing.js (visualization environment , digital art, interactive animations, educational graphs, video games, etc.)
  - Etc.

Exemple Google apps Script



Exemple de fonctionnalités possible par google apps scripts



Syntaxe Javascript

A quel emplacement insérer le code Javascript ?

- Il existe 2 façons d'inclure du JavaScript :
  - Soit ***DANS*** la page HTML:
    - Grâce aux événements (**event handler**) dans des éléments HTML en tant que attribut/valeur.
    - Grâce à la balise <SCRIPT> …</SCRIPT>.
  - Soit ***HORS*** de la page HTML:
    - En mettant le code dans un fichier .js **externe**

Intégrer du code JS dans HTML

Méthode 1

- Pour insérer le code dans une balise, une nouvelle propriété est nécessaire. Il s'agit du **gestionnaire d'événements** (event handler).
  - Cette propriété est caractéristique du JS : elle suffit à dire au navigateur : "attention, ce qui suit est du JS".
  - Elle porte le **nom de l'événement** qui doit **déclencher le script** (c'est pour cela qu'on parle de "gestionnaire d'événements").
  - Elle contient comme valeur **le script à exécuter** lors de cet événement.

Intégrer du code JS dans HTML

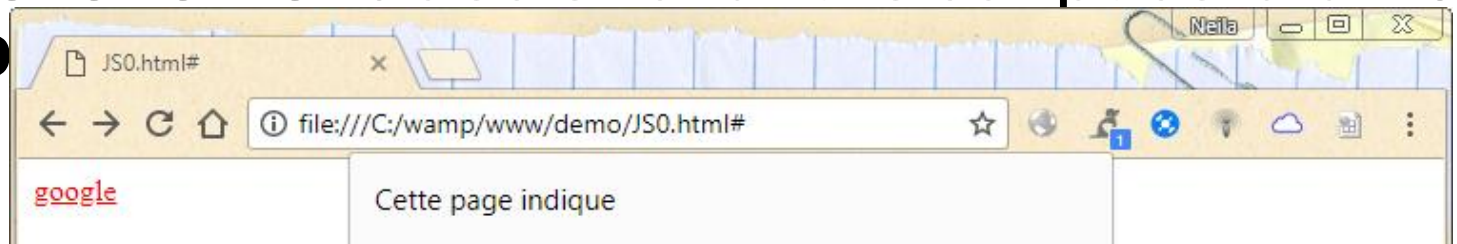
- Pour résumer, la balise en question aura cette forme :

```
<nomdelabalise … monEventHandler="monscript" />
```

- Le script **monscript** est exécuté quand l'événement **monEventHandler** est déclenché;
- Exemple: Nous allons utiliser **un lien hypertexte** :

```
<a href="http://google.com"
  onClick="alert('hello!')">google</a>
```

- **L'événement** déclenchant le script sera un **clic de so**



Intégrer du code JS dans HTML

- Il est possible d'écrire du JavaScript directement à la place de l'adresse d'un lien, en le faisant précéder de **javascript:** comme dans cet exemple :

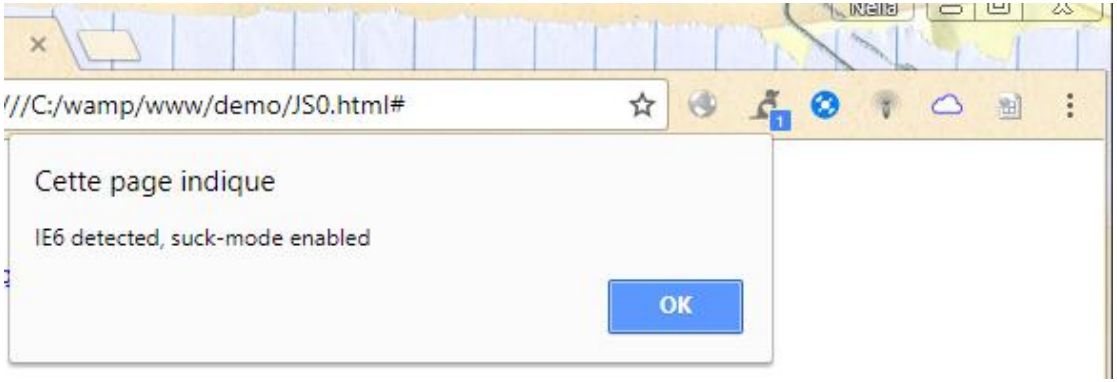
```
<a href="javascript:alert('hello');">Hello</a>
```

- Les 3 écritures sont équivalentes et permettent seulement d'ouvrir une boite de dialogue :

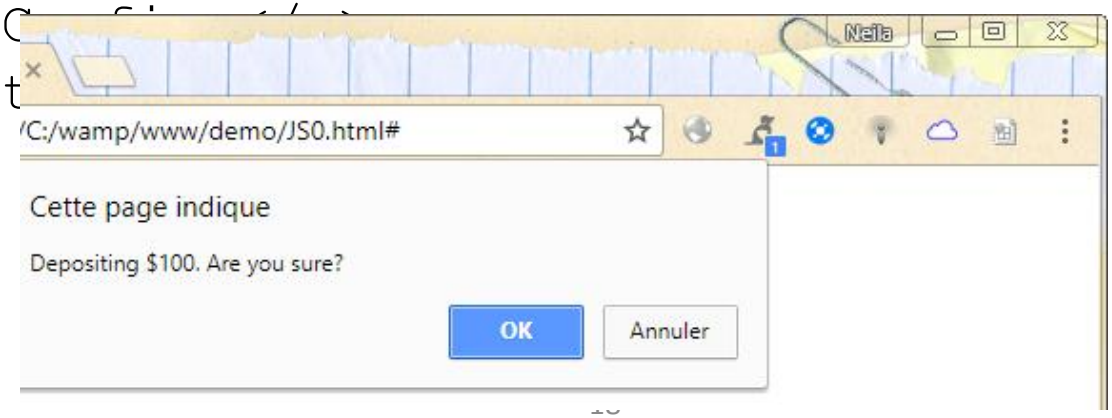
```
<a href="#" onClick="alert('hello!')">Hello</a>
<a onClick="alert('hello!')">Hello</a>
```

Les boites de dialogue

```
<a href="javascript:alert('IE6 detected, suck-mode
enabled');">Red Alert</a>
```

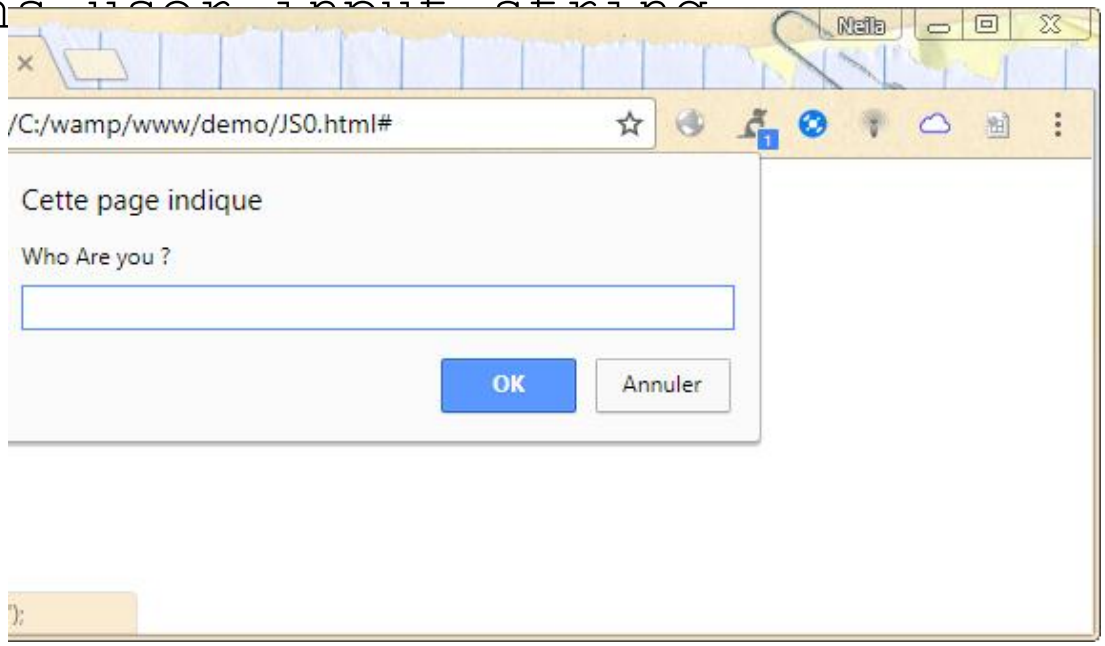



```
<a href="javascript:confirm('Depositing $100. Are you
sure?');">C
//returns t
```



Les boites de dialogue

```
<a href="javascript:prompt('Who Are you ?');">Prompt</a>
// returns user input string
```



Autres gestionnaires d'événements

- Il existe d'autres événements que le clic de souris, leur syntaxe est la même : **onevent**, event étant le nom de l'événement.
- Les événements s'appliquent à la plupart des balises (sauf événements particuliers, relatifs à des balises précises).
- En voici quelques-uns :
  - **onClick** : au clic (gauche) de la souris,
  - **ondblclick** : lors d'un double clic,
  - **onmouseover** : au passage de la souris,
  - **onmouseout** : lorsque la souris "sort" de cet élément (en quelque sorte, le contraire de onmouseover),
  - **onload** : au chargement de la page,
  - Etc..

Autres gestionnaires d'événements

Exemples:

- Voici une image contenant plusieurs gestionnaires d'événements :

```
<img src="monImage.png" alt="Mon Image"
          ondblclick="alert('Double clic');"
          onmouseout="alert('La souris sort');" />
```

- Et voici la balise **<body>** pour créer une page disant "Bonjour" :

```
<body     onload="alert('Bonjour !');" >
    <!-- corps de la page -->
</body>
```

Inconvéniant méthode 1

- Si un même évenement est à détecter dans plusieurs balises du même types, il y aura redonnance!!
- Il est recommendé d'alleger la partie body du code JS.

Intégration de script: méthode 2

Méthode 2 : écrire le script entre **<script>** et **</script>**

```
<script>
 /* votre code javascript se trouve ici c'est déjà plus pratique
    pour un script de plusieurs lignes ceci est un commentaire de
    plusieurs ligne */
</script>
```

- On peut placer l'élément **<script>** :
  - Entre **<body>** et **</body>** : Sont à placer les scripts à exécuter au chargement de cette dernière (lorsque le navigateur "lira" le code, il l'exécutera en même temps).
  - Entre **<head>** et **</head>** : Sont à placer dans l'en-tête les éléments qui doivent être exécutés plus tard (lors d'un événement particulier, par exemple).

Intégration de script: méthode 2 dans le <head> et/ou le <body>

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
        <title>titre de page</title>
        <script>
         // code ici  : exécution différée
        </script>
    </head>
    <body>
        <script>
        // code ici : exécution directe
        </script>
    </body>
</html>
```

Script dans le corps du texte

- **Exécution directe:** le code s'exécute automatiquement lors du chargement de la page HTML dans le navigateur.
- La squelette de la page HTML est :

```
<html>
<head>
   <title> ..... </title>
</head>
<body>
   <script>
      // place du code JavaScript
   </script>
 </body>
</html>
```

Script dans l'entête

- **Exécution différée:** le code est d'abord lu par le navigateur, stocké en mémoire, pour ne s'exécuter que sur demande express.
- La squelette de la page HTML est :

```
<html>
<head>
    <title> ..... </title>
 <script>
     // place du code JavaScript
    </script>
</head>
<body>
    <- - place du code événement - ->
</body> </html>
```

Exemple

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8">
<script>
function saluer() {
alert("bonjour");
}
</script>
</head><body>
<h1>exécution immédiate</h1>
<script>saluer(); </script>
<h1>exécution sur événement onclick</h1>
<form><input  type="button" name="bouton" value="salut"
   onclick="saluer();"/></form>
<h1>exécution sur protocole javascript</h1>
<a href="javascript:saluer();">pour saluer</a>
</body></html>
```

Placer le code JS dans un fichier séparé

- Comme pour le CSS, on peut très bien placer notre code dans un fichier indépendant. On dit que **le code est importé depuis un fichier externe**. L'extension du fichier externe contenant du code JavaScript est **.js**.
- On va indiquer aux balises le nom et l'emplacement du fichier contenant notre (ou nos) script(s), grâce à la propriété `src` (comme pour les images).
- Syntaxe :

  <head>…

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

  …</head>

Exemple

```
<!DOCTYPE html>
<html><head><title>exemples de déclenchements</title>

<script src="monscript.js"></script>
</head><body>
<h1>exécution immédiate</h1>
<script>
saluer();
</script>
<h1>exécution sur événement onclick</h1>
<form><input  type="button" name="bouton" value="salut"
   onclick="saluer();"/>
</form>
<h1>exécution sur protocole javascript:</h1>
<a href="javascript:saluer();">pour saluer</a>
</body>
</html>
```

monscript.js

```
function saluer() {
alert("bonjour !");
}
```

Notations Javascript

- Commentaire :
  - Sur une ligne : // … commentaire …
  - Sur plusieurs lignes : /* … commentaire … */
- Séparateur d'instructions :
  - Comme en C++/Java, les instructions se termine par un **;** (point virgule) :
  - Un retour a la ligne est aussi interprété comme fin de l'instruction.
- Groupement d'instructions
  - Accolades : **{** … instructions … **}**

•Date
•RegExp

Déclaration de variables 1/2

- Les types disponibles en JavaScript sont :
  - **String** : est du texte
  - **Number** : est un nombre
  - **Object** : est un objet (Function, Array, Date et RegExp)
  - **Underfined :** Toute variable non définie est de type **Undefined** et a pour valeur **undefined**
  - **Null** : le type Null ne peut avoir qu'une valeur null
  - **Boolean** : le type Boolean peut prendre deux valeurs **true** ou **false**
    - "falsey" values: 0, 0.0, NaN, "", null, and undefined
    - "truthy" values: anything else

Déclaration de variables 2/3

- Pour définir une variable : `var|let nomvariable=valeur;`
- Pas de typage explicite : détection automatique par l'interpréteur

```
var nulle= null;
// nulle existe, mais a pour valeur null
var caroline;
//caroline a pour valeur undefined :n'existe pas
var age = 9;// Number
var weight = 27.4; // Number
var Name = “Sarah”; //String
var name=‘autrechaine’;//String
console.log(typeof(caroline)); //undefined
```

Type String

```
var x=10.1;
console.log(typeof(x));// Number
var Name= "Sarah BEN";
var fName = Name.substring(0, Name.indexOf("
")); console.log(fName);// "Sarah"
var taille= Name.length;
console.log(taille);// 9
'hello'.length; // 5
```

Methodes: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf,
replace, split, substring, toLowerCase, toUpperCase

Length : property

Concatenation with + : 1 + 1 is 2, but "1" + 1 is "11"

Type Boolean

```
var iLikeJS = true;
var ieIsGood = "IE6" > 0;
console.log(ieIsGood); // false
if ("web dev is great")
{ /* true */ }
if (0) { /* false */ }
```

- Any value can be used as a Boolean
  "falsey" values: 0, 0.0, NaN, "", null, and undefined
  "truthy" values: anything else
- Converting a value into a Boolean explicitly:
  `var boolValue = Boolean(otherValue);`

Déclaration de variables 2/3

- Nomenclature des variables :
  - Nom de variable **sensible à la casse** (**myVar** et **myvar** sont 2 variables différentes).
  - Les variables nommées ne peuvent pas contenir les espaces, ou commencer par tout nombre, et tous les symboles de ponctuation excepté le soulignage (_) sont restreints.

- La portée de variables
  - Les variables déclarées ou initialisées en dehors d'un corps de fonction ont une portée globale, rendant lui accessible à tous les autres le rapport dans le même document.
  - Les variables déclarées ou initialisées dans un corps de fonction a une portée locale, le rendant accessible seulement aux instructions dans le même corps de fonction.

Déclaration de variables : exemple

```
<body><script>
//First we will declare a few variables and assign values to
   them
var myText = "Good day!";
var myNum = 5;
//Note that myText is a string and myNum is numeric.
</script>
<h1>Any other code…</h1>
<script>
//Next we will display these variables to the user.
document.write(myText);
//document.write writes text in the web page,
//The text can be formatted with HTML tags
//To concatenate strings in JavaScript, use the '+' operator
document.write("My favourite number is "+ myNum);
</script>…
```

Portée des variables: exemple 1

```
<script>
   var altitude = 5; // VARIABLE GLOBAL
function declareVar(){
 myVar=17; // VARIABLE GLOBALE
 var my2Var=15; // VARIABLE LOCA
 console.log(my2Var);//affiche 1
}


declareVar();// affichera 15
console.log (altitude) // affichera 5
console.log(myVar);// affichera 17
console.log(my2Var);// affichera une erreur
</script>
```

*Quand vous déclarez dans une fonction des variables sans le mot-clé **var**, alors ces variables sont globales: affectation de valeur sans déclaration de la variable.*

*Vous obtenez une variable locale par la déclaration de la variable avec **var** à l'intérieur d'une fonction.*

Portée des variables: exemple 2

```
<script>
var a = 12;  //variable globale
var b = 4;
function MultipliePar2(b) {
var a = b * 2;  //variable locale
return a; }
document.write("Le double de ",b," est
  ",MultipliePar2(b));
//affichera le double de ,4, est ,8
document.write("La valeur de a est ",a);
//affichera la valeur de a est , 12
</script>
```

Portée des variables: exemple 3

```
<script>
var a = 12;  //variable globale
var b = 4;
function MultipliePar2(b) {
var a = b * 2;  // variable locale globale
return a; }
document.write("Le double de ",b," est
  ",MultipliePar2(b));
//affichera le double de ,4, est ,8
document.write("La valeur de a est ",a);
//affichera la valeur de a est , 12  8
</script>
```

Variables de bloc

```
let a;
let name = 'Sarah';
```

- let allows you to declare block-level variables.
- The declared variable is available from the function block it is enclosed in.

Exemple

```
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<script>
function letVariable(){
// name *is* visible here
let name="sarah as bloc variable"; // the same as if declared with var
// to make it visible implicite declaration
document.write(name+"<br/>");
}
</script>
</head><body>
<script>
letVariable();
// name *is* not visible out here
document.write(name+"<br/>");
</script></body>
```

Exemple de variable de bloc

```
for (let myVariable = 0; myVariable < 5; myVariable++) {
// myVariable is visible here
document.write(myVariable); //01234
}
document.write(myVariable);
// myVariable *is* not visible out here
// Error Undefined Variable
```

Exemple let, var et implicite

```
let x=10;
console.log(x);// Affichera 10 considéré comme Globale
function local(){
let Y=12; console.log(Y);
} local();//Affichera 12
console.log(Y); // Erreur Considéré comme Locale
function Notglobal(){
var Z=13;
console.log(Z);}
Notglobal();// Affichera 13
console.log(Z);//Erreur
function global(){
W=14; console.log(W);
}
global();//Affichera 14
console.log(W);//Affichera 14
```

La conversion de types

Nombre --> chaîne: par l'ajout de deux apostrophes

```
var count = 10; var s1 = "" + count; console.log(typeof(s1)) //
affichera string
```

Chaîne --> nombre:

```
parseInt(chaine_a_convertir)
parseFloat(chaine_a_convertir)
```

```
x= '1024'; y=parseInt(x);
console.log("type of x :" + typeof(x));
//affichera string
console.log("type of y :" + typeof(y));
//affichera number
console.log (x+y);//affichera 10241024
console.log(x-y);//affichera 0
```

Data Types conversion: ParseInt |ParseFloat

```
<!DOCTYPE html>
<html><head>
  <title>Data Type conversion</title>
</head>
<body>
 <script>
    userName = prompt("What is your name?", "");

    userAge = prompt("Your age?", "");
    //userAge = parseInt(userAge);

    document.write("Hello " + userName + ".")
    if (userAge < 18) { //operateur de comparison : conversion implicite
      document.write("  Do your parents know " + "you are online?");
    }
    else {
      document.write("  Welcome friend!");
    }
</script></body></html>
```

Data Types conversion: ParseInt |ParseFloat

```
<!DOCTYPE html>
<html>
   <head><title>Data Types conversion</title></head>
   <body>
     <script>
   var nombre1 = prompt('Premier nombre ?');
   var nombre2 = prompt('Deuxieme nombre ?');
   // on convertit en nombres décimaux
   var resultat = parseFloat(nombre1) +
     parseFloat(nombre2);

   alert("La somme de ces deux nombres est " +
     resultat);
   </script>
   </body>
   </html>
```

Déclaration de constante

- const allows you to declare variables whose values are never intended to change. The variable is available from the function block it is declared in.

```
const Pi = 3.14; // variable Pi is set
Pi = 1; // Error because you cannot change a constant.
```

- Constant are also case sensitive, like variables.

JavaScript Array

- **Array** est utilisé pour stocker une séquence d'éléments, accessible via un index. Comme JS est faiblement typée, les éléments d'un même tableau peuvent être de **types différents**.

- Pour créer un tableau, il faut allouer de l'espace en utilisant **new** ou en lui assignant des valeurs **directement**.
- Le premier élément du tableau est **indexé à 0**.
- Exemples :

```
var items = new Array(10); // allocation d'espace pour 10 elts
var Items = new Array(); // if no size given, will adjust
   dynamically
var ITems = [0,0,0,0,0,0,0,0,0,0]; // can assign size & values []
var divers = ['t', 1978, [0, 1, 2]];
```

JavaScript Arrays

- Pour accéder a un élément du tableau : `[]` (comme C++/Java)

```
var items = new Array(10);
// allocation d'espace pour 10 elts
for (i = 0; i < 10; i++) {
items[i] = 0; // stores 0 at each index
}
```

- `length` indique le nombre d'éléments d'un tableau.

```
for (i = 0; i < items.length; i++) {
document.write(items[i] + "<br/>");
// displays elements
}
```

JavaScript Arrays

```
var myList = [ ,'JS', , 'PHP'];
console.log(myList);
/*affichera :
(4) [empty, "JS", empty, "PHP"]
   1:"JS"
   3:"PHP"
   length:4 */
console.log(myList[2]); //affichera undefined
myList[5] = 'html';
console.log(myList[5]); // 'html'
console.log(Object.keys(myList)); // ["1", "3", "5"]
console.log(myList.length); // 6
```

Increase/decrease Array Size

```
myList.length = 10; // allocation dynamique
console.log(Object.keys(myList));
// ["1", "3", "5"]
console.log(myList.length); // 10


myList.length = 2; // suppression d’éléments
console.log(Object.keys(myList)); // ['1']
console.log(myList.length); // 2
```

Tableaux associatifs

- Principe
  - L'indice est une chaîne de caractères,
- Exemple: Chargement d'une page HTML en fonction du jour de la semaine:

```
var semaine = new Array();
semaine["lundi" ] = "cours.html";
semaine["dimanche" ] = " weekend.html";
```

Accès à la clé et au contenu

```
<script>
var myArray = new Array();
myArray['one'] = 1;
myArray['two'] = 2;
myArray['three'] = 3; // show the values stored
for (var i in myArray) {
console.log('key is: ' + i + ', value is: ' + myArray[i]);
}
</script>
```

Les methodes du tableau

- Methods: concat, join, pop, push, revers
  e, shift, slice,

sort, splice, toString, unshift

Opérateurs de comparaison

- **Les opérateurs d'égalité** (fonctionnent aussi avec des chaînes de caractères)
  - **a= =b** si les deux valeurs sont égales, alors on a true, sinon false.
  - **a!=b** : si les deux valeurs sont différentes, alors on a true, sinon false.
- **Opérateurs de comparaison** de valeurs numériques :
  - **a < b** : si **a** est inférieur à (plus petit que) **b**, alors on a true, sinon false.
  - **a > b** : si **a** est supérieur à (plus grand que) **b**, alors on a true, sinon false.
  - **a <=** b : si **a** est inférieur ou égal à **b**, alors on a true, sinon false.
  - **a >= b** : si **a** est supérieur ou égal à **b**, alors on a true, sinon false.

Incrémentation / décrémentation

- Il existe, de la même manière, les opérateurs suivants:
  - **+=** (on retranche la valeur de la deuxième variable à celle de la première),
  - **-=** (on retranche la valeur de la deuxième variable à celle de la première),
  - ***=** (on multiplie la valeur de la première variable par celle de la deuxième),
  - **/=** (idem mais avec une division) et **%=**.
- **Incrémentation / décrémentation**
- Lorsque l'on veut augmenter de 1 la valeur d'une variable on utilise la notation : variable++;
- De même, pour **décrémenter** (diminuer la valeur de 1) une variable, le code est le suivant : variable--;
- variable += X; équi. a variable = variable + X; de même pour -=, /= , *= et %=.

Les opérateurs logiques.

- Il s'agit de l'opérateur ET et de l'opérateur OU.
- En JS, on les note **&&** (**ET**) et **||** (**OU**) .

Exemple:

```
var taille = prompt('Combien mesurez-vous ? (en
  m)');
var poids = prompt('Combien pesez-vous ? (en
  kgs)');
costaud = (taille>=2 && poids>=90);
alert('Vous êtes costaud : ' + costaud);
```

- **La négation : Symbole "NON" (on le note !)**

Exemple:

```
mineur = !(age >= 18);
```

Conversion implicite dans les expressions

- Les opérateurs logiques > < >= <= && || ! == != font une **conversion automatique** de types avant d'évaluer une expression:

```
5 < "7" //is true
42 == 42.0 //is true
alert("5.0" == 5) //affichera true
```

- === and !== sont plus stricts et vérifient le type des deux opérandes:

```
alert("5" === 5) //affichera false
```

Les instructions conditionnelles

Syntaxe :

```
if (condition)
   instructions
 else
   instructions
```

```
switch (expression)
  { case valeur1 :
  instructions
     break;
 case valeur2
  :instructions
     break;
...
default :
  instructions
     break; }
```

- break; arrête une boucle
- continue; passe à l'itération suivante de la boucle

If else

```
<script>
//If the time is less than 10, you will get "Good
morning" //Otherwise you will get a "Good day"
var d = new Date() // créer un objet de type Date
var time = d.getHours() //retourne l'heure actuelle
 if (time < 10) {
document.write("Good morning!")
}
else {
document.write("Good day!")
}
</script>
```

Les boucles

```
while (condition)
    instructions
```

```
initialisation;
    while (condition)
    {
      instructions
     incrémentation
    }
```

```
do {
     instructions
    }
    while
    (condition) ;
```

```
for (initialisation; condition; incrémentation)
 {
 instructions
}
```

```
<html>
<body>
<script>
 var i=0 for (i=0;i<=10;i++) {
document.write("The number is " + i)
document.write("<br/>")
 }
</script>

</body>
 </html>
```

Exemple : switch

```
<script>
var d = new Date()
 theDay = d.getDay()
 switch (theDay){
 case 5:document.write("Finally Friday")
 break
case 6:document.write("Super Saturday")
break
case 0:document.write("Sleepy Sunday")
break
default:document.write("I'm really looking forward to this weekend!"
</script>
```

Les fonctions

- Emplacement de la déclaration
  - Dans l'entête de la page,
  - Utilisation de la syntaxe : `function nom_fonction(param1, …)`
- Le corps de la fonction est délimité par { … }
- Contenu :
    - Déclaration des variables locales, propres à la fonction,
    - Instructions réalisés par la fonctions,
    - Instruction return pour renvoyer une valeur ou un objet : cette instruction n'est pas obligatoire vu qu'il y a des fonctions qui ne renvoie pas de valeur,
    - Les paramètres et la valeur de retour n'ont pas de type prédéfinis, contrairement a Java /C ++ car JS est faiblement typé.

Appel de fonction

- Peut avoir lieu à n'importe quel endroit de la page dans d'autres fonctions, dans le corps de la page.
- On peut appeler une fonction avant sa déclaration.
- Utilisation de : nom_fonction(param1, …);

```
<HTML>
<HEAD>
 <SCRIPT>
// déclaration de fonction
function bonjour(prenom) { document.write("Bonjour
   "+prenom+"<br/>"); }

 </SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
// appel de la fonction
bonjour("Toto");
</SCRIPT></BODY></HTML>
```

User-Defined Functions

- Il est possible de limiter la portée d'une variable ,
- Si lors de la première utilisation d'une variable, elle est précédée par var, alors il s'agit de variable locale à la fonction.

```
function isPrime(n)
// Assumes: n > 0
// Returns: true if n is prime, else false
{ if (n < 2)
{ return false; }
 else if (n == 2) {
return true; }
Else
 { for (var i = 2; i <= Math.sqrt(n); i++)
 { if (n % i == 0)
 {return false;}
} return true;
}
}
```

L'objet Math:

```
three = Math.floor(Math.PI);
```

méthodes:

abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan

Propriétés : E, PI

Exemple:

Math.sqrt(n) : Retourne la racine carrée du nombre réel passé en paramètre.

Function Example

```
<html><head><title>Prime Tester</title>
<script>
function isPrime(n) {// CODE PREVIOUS SLIDE }
</script></head>
<body>
<script>
testNum = parseInt(prompt("Enter a positive integer");
if (isPrime(testNum)) {
document.write(testNum + " <b>is</b> a prime number."); }
 else
{
document.write(testNum + " <b>is not</b> a prime number.");
 }
</script>
</body></html>
```

Anonymous functions

- Une fonction anonyme est une fonction dont le nom a été omis vu qu'elle va pas être appelé plusieurs fois, c'est un bloque de code simple et qui sera exécuté tout de suite, elle va pas être réutiliser.
- Exemple :

```
<a id="mylink" href="#">link</a>
<script>
mylink.onclick = function() {
alert('hello');
// I can put as much code
// inside here as I want
}
</script>
```

IIFE (Immediately Invokable Function Expression)

- Quand une function est appelé une et une seule fois une forme pratique est IIFE.
- Déclaration et appel de la function en une même instruction SANS attribuer un nom à la fonction.

Exemple :

```
(function () {
var x = "Hi!!";
alert(x);
}
)(); // I will invoke myself
```

Fonction anonyme /expression

- Une fonction JS peut être stocker dans une variable :

```
var expression=function (a,b) { return
a*b;} console.log(expression); //affiche le code de
la fonction
console.log(expression(5,10)); // affichera 50
```

- Une fonction JS peut être utiliser dans une expression :

```
function somme (a,b) { return a*b;}
var z=somme(10,20)*10;
console.log(z);//affichera 2000
```

Fonction dans une fonction

- Une function peut être définie dans le corps d'une d'autre :
- Exemple:

```
function myBigFunction()
{ myValue=10;
subFunction1();
}
function subFunction1() { console.log(myValue); }

myBigFunction();//affichera 10
subFunction1();//affichera 10
```

Les objets

- “An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called properties and methods when they are inside objects.”
- En JS, vous pouvez créer vos propres objets :
- Il y a plusieurs manières de créer un objet :
- Méthode 1: La noation JS:

```
var person = {};
```

- Méthode 2: Le constructeur Object() (besoin de plusieurs instances):

```
var person1 = new Object();
```

- Méthode 3: Une fonction constructeur :

```
function Person(name) {…};
var person1 = new Person('Bob');
```

Exemple d'objets : méthode 1

```
var person = {};
var person = {
name: ['Bob', 'Smith'],
age: 32, gender: 'male',
greeting: function() { alert('Hi! I\'m ' +
this.name[0] + '.'); }};
console.log(person.name[0])//Bob
person.age//32
person.greeting()
```

Exemple d'objets : méthode 2

```
var person1 = new Object();
person1.name = 'Chris';
person1['age'] = 38;
person1.greeting = function() { alert('Hi! I\'m ' + this.name + '.'); };

var person2 = Object.create(person1);
/*person2 aura les mêmes propriétés et valeurs que person1*/
console.log(person2.name)// affichera Chris
person2.greeting()
```

Exemple d'objets : méthode 3

```
function Person(name) {
this.name = name;
this.greeting = function() { alert('Hi! I\'m '
+ this.name + '.'); };
}
var person1 = new Person('Bob');
var person2 = new Person('Sarah');
Console.log(person1)
/* affichera :
Person {name: "Bob", greeting: f}
   greeting:f ()
   name:"Bob"   */
```

Exercice

- On considère la fonction constructeur d'objet suivante :

```
function Person(first, last, age, gender, interests)
{ this.name = { 'first': first, 'last' : last };
this.age = age;
this.gender = gender;
this.interests = interests;
this.bio = function()
{ alert(this.name.first + ' ' + this.name.last + ' is ' +
this.age + ' years old. He likes ' + this.interests[0] +
' and ' + this.interests[1] + '.'); };
this.greeting = function() { alert('Hi! I\'m ' +
this.name.first + '.'); };
}
```

Exercice

- Création d'une instance :

```
var person1 = new Person('Bob',
'Smith', 32, 'male', ['music',
'skiing']);
```

- On souhaite personnaliser les messages de la fonction bio() et greeting() pour qu'ils s'adaptent en fonction du genre et du nombre d'activités favorites de la personne.