Université Sultan Moulay Slimane
Ecole Supérieure de Technologie - Beni Mellal
Département Informatique & Techniques de Gestion
Filières : GI /ARI - S3

JAVASCRIPT

Pr. M. OUTANOUTE

Année Universitaire : 2023/2024

JAVASCRIPT

- JavaScript est un langage de programmation de scripts principalement utilisé dans les pages web interactives (comme par exemple, la validation de données d'un formulaire).
- Le code JavaScript est interprété côte client.
- Les scripts ne sont pas obligatoirement exécutés au chargement de la page. Ils sont lancés lorsqu'un événement spécifique se produit.
- Voici quelques exemples (on pourrait en citer beaucoup d'autres) de ce que l'on peut en faire dans une page Web :
  - ouvrir des *pop-up* (les petites fenêtres qui s'ouvrent de manière intempestive)
  - faire défiler un texte
  - insérer un menu dynamique (qui se développe au passage de la souris)
  - proposer un diaporama (changement d'image toute les X secondes, boutons pour mettre en pause, aller à l'image précédente / suivante, etc.)
  - avoir une horloge "à aiguilles" (avec la trotteuse=aiguille des secondes)
  - faire en sorte que des images suivent le pointeur de la souris
  - créer de petits jeux
  - insérer des balises du zCode (les balises qui apparaissent en cliquant sur le bouton)
  - ...

JAVASCRIPT

- L'élément HTML **<script>** permet d'intégrer du code JavaScript dans une page.
  - Déclaration de fonctions dans l'en-tête HTML (entre **<head>** et **</head>**)
  - Appel d'une fonction ou exécution d'une commande JavaScript dans **<body>** et**</body>**
  - Insertion d'un fichier externe (usuellement '**.js**')
- Utilisation dans une URL, en précisant que le protocole utilisé est du JavaScript, ex. :

```
<a href="javascript:instructionJavaScript;">Texte</a>
```

- Utilisation des attributs de balise pour la gestion événementielle :

```
<balise onEvenement="instructionJavaScript">...</balise>
```

- Les attributs de l'élément **<script type="" src="" charset="" defer="">** sont :
  - **type** : indique le type du contenu. La valeur est typiquement "**text/javascript**".
  - **src** (optionnel) : indique que le code se situe dans un fichier externe.
  - **charset** (optionnel) : indique le jeu de caractères utilisé.
  - **defer** (optionnel) : indique si l'exécution du script doit être décalée.

JAVASCRIPT

Inline Code

- Il suffit d'utiliser uniquement l'attribut **type** et de placer le code au cœur de l'élément **script**.

```
<body>
<script type="text/javascript">
                function sayHi() {
                                alert("Hi!");
                }
</script>
```

Fichier Externe

- Il suffit d'utiliser uniquement l'attribut **type** avec l'attribut **src**.

```
<head>
        <title> Example </title>
        <script type="text/javascript" src="example1.js"> </script>
        <script type="text/javascript" src="example2.js"> </script>
...
```

- Pour exécuter le code après le chargement de la page, on utilisera :

```
<body>
        <!– le contenu ici-->
        <script type="text/javascript" src="example1.js"> </script>
        <script type="text/javascript" src="example2.js"> </script>
</body>
```


Exemple

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple de page XHTML contenant du Javascript</title>
    <script type="text/javascript">
          function fenetre() { alert('Message d alerte dans une fonction.');}
    </script>
  </head>
  <body onload="alert('Message d alerte genere a la fin du chargement.')">
        <script type="text/javascript">
              alert('Message d alerte dans le corps du document.');
        </script>
        <p>
              Ceci est le corps du document.
              <a href="javascript:fenetre()">Message d'alerte</a>.
        </p>
    </body>
</html>
```

Exemple

Page statique

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page statique</title>
      </head>
      <body>
            <div>
              Nous sommes le 31/10/2017
            </div>
      </body>
</html>
```

Page dynamique JavaScript

```
<!DOCTYPE html>
<html>
  <head>
        <title>Page dynamique</title>
  </head>
  <body>
     <scripttype = "text/javascript ">

        date = new Date();
        document.write("Nous sommes le ", date);

     </script>
  </body>
</html>
```


Bases du langage

- **Variable** : Déclarer une variable nécessite l'utilisation du mot-cle **var**.
- **Les identificateurs**: sont sensibles à la casse: **test** n'est pas **Test**
- **Les instructions** :
  - se terminent par un point-virgule **;**
  - nécessitent des accolades lorsqu'elles forment un bloc **{…}**
- **Les commentaires**
  - **//** commentaire d'un seule ligne
  - **/***

    ***** commentaire

    *multi-ligne

    ***/**

Entrée et sortie

- 3 types de boites de messages peuvent être affichés en utilisant JavaScript :

  Alerte, **Confirmation** et **Invite**

  - **Méthode alert()**
    - sert à afficher à l'utilisateur des informations simples de type texte. Une fois que ce dernier a lu le message, il doit cliquer sur OK pour faire disparaître la boîte.
  - **Méthode confirm()**
    - permet à l'utilisateur de choisir entre les boutons OK et Annuler.
  - **Méthode prompt()**
    - permet à l'utilisateur de taper son propre message en réponse à la question posée.

- La méthode **document.write** permet d 'écrire du code HTML dans la page WEB

Exemple

```
<html>
   <head>
        <title> une page simple </title>
   </head>
   <body>
        Bonjour
        <script  type = "text/javascript">
          alert('bonjour');
          document.write (prompt('quel est votre nom ?','Indiquer votre nom ici') );
          confirm('quel bouton allez-vous choisir ?');
        </script>
   </body>
</html>
```

Les variables

- Le mot-clé ***var*** permet de déclarer une ou plusieurs variables.
- Après la déclaration de la variable, il est possible de lui affecter une valeur par l'intermédiaire du signe d'égalité (=).
- Si une valeur est affectée à une variable sans que cette dernière ne soit déclarée, alors JavaScript la déclare automatiquement.

```
//Déclaration de i, de j et de k
var i, j, k;

 //Affectation de i
i = 1;
//Déclaration et affectation de prix
 var prix = 0;
//Déclaration et affectation de caractere
var caractere = ["a", "b", "c"];
```

- La lecture d'une variable non déclarée provoque une erreur.
- Une variable correctement déclarée mais dont aucune valeur n'est affectée, est indéfinie (undefined).

Exemple

- les variables peuvent être globales ou locales.
- Une variable globale est déclarée en début de script et est accessible à n'importe quel endroit du programme.
- Une variable locale est déclarée à l'intérieur d'une fonction et n'est utilisable que dans la fonction elle-même.

Portée locale

```
function test() {
        var message = "hi";
}
test();
alert(message); // undefined
```

Portée globale

```
var message;
function test() {
        message = "hi";
}
test();
alert(message); // "hi"
```

Type

- Il y a principalement trois types classiques de données (**Boolean**, **Number** et **String**) et un type complexe **Object** (liste de couples nom-valeurs).
  Il est possible de déterminer le type (courant) d'une variable avec l'operateur **typeof** qui retourne l'une des valeurs suivantes:
  - **undefined** : si la valeur est indéfinie.
    (variable déclarée mais pas initialisée ou variable non déclarée)
  - **Boolean** : true (vrai) et false (faux)
  - **Number** : Les nombres entiers, les nombres décimaux en virgule flottant
  - **string**
  - **object** : si la valeur est un **objet** ou **null**
  - **function** : si la valeur est une **fonction**

Type : Valeurs undefined et null

- La valeur **undefined** : fait référence à une variable non déclarée ou déclarée mais pas initialisée,
- La valeur **null** : signifie l'absence de données dans une variable

```
var message = "coucou";
alert (typeof message); // "string"
var m1;
alert(m1); // "undefined"
alert(m2); // erreur
alert(typeof m1); // "undefined"
alert(typeof m2); // undefined
...
var car = null;
alert(typeof car); // "object"
```

Type : Boolean et Number

- Le type **Boolean** a deux valeurs **true** et **false**.
- Le type **Number** sert à représenter des valeurs entières et des valeurs flottantes (réelles).

```
var x = 55;
var y = 070;          //octal pour 56 (en base 10)
var z = 0xA,          //hexadécimal pour 10
var f1 = 10.2;
var f2 = 3.12e7;      // représente 31 200 000
var f3 = 3e-10;       // représente 0,0000000003
```

- **Remarque :** Les valeurs particulières du type **Number** sont :
  - **Number.MIN_VALUE**, **Number.MAX_VALUE**
  - **Number.NEGATIVE_INFINITY**, **Number.POSITIVE_INFINITY**
  - **NaN** (Not a Number)

Type : Conversions numeriques

- Relatifs aux valeurs et conversions des nombres, on trouve 4 fonctions :
  - **isNaN** détermine si un paramètre donné n'est pas un nombre
  - **Number** effectue une conversion
  - **parseInt** effectue une conversion en valeur entière
  - **parseFloat** effectue une conversion en valeur réelle

```
alert(isNaN(10));                    // false
alert(isNaN("10"));                  // false - peut être convertie
alert(isNaN("blue"));                // true - ne peut être convertie
var num1 = Number("hello world");    // NaN
var num2 = Number("00001");          // 1
var num3 = Number(true);             // 1
var num4 = parseInt("");             // NaN
var num5 = parseInt(22.5);           // 22
var num6 = parseInt("70",10);        // 70 - la base 10 est spécifiée
var num7 = parseFloat("22.5");       // 22.5
alert(isNaN(num1));                  // true
alert(isNaN(num2));                  // false
alert(isNaN(num3));                  // false
alert(isNaN(num4));                  // true
alert(isNaN(num5));                  // false
alert(isNaN(num6));                  // false
alert(isNaN(num7));                  // false
```

Type : String

- On utilise les quottes simples (**apostrophes '...'**) ou doubles (**guillemets "..."**) pour définir des valeurs chaînes de caractères. L'attribut **length** permet de déterminer la longueur d'une chaîne.

```
var nom = "Wilde";
var prenom = 'Oscar';
alert(prenom.length);                    // 5
var message = "toto a dit \"je suis malade\".";
```

- Il est possible de **transtyper** une valeur en chaîne avec **String().**

```
var v1 = 10; alert(String(v1));       // "10"
var v2 = 13.25; alert(String(v2));    // "13.25"
var v3 = true; alert(String(v3));     // "true"
var v4 = null; alert(String(v4));     // "null"
```


Type : String

- Il existe de nombreuses fonctions sur les chaînes de caractères.

```
var s = "hello world";
alert(s.length);                    // 11
alert(s.charAt(1));                 // "e"
alert(s.charCodeAt(1));             // 101
alert(s.slice(3));                  // "lo world"
alert(s.slice(-3));                 // "rld"
alert(s.substring(3,7));            // "lo w"
alert(s.indexOf("o"));              // 4
alert(s.lastIndexOf("o"));          // 7
alert(s.toUpperCase());             // HELLO WORLD
alert(s + " !");                    // hello world !
var hi = 'Bonjour', name = 'toi', result;
result = hi + name;
alert(result);                      //  Bonjourtoi
```

Type : Math

- Il s'agit d'un objet définissant de nombreuses constantes et fonctions mathématiques.

```
alert(Math.E);                  // la valeur de e
alert(Math.PI);                 // la valeur de pi
alert(Math.min(5,12));          // 5
alert(Math.max(23,5,7,130,12)); // 130
alert(Math.ceil(25.3));         // 26
alert(Math.floor(25.8));        //25
alert(Math.random());           // valeur aleatoire entre 0 et 1
var n = Math.floor(Math.random()*nb + min);
alert(n);                       // valeur aleatoire entre min et min+nb (exclus)
```

- D'autres fonctions :

Math.abs(x) **Math.exp(x)** **Math.log(x)**

Math.pow(x,y) **Math.sqrt(x)**

Math.sin(x) **Math.cos(x)** **Math.tan(x)**

Operateurs

- Typiquement, ceux de C, C++ et java:
  - incrémentation/décrémentation (++, --)
  - arithmétiques (+-,*,/,%)
  - relationnels (>,<,>=,<=,==,! =) et (===,! ==)
  - logique (!,&&,||)
  - affectation (=,+=,-=,*=,/=,%=)
  - bit a bit (&,|,<<,>>, ...)

```
var age = 10;
age++;
alert(age);                    // 11
alert(age > 10 && age < 20);   // true
alert(26 % 5);                 // 1
age*=2;
alert(age);                    // 22
```

Structures de contrôle

- Elles sont très proches de celles de langages tels que C, C++ et Java.

  Pour rappel, les structures de contrôles sont de trois types :

  - **Séquence** : exécution séquentielle d'une suite d'instructions séparées par un point-virgule
  - **Alternative** : structure permettant un choix entre divers blocs d'instructions suivant le résultat d'un test logique
  - **Boucle** : structure itérative permettant de répéter plusieurs fois le même bloc d'instructions tant qu'une condition de sortie n'est pas avérée

- Toute condition utilisée pour une alternative ou boucle sera toujours placée entre parenthèses.
- Il ne faut pas confondre x == y (test d'égalité) avec x = y (affectation).

Structures de contrôle : Alternative

- L'instruction **if** sans partie **else** :

```
if (condition) instruction;
```

```
if (condition) {
        instruction;
}
```

```
if (condition) {
        instruction1;
        instruction2;
        ...
}
```

```
if (x >= 0) alert("valeur positive ou nulle");
...
if (note > 12 && note <= 14) {
      alert("bravo");
      mention="bien";
}
```

Structures de contrôle : Alternative

- L'instruction **if**...**else** :

```
if (condition) instruction1;
else instruction2;
```

```
if (condition) {
        instructions1;
} else {
        instructions2;
}
```

```
if (condition1) {
        instructions1;
} else if (condition2) {
        instructions2;
} else {
        instructions3;
}
```

```
if (rank == 1)
        medaille="or";
else if (rank == 2)
        medaille="argent";
else if (rank == 3)
        medaille="bronze";
```


Structures de contrôle : Alternative

- **L'instruction switch** :

```
switch (expression) {
   case valeur1 :
        instructions1;
        break;
   case valeur2 :
        instructions2;
        break;
   ...
   case valeurN :
        instructionsN;
        break;
   default:    instructionsDefault;
}
```

- **Remarque :** Le branchement par default n'est pas obligatoire.

Structures de contrôle : Alternative

- L'opérateur ternaire **Condition?Expression1:Expression2** permet de remplacer une instruction **if...else** simple.

  Sa syntaxe (lorsqu'utilisée pour affecter une valeur à une variable) est :

```
variable = condition ? expressionIf : expressionElse;
```

- Elle est équivalente a :

```
if (condition) variable=expressionIf;
else variable=expressionElse;
```

```
var civilite = (sexe == "F") ? "Madame" : "Monsieur";
```

- Cet opérateur est utile pour les expressions courtes.

Structures de contrôle : Boucle

- **L'instruction while :**

```
while (condition) instruction;

while (condition) {
   instruction1;
   instruction2; ...
}
```

```
var num = 1;
while (num <= 5)
{
    alert(num);
    num++;
}
```

Structures de contrôle : Boucle

- **L'instruction for :**

```
for (instruction-Initial; condition; instruction-Iteration) instruction;

for (instruction-Initial; condition; instruction-Iteration) {
        instruction1;
        instruction2; ...
}
```

```
for (var num = 1; num <= 5; num++)
        alert(num);
```

- Il n'est pas nécessaire de placer des parenthèses autour de la condition pour le for.

Structures de contrôle : Boucle

- **L'instruction do...while :**

```
do{
        instruction1;
        instruction2;
        ...
}while(condition);
```

- **L'instruction for-in** pour les objets :

```
for (variable in objet) {
        instructions
}
```

variable
Un nom de propriété différent est assigné à la variable à chaque itération de la boucle.

objet
L'objet dont les propriétés énumérables et qui ne sont pas des symboles sont parcourues par la boucle.

Structures de contrôle : Boucle

- Certaines instructions permettent un contrôle supplémentaire sur les boucles :
  - **break** permet de quitter la boucle courante
  - **continue** permet de terminer l'itération en cours de la boucle courante.

```
for (var i=0; i<5; i++) {
        for (var j=0; j<5; j++) {
                if ((i+j)%3 == 0)   continue;
                for (var k=0; k<5; k++) {
                        if ((i+j+k)%3 == 0)   break;
                        alert(i + " " + j + " " + k);
                }
        }
}
```


Exercice sur les conditions :

Fournir un commentaire selon l'âge de la personne.

Vous devez fournir un commentaire sur 4 tranches d'âge qui sont les suivantes :

Tranche d'âge	Commentaire
1 à 6 ans	« Vous êtes un jeune enfant. »
7 à 11 ans	« Vous êtes un enfant qui a atteint l'âge de raison. »
12 à 17 ans	« Vous êtes un adolescent. »
18 à 120 ans	« Vous êtes un adulte. »

```
<script>
        var age = parseInt(prompt('Quel est votre âge ?'));
        if (1 <= age && age <= 6) {
                alert('Vous êtes un jeune enfant.');
        } else if (7 <= age && age <= 11) {
                alert ('Vous êtes un enfant qui a atteint l\'âge de raison.');
        } else if (12 <= age && age <= 17) {
                alert ('Vous êtes un adolescent.');
        } else if (18 <= age && age <= 120) {
                alert ('Vous êtes un adulte.');
        } else {
                alert ('Erreur !!');
        }
</script>
```

Types : Object

- Un objet est une collection de paire **nom/valeur** utilisée pour stocker des données.
- Création d'une variable (de type Object) avec **new Object()** ou de manière littérale (énumération entre accolades).

```
var person = new Object();
person.name = "Ali";
person.age = 23;
```

```
var person = {
        name : "Ali",
        age : 23
}
```

- Il est possible d'utiliser les crochets pour accéder à un champ.

```
alert(person.name);
alert(person["name"]);
var field="name";
alert(person[field]);
```


JAVASCRIPT

Types : Object

- On ne peut pas parcourir l'objet avec for, car **for** s'occupe d'incrémenter des variables numériques.

```
var family = {
        self: 'Ali',
        sister: 'Amal',
        brother: 'Ahmed',
        cousin_1: 'Brahim',
        cousin_2: 'Anass‘
};

var id = 'sister';
alert(family[id]);              // Affiche : « Amal »
alert(family.brother);          // Affiche : « Ahmed »
alert(family['self']);          // Affiche : « Ali »
family['uncle'] = 'Salim';      // On ajoute une donnée, avec un identifiant.
family.aunt = 'Karim';          // On peut ajouter aussi de cette manière.
alert(family.uncle);            // Affiche : « Salim »


for (var id in family) {        // On stocke l'identifiant dans « id » pour parcourir l'objet « family »
    alert(id+ ":" + family[id]);
}
```

Types : Array

- Les tableaux peuvent contenir des données de natures différentes.

```
var colors1 = new Array();                          // tableau vide
var colors2 = new Array(20);                        // tableau avec 20 cases
var colors3 = new Array("red","blue","green");      // 3 cases
var colors4 = ["red","blue","green"];               // notation littérale
```

- Le champ **length** indique la taille d'un tableau.

```
var colors = ["red","blue","green"];
alert(colors.length);                   // 3
colors[colors.length]="black";          // nouvelle couleur
colors[99]="pink";
alert(colors.length);                   // 100
alert(colors[50]):                      // undefined
colors.length=10;                       // undefined
```


Types : Array

- De nombreuses méthodes existent sur les tableaux :
  - On peut ajouter des items avec la méthode **push()** ;
  - La méthode **unshift()** fonctionne comme **push()**, excepté que les items sont ajoutés au début du tableau ;
  - Les méthodes **shift()** et **pop()** retirent respectivement le premier et le dernier élément du tableau ;
  - On peut découper une chaîne de caractères en tableau avec **split()** ;
  - On fait l'inverse avec **join()** ;

Types : Array

```
var myArray = ['Ali', 'Mouhssin'];
myArray.push('Ahmed');                  // Ajoute « Ahmed » à la fin du tableau
myArray.push('Driss', 'Amal');          // Ajoute « Driss » et « Amal » à la fin du tableau
alert(myArray);

myArray.shift();                        // Retire « Ali»
myArray.pop();                          // Retire « Amal »
alert(myArray);

var uneString = 'casa rabat kenitra',
unArray = uneString.split(' ');         // Avec les espaces, on a trois items
alert(uneString);
alert(unArray);

var uneString_2 = unArray.join('***');
alert(uneString_2);
```


Array : Exercice

- Demandez les prénoms aux utilisateurs et stockez-les dans un tableau. Pensez à la méthode **push()**. À la fin, il faudra afficher le contenu du tableau, avec **alert()**, seulement si le tableau contient des prénoms.
  Pour l'affichage, séparez chaque prénom par un espace (**méthode join()**).
  Si le tableau ne contient rien, faites-le savoir à l'utilisateur, toujours avec **alert().**

```
var prenoms = [ ],prenom;
while (prenom = prompt('Entrez un prénom :')) {
        prenoms.push(prenom);
}
if (prenoms.length > 0) {
        alert(prenoms.join(' '));
} else {
        alert('Il n\'y a aucun prénom en mémoire');
}
```

Types : Array

- On peut parcourir un tableau avec for :

```
<script>
    var myArray = ['Ali', 'Mouhssin', 'Ahmed', 'Driss', 'Amal']; // La length est de 5
    alert(myArray);
    // On crée la variable c pour que la condition ne soit pas trop lourde en caractères
    for (var i = 0, c = myArray.length; i < c; i++) {
    // On affiche chaque item, l'un après l'autre, jusqu'à la longueur totale du tableau
        alert(myArray[i]);
    }
</script>
```


Types : Réordonner les tableaux

- On peut utiliser **reverse** et **sort** :

```
var values = [0, 1, 2, 3, 4];
alert(values.reverse());                    // 4,3,2,1,0
values = [0, 1, 5, 10, 15];
alert(values.sort());                       // 0,1,10,15,5
function compare(a, b) {
    return a - b;
}
alert(values.sort(compare));                // 0,1,5,10,15
```

Fonctions

- La syntaxe pour définir une fonction est :

```
function name(arg1, arg2, ..., argN) {
        instructions
}
```

```
function sayHi(name) {
        alert("Hello " + name);
}

sayHi("Ahmed");     // Hello Ahmed
```

```
function sum(num1, num2) {
        return num1+num2;
}

alert(sum(5,10));     //15
```

- Une fonction peut retourner un résultat avec l'instruction **return** même si rien ne l'indique au niveau de la signature (en-tête) de la fonction.

Arguments

- Il existe toujours un tableau arguments implicite lors de l'appel d'une fonction:

```
function f() {
        alert(arguments.length);
        for (var i=0; i<arguments.length; i++)
                alert(arguments[i]);
}
f("Ahmed");                 // affiche: 1 Ahmed
f("Ahmed",25);              // affiche: 2 Ahmed 25
```

- La possibilité de faire appel à une fonction avec un nombre variable de paramètres pallie l'impossibilité de surcharge (overloading).

L'OBJET DATE

- L'objet Date en javascript permet de manipuler la date et l'heure
- Syntaxe:

Variable = new Date();

- Quelques méthodes:
- getYear(): retourne l'année sur deux chiffres
- getFullYear(): retourne l'année sur 4 chiffres
- getDate(): retourne le jour du mois compris entre 1 et 31
- getDay (): retourne un indice qui représente le jour de la semaine
- getMonth (): retourne un indice qui représente le mois
- getHours (): retourne l'heure
- getMinutes (): retourne les minutes
- getSecondes (): retourne les secoundes

EXEMPLE 1

```
<h3>Affichage de la date d'aujourd'hui</h3>
<script>
jour=new Array("Dimanche","Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi");
mois=new Array("janvier","février","mars","avril","mai","juin","juillet","août",
"septembre","octobre","novembre","décembre");
d=new Date();
aujourdhui=jour[d.getDay()]+ " "+ d.getDate()+" "+ mois[d.getMonth()]+" "+ d.getFullYear();
document.write('Nous sommes le :'+aujourdhui);
</script>
```

EXEMPLE 2

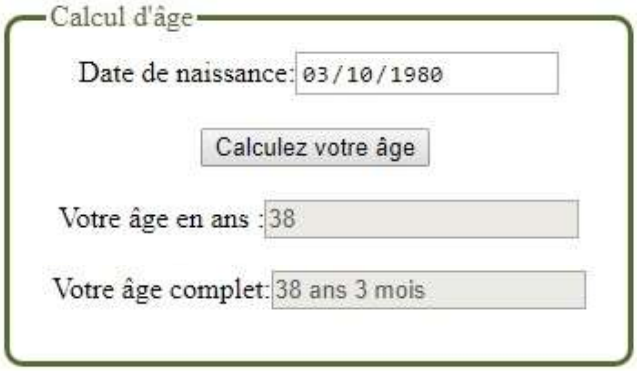
```
<script>
var now = new Date();
var annee   = now.getFullYear();
var mois    = ('0'+(now.getMonth()+1)).slice(-2);
var jour    = ('0'+now.getDate()   ).slice(-2);
var heure   = ('0'+now.getHours()  ).slice(-2);
var minute  = ('0'+now.getMinutes()).slice(-2);
var seconde = ('0'+now.getSeconds()).slice(-2);

document.write("Nous sommes le "+jour+"/"+mois+"/" +annee+"
et il est "+heure+" heure "+minute +" minutes "+seconde+" secondes" );

</script>
```


EXERCICE

Exercice 01

1. Énoncé

- Écrire une page **html5** qui contient:
- Une zone de texte de type date
- Un bouton dont la valeur est « Calculez votre âge »
- Une zone de texte pour afficher l'age en année
- Une zone de texte pour afficher l'age compmlet
- Réaliser un script qui permet de calculer l'âge en fonction de la date de naissance rempli par l'utilisateur.
- Traiter les cas des zéros pour les mois et les jours seulement
- Le résultat final doit correspondre au schémas suivant:



LES ÉVÉNEMENTS

- Les événements permettent de déclencher une fonction selon qu'une action s'est produite ou non
- Un évènement est provoqué par une action de l'utilisateur ou du navigateur lui-même.

  Les évènements ont des noms tels que **click**, **load** et **mouseover**.
- Une fonction appelée en réponse à un évènement se nomme un **écouteur** (**event handler** ou **event listener**).

  Souvent, leurs noms commencent par **on** comme par exemple **onclik** ou **onload.**
- Associer des écouteurs aux évènements possibles peut se faire de trois manières différentes:
  - HTML
  - DOM Level 0
  - DOM Level 2

LES ÉVÉNEMENTS

HTML Event Handlers

- On utilise des attributs HTML pour déclarer les écouteurs. La valeur de ces attributs est le code JavaScript à exécuter lorsque l'évènement est produit.

```
<input type="button" value="but1" onclick="alert('clicked')" />
<script type="text/javascript">
        function showMessage() {
                alert("hello world");
        }
</script>
<input type="button" value="but2" onclick="showMessage()" />
```

Dom Level 0 Event Handlers

- On utilise les propriétés des éléments pour leur associer des écouteurs.

```
<input type="button" id="b1" value="bouton 1" />
<script type="text/javascript">
        function but1Listener() {
                alert("button1 cliqued");
        }
        var but1 = document.getElementById("b1");
        but1.onclick=but1Listener;
</script>
```


L'objet event

- Quand un évènement se produit, toutes les informations le concernant sont enregistrées dans un objet appelé **event**.
  Il est possible de récupérer cet objet sous forme de paramètre d'une fonction écouteur.

```
<input type="button" id="but3" value="bouton 1" />
<script type="text/javascript">
    function but3Listener(event) {
        switch(event.type) {
            case "click" : alert("clicked at " + event.clientX + " " + event.clientY); break;
            case "mouseover" : this.style.backgroundColor="red"; break;
            case "mouseout" : this.style.backgroundColor="green"; break;
        }
    }
    but3.onclick=but3Listener;
    but3.onmouseover=but3Listener;
    but3.onmouseout=but3Listener;
</script>
```

Exemples d'événements

- **click** Cliquer (appuyer puis relâcher) sur l'élément
- **dblclick** Double-cliquer sur l'élément
- **mouseover** Faire entrer le curseur sur l'élément
- **mouseout** Faire sortir le curseur de l'élément
- **mousedown** Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
- **mouseup** Relâcher le bouton gauche de la souris sur l'élément
- **mousemove** Faire déplacer le curseur sur l'élément
- **keydown** Appuyer (sans relâcher) sur une touche de clavier sur l'élément
- **keyup** Relâcher une touche de clavier sur l'élément
- **keypress** Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
- **focus** « Cibler » l'élément
- **blur** Annuler le « ciblage » de l'élément
- **change** Changer la valeur d'un élément spécifique aux formulaires (input, checkbox, etc.)
- **select** Sélectionner le contenu d'un champ de texte (input, textarea, etc.)

Attribut	Propriété associée	Description
onAbort	**onabort**	Est déclenché lorsque l'utilisateur interrompt le chargement d'une image.
onBlur	**onblur**	Est déclenché lorsqu'un objet perd le focus.
onClick	**onclick**	Est déclenché lorsque l'utilisateur clique sur un objet.
onChange	**onchange**	Est déclenché lorsque l'utilisateur a changé la valeur d'un champ de saisie.
onDbIClick	**ondblclick**	Est déclenché lorsque l'utilisateur double-clique sur un objet.
onDragDrop	**ondrapdrop**	Est déclenché lorsque l'utilisateur fait glisser un objet dans le browser.
onError	**onerror**	Est déclenché lorsqu'une erreur se produit lors du chargement d'une image ou de l'exécution de code JavaScript.
onFocus	**onfocus**	Est déclenché lorsqu'un objet reçoit le focus.
onKeyDown	**onkeydown**	Est déclenché lorsque l'utilisateur enfonce une touche.
onKeyPress	**onkeypress**	Est déclenché lorsque l'utilisateur a appuyé sur une touche.
onKeyUp	**onkeyup**	Est déclenché lorsque l'utilisateur relâche une touche.
onLoad	**onload**	Est déclenché lorsqu'un document (ou une image) commence à se charger.
onMouseDown	**onmousedown**	Est déclenché lorsque l'utilisateur enfonce un des boutons de la souris.

onMouseMove	**onmousemove**	Est déclenché lorsque l'utilisateur déplace la souris.
onMouseOut	**onmouseout**	Est déclenché lorsque la souris vient de passer au-dessus d'un lien, d'une ancre ou d'une zone d'une carte cliquable et qu'elle retourne sur le fond de la fenêtre.
onMouseOver	**onmouseover**	Est déclenché lorsque la souris passe au-dessus d'un lien, d'une ancre ou d'une zone d'une carte cliquable.
onMouseUp	**onmouseup**	Est déclenché lorsque l'utilisateur relâche un des boutons de la souris.
onMove	**onmove**	Est déclenché lorsqu'une fenêtre est déplacée.
onReset	**onreset**	Est déclenché au moment où l'utilisateur annule les données saisies dans un formulaire.
onResize	**onresize**	Est déclenché lorsqu'une fenêtre ou une frame est redimensionnée.
onSelect	**onselect**	Est déclenché lorsque l'utilisateur sélectionne du texte dans un champ de saisie.
onSubmit	**onsubmit**	Est déclenché au moment où un formulaire est soumis.
onUnload	**onunload**	Est déclenché au moment où un document se décharge (c'està-dire lors du chargement d'un nouveau document).

ÉVÉNEMENTS

Les objets auxquels peuvent être associés chaque événement

Evénements	Objets concernés
abort	Image
blur	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, window
change	FileUpload, Select, Submit, Text, TextArea
click	Button, document, Checkbox, Link, Radio, Reset, Select, Submit
dblclick	document, Link
dragdrop	window
error	Image, window
focus	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, window
keydown	document, Image, Link, TextArea
keypress	document, Image, Link, TextArea
keyup	document, Image, Link, TextArea

ÉVÉNEMENTS

keyup	document, Image, Link, TextArea
load	Image, Layer, window
mousedown	Button, document, Link
mousemove	Aucun spécifiquement
mouseout	Layer, Link
mouseover	Area, Layer, Link
mouseup	Button, document, Link
move	window
reset	form
resize	window
select	text, Textarea
submit	Form
unload	window
Source du tableau:www.commentcamarche.net	

LES ÉVÉNEMENTS

Exemples d'événements

- **Deux événements spécifiques à <form> :**
  - **submit :** Envoyer le formulaire
  - **reset :** Réinitialiser le formulaire

```
<span onclick="alert('Hello !');">Cliquez-moi !</span>
```

```
<input id="input" type="text" size="50" value="Cliquez ici !"
    onfocus="this.value='Appuyez maintenant sur votre touche de tabulation.' ; "
    onblur="this.value='Cliquez ici !' ; " />
```

- Le Javascript est un langage qui permet de créer ce que l'on appelle des *pages DHTML*. Ce terme désigne les pages Web qui modifient elles-mêmes leur propre contenu sans charger de nouvelle page.

- Le **DOM** (*Document Object Model*) est un programme interne à notre document HTML qui fait en sorte que:
  - **chaque élément** (balises <h1>, <body>, <p>, ...),
  - **chaque événement** (clic de la souris, chargement de la page, ...),
  - **chaque attribut** (href, alt, title, ...)

  est **RÉCUPÉRABLE**, **MANIPULABLE** et **EXPLOITABLE** par un langage de programmation.

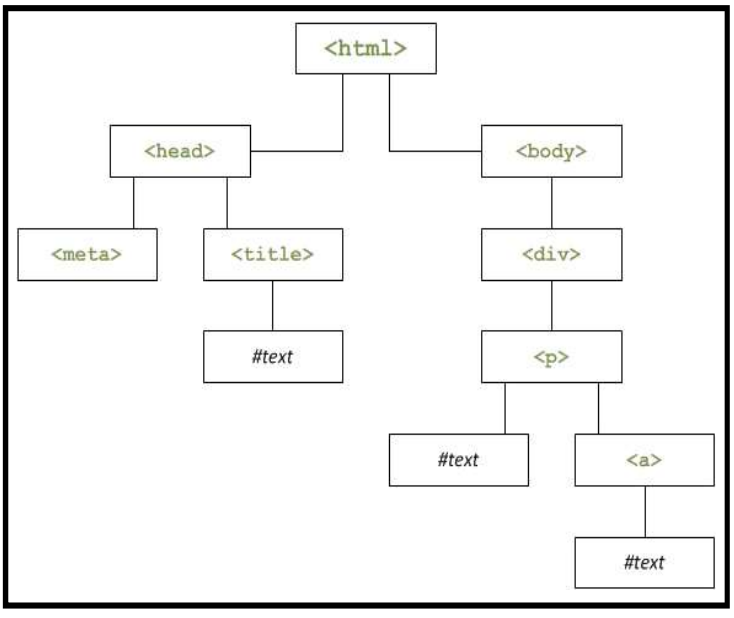
- Le Javascript va récupérer chaque "**composite**" de notre document HTML et l'exploiter.

- L'**API** (*Application Programming Interface*) **DOM** fait en sorte que tout ce qui compose notre document HTML soit un **objet** qui peut être récupéré et exploité par le Javascript (ou un autre langage de programmation).

MODELER VOS PAGES WEB

La structure DOM

- Le **DOM** pose comme concept que la page Web est vue comme un arbre, comme une hiérarchie d'éléments.




```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Le titre de la page</title>
    </head>
    <body>
        <div>
            <p>Un peu de texte
                <a>et un lien</a>
            </p>
        </div>
    </body>
</html>
```

L'objet window

- L'objet **window** est ce qu'on appelle un ***objet global*** qui représente la fenêtre du navigateur.
- Contrairement à ce qui a été vu avant, **alert()** n'est pas vraiment une fonction, mais une méthode qui appartient à l'objet **window**, qui est implicite (il y a en fait très peu de variables globales).
  Les deux lignes suivantes signifient la même chose :

```
<script>
        alert('Hello world !');
        window.alert('Hello world !');
        var text = 'Variable locale !';
        alert(text); // Forcément, la variable locale prend le dessus
        alert(window.text);
</script>
```


MODELER VOS PAGES WEB

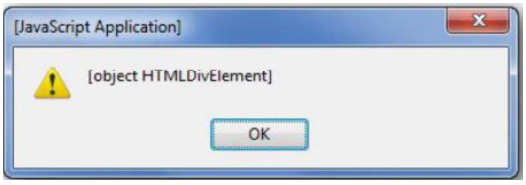
L'objet Document

- L'objet **document** est un sous-objet de **window**.
- Pour l'accès aux éléments HTML via le DOM, l'objet document possède trois méthodes principales :
  - **getElementById()**,
  - **getElementsByTagName()**,
  - **getElementsByName()**.
- C'est grâce à cet objet-là que nous allons pouvoir accéder aux éléments HTML et les modifier.

La méthode getElementById()

- Cette méthode permet d'accéder à un élément en connaissant son ***ID*** qui est simplement l'***attribut id*** de l'élément.

```
<div id="myDiv">
    <p>Un peu de texte<a>et un lien</a></p>
</div>
<script>
    var div = document.getElementById('myDiv');
    alert(div);
</script>
```



- Il nous dit que **div** est un objet de type **HTMLDivElement**.
  En clair, c'est un élément HTML qui se trouve être un ** **, ce qui nous montre que le script fonctionne correctement.

La méthode getElementsByTagName()

- Cette méthode permet de récupérer, sous la forme d'un tableau, tous les éléments de la famille.
  Si, dans une page, on veut récupérer tous les ** **, il suffit de faire comme ceci :

```
<div id="myDiv1">
        <p> paragraphe 1 </p>
</div>
<div id="myDiv2">
        <p> paragraphe 2 </p>
</div>
<script>
     var divs = document.getElementsByTagName('div');
      for (var i = 0, c = divs.length ; i < c ; i++) {
            alert('Element n° ' + (i + 1) + ' : ' + divs[i]);
      }
</script>
```

- On parcourt le tableau avec une boucle pour récupérer les éléments.

La méthode getElementsByName()

- Cette méthode est semblable à **getElementsByTagName()** et permet de ne récupérer que les éléments qui possèdent un **attribut name** que vous spécifiez.
- L'attribut **name** n'est utilisé qu'au sein des formulaires.

```
<form name="up">
        <input type="text">
</form>
<div name="down">
        <input type="text">
</div>
<script>
        var up_forms = document.getElementsByName("up");
        alert(up_forms[0].tagName);
</script
```


MODELER VOS PAGES WEB

- **querySelector()** : renvoie le premier élément trouvé correspondant au sélecteur CSS spécifié.
- **querySelectorAll()** : renvoie tous les éléments (sous forme d'un tableau) correspondant au sélecteur CSS spécifié entre parenthèses.
- **innerHTML** permet de récupérer le code HTML enfant d'un élément en texte.
- Essayons le sélecteur CSS suivant: **#menu .item span**

```
<div id="menu">
    <div class="item">
        <span>Élément 1</span><span>Élément 2</span>
    </div>
    <div class="publicite">
        <span>Élément 3</span><span>Élément 4</span>
    </div>
</div>
<div id="contenu">
    <span>Introduction au contenu de la page...</span>
</div>
<script>
    var query = document.querySelector('#menu .item span'),
    var queryAll = document.querySelectorAll('#menu .item span');
    alert(query.innerHTML);                                          // Élément 1
    alert(queryAll.length);                                          // 2
    alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML);    // Élément 1 - Élément 2
</script>
```

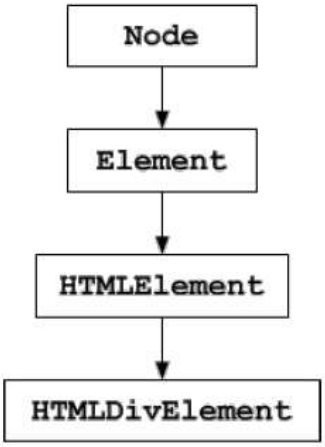
L'héritage des propriétés et des méthodes

- On peut jouer sur les attributs d'une balise HTML avec l'objet **Element** et les méthodes **getAttribute()** et **setAttribute(),** permettant par exemple de modifier un lien :

```
<a id="lien" href="http://www.estbm.ac.ma">Un lien modifié dynamiquement</a>
<script>
    var link = document.getElementById('lien');
    var href = link.getAttribute('href');          // On récupère l'attribut « href »
    alert(href);                                   // Affiche: http://www.estbm.ac.ma
    link.setAttribute('href', 'http://www.google.ma');   // on édite
</script>
```


MODELER VOS PAGES WEB

- Nous avons vu qu'un élément ** ** est un objet **HTMLDivElement**, mais un objet, en Javascript, peut appartenir à différents groupes.
- Ainsi, notre ** ** est un **HTMLDivElement**, qui est un sous-objet d'**HTMLElement**
- **HTMLElement** est lui-même un sous-objet d'**Element**.
- **Element** est enfin un sous-objet de **Node**. Ce schéma est plus parlant :



ÉDITER LES ÉLÉMENTS HTML

Les attributs

- ***Via l'objet Element :***

  Pour interagir avec les attributs, l'objet Element nous fournit deux méthodes :

  - **getAttribute()** permettant de récupérer un attribut.
  - **setAttribute()** permettant d'éditer un attribut.

```
<a id="myLink" href="http://www.fs.uit.ac.ma">Un lien modifié dynamiquement</a>
<script>
        var link = document.getElementById('myLink');
        var href = link.getAttribute('href');                    // On récupère l'attribut « href »
        alert(href);
        link.setAttribute('href', 'http:www.sadiq.ma');          // on édite
</script>
```


MODELER VOS PAGES WEB

Les attributs

- ***La classe***

  - Pour modifier l'attribut class d'un élément HTML, il faudra utiliser **className**.

```
<!doctype html>
<html>
    <head>
        <title>Le titre de la page</title>
        <style type="text/css">
            .blue {
                background: blue;
                color: white;
            }
        </style>
    </head>
    <body>
        <div id="myColoredDiv">
            <p>Un peu de texte <a>et un lien</a></p>
        </div>
        <script>
            document.getElementById('myColoredDiv').className = 'blue';
        </script>
    </body>
</html>
```

Le contenu : innerHTML

- **innerHTML** permet de récupérer le code HTML enfant d'un élément sous forme de texte.

```
<body>
    <div id="myDiv">
        <p>Un peu de texte <a>et un lien</a></p>
    </div>
    <script>
        var div = document.getElementById('myDiv');
        alert(div.innerHTML);
    </script>
</body>
```

MODELER VOS PAGES WEB

Le contenu : innerHTML

- **Ajouter ou éditer du HTML :**
- Pour éditer ou ajouter du contenu HTML, il suffit de faire l'inverse, c'est-à-dire de définir un nouveau contenu :

```
document.getElementById('myDiv').innerHTML =
 '<blockquote>Je mets une citation à la place du paragraphe</blockquote>' ;
```

- Si vous voulez ajouter du contenu, et ne pas modifier le contenu déjà en place, il suffit d'utiliser **+=** à la place de l'opérateur d'affectation :

```
document.getElementById('myDiv').innerHTML +=
                    ' et <strong>une portion mise en emphase</strong>.';
```

LES FORMULAIRES

- Avec les formulaires commence l'interaction avec l'utilisateur grâce aux nombreuses propriétés et méthodes dont sont dotés les éléments HTML utilisés dans les formulaires.

- Il est possible d'accéder à n'importe quelle propriété d'un élément HTML juste en tapant son nom, il en va donc de même pour des propriétés spécifiques aux éléments d'un formulaire comme :

  value,
  disabled,
  checked,
  etc.

LES FORMULAIRES

Une propriété classique : value

- Cette propriété permet de définir une valeur pour différents éléments d'un formulaire comme les **<input>**, les **<button>**, etc.
  Cette propriété s'utilise aussi avec un élément **<textarea>**

```
<input id="text" type="text" size="60" value="Vous n'avez pas le focus !" />
<script>
        var text = document.getElementById('text');
        text.addEventListener('focus', function(e) {
                e.target.value = "Vous avez le focus !"; } );
        text.addEventListener('blur', function(e) {
                e.target.value = "Vous n'avez plus le focus !"; } );
</script>
```

Les booléens avec disabled, checked et readonly

- Chaque bouton radio coché se verra attribuer la valeur **true** à sa propriété **checked**, il va donc nous falloir utiliser une boucle **for** pour vérifier quel bouton radio a été sélectionné :

```
<fieldset>
        <legend>Produit</legend>
        <label><input type="radio" name="check" value="1" /> Pomme</label><br />
        <label><input type="radio" name="check" value="2" /> Banane</label><br />
        <label><input type="radio" name="check" value="3" /> Fraise</label><br />
        <label><input type="radio" name="check" value="4" /> Pastèque</label> <br /><br />
        <input type="button" value="Afficher la case cochée" onclick="check();" />
</fieldset>
<script>
        function check() {
                var inputs = document.getElementsByTagName('input'),
                inputsLength = inputs.length;
                for (var i = 0 ; i < inputsLength ; i++) {
                        if (inputs[i].type == 'radio' && inputs[i].checked) {
                                alert('La case cochée est du produit'+ inputs[i].value);
                        }}}
</script>
```


Les listes déroulantes avec selectedIndex et options

- Les listes déroulantes possèdent elles aussi leurs propres propriétés.
  Nous allons en retenir seulement deux parmi toutes celles qui existent :
  selectedIndex, qui nous donne l'index (l'identifiant) de la valeur sélectionnée,
  options qui liste dans un tableau les éléments **<option>** de notre liste déroulante.

```
<select id="list">
        <option>Sélectionnez votre sexe</option>
        <option>Homme</option>
        <option>Femme</option>
</select>
<script>
   var maliste = document.getElementById('list');
   maliste.addEventListener('change', function() {
   // On affiche le contenu de l'élément <option> ciblé par la propriété selectedIndex
   alert(maliste.options[maliste.selectedIndex].innerHTML);});
</script>
```

submit et reset

```
<form id="myForm">
        <input type="text" value="Entrez un texte" /> <br /><br />
        <input type="submit" value="Submit !" />
        <input type="reset" value="Reset !" />
</form>
<script>
        var myForm = document.getElementById('myForm');
        myForm.addEventListener('submit', function(e) {
                alert('Vous avez envoyé le formulaire !\n\nMais celui-ci a été bloqué pour
                        que vous ne changiez pas de page.');
                e.preventDefault(); } ) ;
        myForm.addEventListener('reset', function(e) {
                alert('Vous avez réinitialisé le formulaire !'); } );
</script>
```


- Recherche :
  - document.getElementById()
  - document.getElementsByTagName()
  - document.getElementsByClassName()
- Modification éléments :
  - element.innerHTML=
  - element.setAttribute(attribute,value)
  - element.style.property=
- Modification document :
  - document.createElement()
  - document.removeChild(child)
  - element.appendChild(child)
  - element.replaceChild(newchild, oldchild)
- Evenements :

  element.on****=function(){…}