Initiation au langage JavaScript

Plan

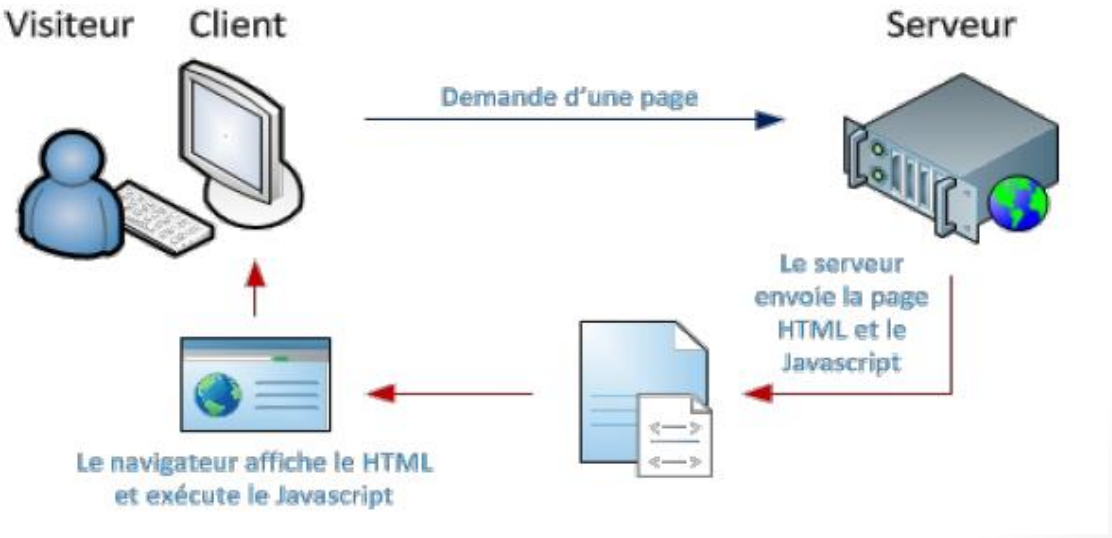
- Introduction
- HTML et JavaScript
- Variables et manipulations
- Opérateurs arithmétiques et logiques
- Instructions conditionnelles
- Instructions itératives
- Fonctions
- Notion d’objet en JavaScript
- Structure DOM
- Programmation événementielle

Introduction

- JavaScript est un **langage de programmation de scripts** principalement utilisé dans les pages web interactives côté client.
- C’est un langage interprété c.à.d. exécuté par un autre programme (c’est le navigateur)

Introduction

- Javascript est un langage dit client-side, c'est-à-dire que les scripts sont exécutés par le navigateur chez l'internaute (le client).
- Cela diffère des langages de scripts dits server-side qui sont exécutés par le serveur Web. C'est le cas des langages tel que PHP.



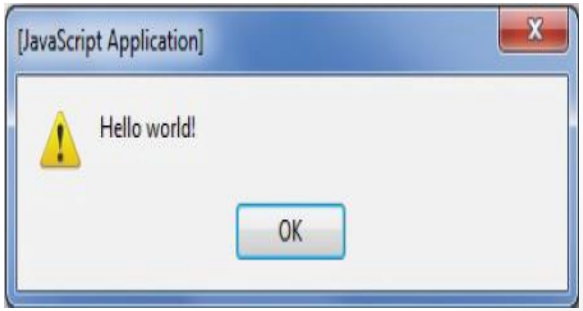
Intérêts de JavaSript

- Intérêt d'utilisation de JavaScript:
- Contrôler les données saisies dans des formulaires HTML
- Interagir avec le document HTML via l'interface DOM (*Document Object Model*) fournie par le navigateur (on parle alors parfois de HTML dynamique ou DHTML)
- Modifier le contenu des pages web par programmation avec la méthode Ajax (*Asynchronous Javascript And XML*)
- Remarque : javascript est aussi utilisé pour réaliser des services dynamiques, ou à des fins ergonomiques.

HTML et JavaScript

- On peut inclure notre script dans une page HTML selon les deux méthodes suivantes:
- A l'intérieur de la page HTML en utilisant la balise <script type='text/javascript'> … code JavaScript </script>

```
<HTML>
  <HEAD>
    <TITLE>Exemple 1</TITLE>
    <SCRIPT TYPE="text/javascript">
    // Du code Javascript
    alert("Hello world !"); // affiche une boîte de dialogue modale
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```



HTML et JavaScript

- Dans un fichier externe de page HTML en précisant l'extention .js pour le fichier script et en l'appelant à travers la balise:

<script type='text/javascript' src='monfichier.js> </script>

```
<HTML>
   <HEAD>
      <TITLE>Exemple 3</TITLE>
      <SCRIPT TYPE="text/javascript" SRC="monscript.js"></SCRIPT>
   </HEAD>
   <BODY>
   </BODY>
</HTML>
```

HTML et JavaScript

- L'élément NOSCRIPT permet de fournir un contenu de remplacement pour les navigateurs qui ne peuvent exécuter un script.

```
<HTML>
   <HEAD>
      <TITLE>Exemple 2</TITLE>
      <SCRIPT TYPE="text/javascript">
      <!--
      document.write("<P>Votre navigateur accepte le Javascript.</P>");
      //-->
      </SCRIPT>
      <NOSCRIPT><P>Votre navigateur n'accepte pas le Javascript.</P></NOSCRIPT>
   </HEAD>
   <BODY>
   </BODY>
</HTML>
```

Variables et manipulations

- Variables:
- Dans JavaScript les variables ne sont pas typés

Il suffit d'écrire:

var mavariable;

pour déclarer une variable.

Variables et manipulations

- En fait, Javascript n'autorise la manipulation que de 4 types de données :
- Des nombres : entiers ou à virgules
- Des chaînes de caractères (string) : une suite de caractères : entre deux côtes ou entre deux côtes.
- Des booléens : des variables à deux états permettant de vérifier une condition :
    - *false*: lors d'un résultat faux
    - *true*: si le résultat est vrai
- Des variables de type *null* : un mot caractéristique pour indiquer qu'il n'y a pas de données.

Opérateurs arithmétiques

- Les opérations d'addition, soustraction, division, multiplication et modulo sont disponibles en mode texte.

```
var result = 3 + 2;
alert(result); // Affiche "5".
```

```
var number1 = 3, number2 = 2, result;
result = number1 * number2;
alert(result); // Affiche "6".
```

```
var divisor = 3, result1, result2, result3;

result1 = (16 + 8) / 2 - 2 ; // 10
result2 = result1 / divisor;
result3 = result1 % divisor;

alert(result2); // Résultat de la division : 3,33
alert(result3); // Reste de la division : 1
```

Opérateur	Signe assigné
addition	+
soustraction	-
multiplication	*
division	/
modulo	%

Opérateurs de comparaison

- Les opérateurs de comparaison servent à exprimer des conditions. Attention à ne pas confondre l'opérateur de comparaison == avec le signe = d'assignation.

Opérateur	Signification
==	égal à
!=	différent de
===	contenu et type égal à
!==	contenu ou type différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Opérateurs logiques

- Les opérateurs logiques servent à créer des jonctions ou des conjonctions ou la négations
- Il peuvent être utilisé dans les structures de contrôles
- Retournent true ou bien false

```
var result = 2 > 8 && 8 > 2;
alert(result); // Affiche "false";
```

Opérateur	Type de logique	Utilisation
&&	ET	valeur1 && valeur2
||	OU	valeur1 || valeur2
!	NON	!valeur

Opérations de conversion de type

- Concaténation: Chaine1+chaine2

```
var hi = 'Bonjour', name = 'toi', result;
result = hi + name;
alert(result); // Affiche "Bonjourtoi".
```

```
var start = 'Bonjour ', name, end = ' !', result;

name   = prompt('Quel est votre prénom ?');
result = start + name + end;
alert(result);
```

- La fonction parseInt() convertit une chaine de caractére en un nombre:

```
var text = '1337', number;

number = parseInt(text);
alert(typeof number); // Affiche "number".
alert(number); // Affiche "1337".
```

```
var first, second, result;

first = prompt('Entrez le premier chiffre :');
second = prompt('Entrez le second chiffre :');
result = parseInt(first) + parseInt(second);

alert(result);
```

- On peut convertir un nombre en une chaine de caractères en utilisant les cotes et l'opérateur +

```
var text, number1 = 4, number2 = 2;
text = number1 + '' + number2;
alert(text); // Affiche "42";
```

Instructions conditionnelles

- Verifie si une condition est vrai ou non à travers l'instruction:

if (condition)

{

Traitements;

} else

{

Traitements;

}

```
if (2 < 8 && 8 >= 4) { // Cette condition renvoie "true", le code
est donc exécuté.
  alert('La condition est bien vérifiée.');
}

if (2 > 8 || 8 <= 4) { // Cette condition renvoie "false", le code
n'est donc pas exécuté.
  alert("La condition n'est pas vérifiée mais vous ne le saurez pas
vu que ce code ne s'exécute pas.");
}
```

```
var floor = parseInt(prompt("Entrez l'étage où l'ascenseur doit se
rendre (de -2 à 30) :"));

if (floor == 0) {

  alert('Vous vous trouvez déjà au rez-de-chaussée.');

} else if (-2 <= floor && floor <= 30) {

  alert("Direction l'étage n°" + floor + ' !');

} else {

  alert("L'étage spécifié n'existe pas.");

}
```

Instructions conditionnelles

- L'instruction switch

switch (var)

case 1

Traitements 1;

break;

case 2

Traitement 2;

break;

…

Default

Traitements par défaut;

```
var drawer = parseInt(prompt('Choisissez le tiroir à ouvrir (1 à 4)
:'));

switch (drawer) {
  case 1:
    alert('Contient divers outils pour dessiner : du papier, des
crayons, etc...');
  break;

  case 2:
    alert('Contient du matériel informatique : des câbles, des
composants, etc...');
  break;

  case 3:
    alert('Ah ? Ce tiroir est fermé à clé ! Dommage !');
  break;

  case 4:
    alert('Contient des vêtements : des chemises, des pantalons,
etc...');
  break;

  default:
    alert("Info du jour : le meuble ne contient que 4 tiroirs et,
jusqu'à preuve du contraire, les tiroirs négatifs n'existent pas.");
}
```

Instructions itératives

- Des instructions qui donnent la main pour répéter un ou plusieurs traitements tant de fois
- Boucle while

while (condition)

{

Traitements

}

```
var number = 1;

while (number < 10) {
  number++;
}

alert(number); // Affiche 10
```

Instructions itératives

- Boucle do { … } while, exécute un ou plusieurs traitements avant la boucle while

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

Instructions itératives

- Boucle for:
- Contient 3 blocs:
- initialisation
- Condition
- Incrémentation

```
for (initialisation; condition; incrémentation) {
  instruction_1;
  instruction_2;
  instruction_3;
}
```

```
for (var iter = 0; iter < 5; iter++) {
  alert('Itération n°' + iter);
}
```

```
for (var nicks = '', nick; true;) {
  nick = prompt('Entrez un prénom :');

  if (nick) {
    nicks += nick + ' ';
  } else {
    break;
  }
}

alert(nicks);
```

Fonctions

- Une fonction contient un ensemble d'instructions à exécuter lors de son appel.

```
// Définition :
function mafonction(param1, ..., paramN)
{
   // code JavaScript
   // ...
   return variable_ou_valeur ;
}

// Appel :
var res = mafonction(var1, val2, varN);

// Remarque : la passage des paramètres est réalisé par valeur
```

```
function isHumanAge(age) {
   if ((age < 0) || (age > 120)) { return false; }
   else { return true; }
}

var age = window.prompt("Donnez votre age : ", "1");

if(!isHumanAge(age)) {
   window.alert("Vous ne pouvez pas avoir " + age + " ans !");
}
```

Fonctions

- Une fonction peut utiliser des variables locales c.à.d des variables déclarés au sien de la fonction même.

```
function sayHello() {
  var ohai = 'Hello world !';
}

sayHello();

alert(ohai);
```

- Des variables globales qui sont des variables qui se trouvent dans le script et accessibles dans tout le script

```
var ohai = 'Hello world !';

function sayHello() {
  alert(ohai);
}

sayHello();
```

Fonctions

- On peut utiliser des fonctions avec ou sans arguments selon le besoin:

```
// Voici la fonction alert sans argument, elle n'affiche rien.
  alert();

// Et avec un argument, elle affiche ce que vous lui demandez.
  alert('Mon message à afficher');
```

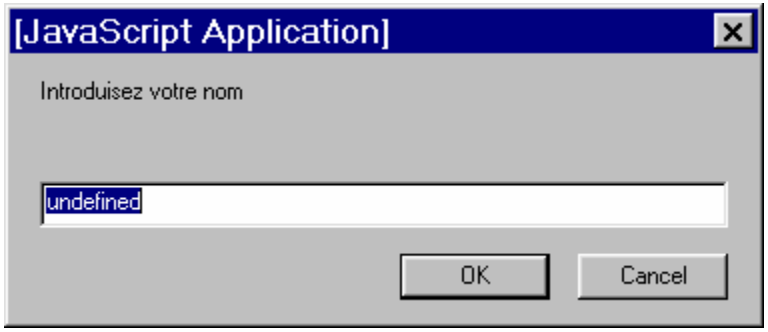
```
function myFunction(arg) { // Notre argument est la variable "arg".
  // Une fois que l'argument a été passé à la fonction, vous allez
le retrouver dans la variable "arg".
    alert('Votre argument : ' + arg);
}

myFunction('En voilà un beau test !');
```

Fonctions

- Fonction alert(): sert à afficher un message dans un fenêtre d'un bouton de type 'ok'

- Fonction prompt(): saisie d'une chaine de caractère.



Fonctions

- Fonction confirm():retourne un boléen true en cas de confirmation false sinon.
- Fonction typeof(): retourne le type de l'argument.
- Fonction parseInt(): convertit l'argument de la fonction en entier.

Notion d’objet en JavaScript

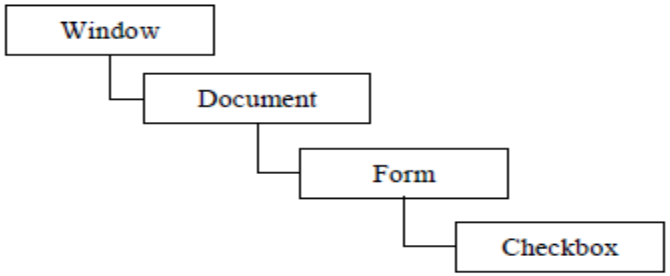
- Un objet possède une structure qui lui permet de pouvoir fonctionner et d'interagir avec d'autres objets.
- Javascript met à notre disposition des objets natifs, c'est-à-dire des objets directement utilisables.
- Les objets contiennent 3 choses distinctes :
  - Un constructeur
  - Des propriétés
  - Des méthodes

Structure DOM

- Le DOM (Document Object Model) est un modèle standardisé par le W3C (World Web Consortium).
- Ce modèle propose de représenter un document sous la forme d'un arbre. Toutes les balises HTML sont donc des noeuds de l'arbre et les feuilles sont soit des balises sans contenu, soit le texte de la page HTML.

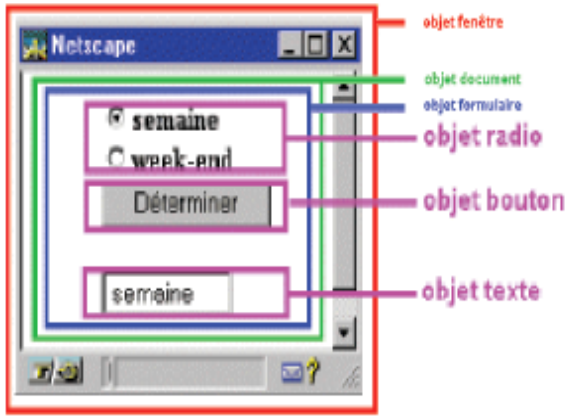
Structure DOM

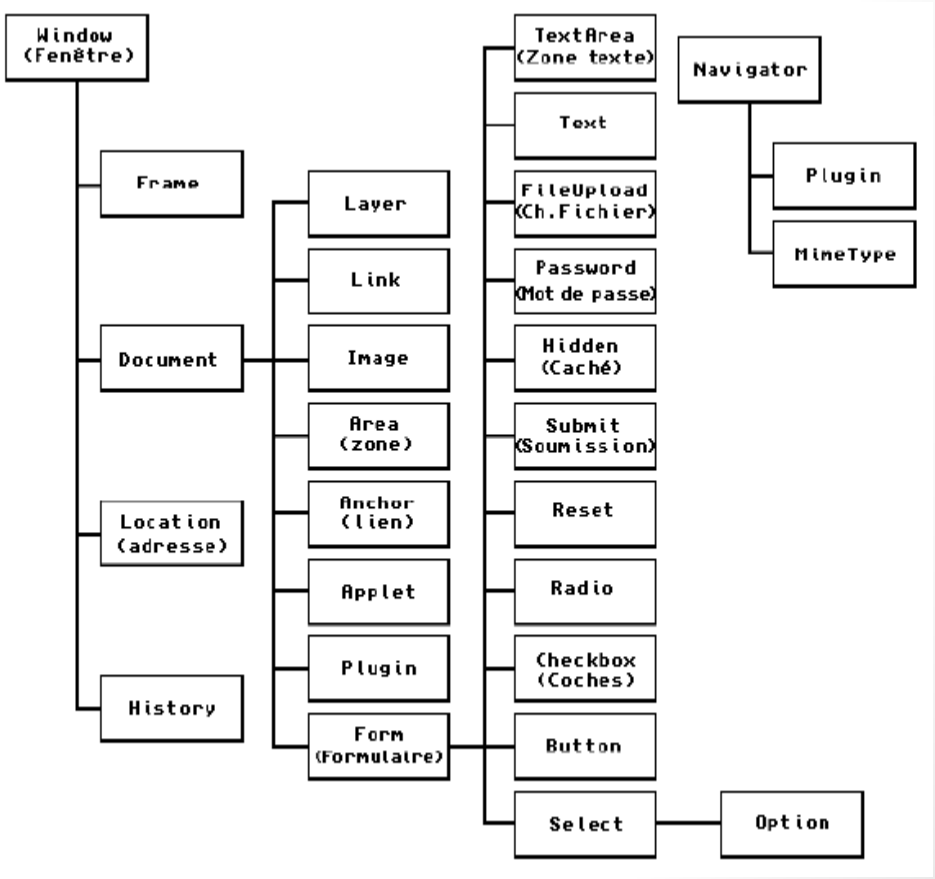
- Javascript traite les éléments qui s'affichent dans votre navigateur comme des objets, c'est-à dire des éléments classés selon :
- Une hiérarchie pour pouvoir les désigner précisément auxquels on associe des propriétés
- Par exemple, pour atteindre un bouton à l'intérieur d'un formulaire, la hiérarchie est:



Structure DOM

- Une page HTML peut être décrit comme suit





Structure DOM

- L'accés au propriétés et méthodes se fait comme suit:
- ob.pr pour accéder à une propriété pr d'un objet ob
- ob.m() pour appeler une méthode m d'un objet ob
- Deux méthodes connues:
- document.write(): écriture dans un objet de type document (du DOM)
- document.read(): lecture à partir d'un objet de type document (du DOM)
- On recommande d'utiliser la méthode getElementById() pour accéder aux objets par leur identifiant (attribut ID de l'élément HTML)

```
var bouton = document.getElementById('id_button');

bouton.click(); // pour simuler un clic de souris sur ce bouton
```

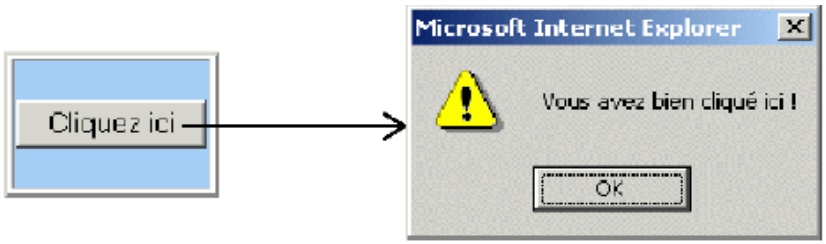
Programmation événementielle

- Les IHM (Interface Homme-Machine) sont généralement basées sur la programmation événementielle qui permet la gestion d'événements.
- Un événement est généralement associé à une action de l'utilisateur :
- appui sur une touche, clic ou déplacement de la souris, ...
- En HTML, il y a très peu d'événements qui sont gérés par défaut :
- clic sur un lien ou sur un bouton de formulaire.
- Le javascript va permettre de gérer et contrôler ces événements (EVENT) par des gestionnaires d'événements (EVENT HANDLER).

Programmation événementielle

- Pour gérer un évènement en JavaScript, il faut installer un gestionnaire d'événement :
- Un gestionnaire d'événement sera l'action déclenchée automatiquement lorsque l'évènement associé se produit.
- La syntaxe courante est la suivante : onEvenement=fonction() où
- Evenement est le nom de l'évènement géré.

```
<FORM>
   <INPUT TYPE="button" VALUE="Cliquez ici" onClick="alert('Vous avez bien
        cliqué ici !')">
</FORM>
```



Programmation événementielle

- Résumé des objets et les évènements associés

Objet	Evénements associables
Lien hypertexte	*onClick, onMouseOver, onMouseOut*
Page du navigateur	*onLoad, onUnload*
Bouton, Case à cocher, Bouton radio, Bouton Reset	*onClick*
Liste de sélection d'un formulaire	*onBlur, onChange, onFocus*
Bouton Submit	*onSubmit*
Champ de texte et zone de texte	*onBlur, onChange, onFocus, onSelect*

Merci pour votre attention