Technologies du Web

Chapitre 3 :
Langage JavaScript

Caractéristiques de JavaScript

- Langage de scripts : petits programmes dédiés à une interaction avec l'utilisateur.
- JavaScript ≠ Java (aucune relation entre les deux langages).
- Langage interprété et faiblement typé.

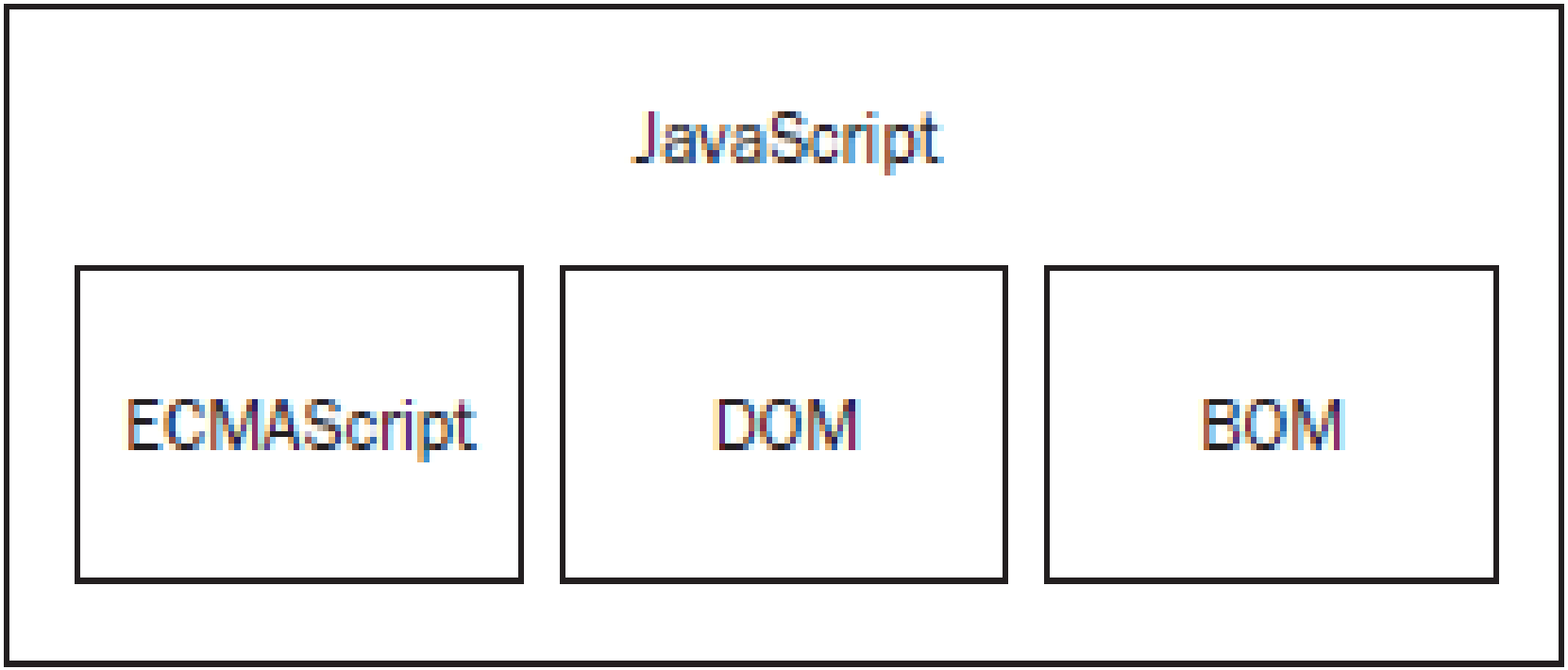
Caractéristiques de JavaScript

- Permet de développer des sites Web dynamiques par le biais de scripts.
- Langage de programmation assez riche : structurelle, orientée objet, événementielle, interactive.

Caractéristiques de JavaScript

- Trois facettes :
  - ✓ Core JavaScript : noyau du langage (ECMAScript).
  - ✓ Client-side : JavaScript côté client.
  - ✓ Server-side : JavaScript côté serveur.

Implémentations de JavaScript



Implémentations de JavaScript

- **ECMAScript** :
  - ✓ Noyau du langage (syntaxe, types, mots-clés, instructions, opérateurs et objets).
  - ✓ Plusieurs versions.
  - ✓ Supporté par la plupart des navigateurs.

Implémentations de JavaScript

- **DOM** : Document Objet Model.
  - ✓ Un API (*Application Programming Interface*) pour XML.
  - ✓ Étendu pour HTML.
  - ✓ Traduit un document HTML plat en un arbre (*DOM Tree*).

Implémentations de JavaScript

- **BOM** : Browser Objet Model.
  - ✓ Interface qui permet d'accéder et de manipuler la fenêtre du navigateur.
  - ✓ Interface ayant une structure d'arbre.

JavaScript dans HTML

- **<script>** : permet d'insérer un script dans un document HTML.
- Attributs :
  - ✓ **language** : langage dans lequel est codé le script (VBScript, JavaScript).
  - ✓ **type** : type MIME du script.

JavaScript dans HTML

- ✓ **src** : indique l'adresse URL du code à exécuter (script externe).
- ✓ **type** : type MIME du script.

- **<noscript>** : permet de cacher un script lorsque le navigateur ne prend pas en charge les scripts.

JavaScript dans HTML

- Méthodes pour insérer un script dans un document HTML :
  - ✓ En ligne.
  - ✓ Dans un fichier de script externe portant l'extension « **.js** ».
  - ✓ dans le navigateur via le protocole « javascript: ».

JavaScript dans HTML

- Placement du code JavaScript :
  - ✓ Dans la section <head> : déclarations globales, liaison des scripts externes.
  - ✓ Dans la section <body> : script en ligne, appel d'une fonction déclarée dans <head>.

JavaScript dans HTML

- Exemple d'un script en ligne :

```
<script language = "JavaScript"
   type = "text/javascript">
alert("Bonjour");
</script>
```

JavaScript dans HTML

- Exemple d'un script externe :

```
<script language = "JavaScript1.0"
    type = "text/javascript"
    src = "mon_script.js">
</script>
```

JavaScript dans HTML

- Exemple d'un script via le protocole « javascript: » :

  javascript:alert("Bonjour");

Syntaxe de JavaScript

- Sensibilité à la casse.
- **Identificateurs** : comme C avec la possibilité de commencer avec le symbole **$**.
- **Commentaires** : au style du C (**//** ou bien **/* . . . */**).

Syntaxe de JavaScript

- **Instructions** :
  - ✓ Le point virgule « ; » facultatif à la fin de ligne (mais obligatoire à l'intérieur).
  - ✓ Les accolades « { » et « { » pour écrire des blocs d'instructions.

Syntaxe de JavaScript

- **Liste des mots-clés** :

```
break case catch continue
   default delete do else
 finally for function if in
   instanceof new return
   switch this throw try
 typeof var void while with
```

Syntaxe de JavaScript

- **Déclaration des variables** :
  - ✓ Avec utilisation de « **var** » :

    var identificateur [= valeur] [;]
  - ✓ Sans l'utilisation de « **var** ».

    identificateur [= valeur] [;]
  - ✓ Déclaration multiple avec la virgule « **,** ».

Types de données JavaScript

- Types prédéfinis :
  - ✓ Boolean
  - ✓ Number
  - ✓ String
  - ✓ Null
  - ✓ Undefined
  - ✓ Object

Types de données JavaScript

- Opérateur **typeof** : renvoie un String contenant le type d'une variable ou d'une valeur.
- Exemples :
  - ✓ **typeof** 5 --> 'number'
  - ✓ **typeof** true --> 'boolean'
  - ✓ **typeof** "5" --> 'string'

Types de données JavaScript

- Le type **Undefined** :
  - ✓ Possède une seule valeur qui est **undifend**.
  - ✓ Une variable déclarée, mais non initialisée, prend par défaut la valeur undefined : elle est donc de type Undefined.

Types de données JavaScript

- Exemples :
  - ✓ **var** n;
  - ✓ **typeof** n --> 'undefined'
  - ✓ n = 17;
  - ✓ **typeof** n --> 'number'
  - ✓ **typeof** x // erreur

Types de données JavaScript

- Le type **Null** :
  - ✓ Une seule valeur : **null**.
  - ✓ Si une variable est une référence sur un objet non défini (vide), sa valeur est null et elle est de type Object.
  - ✓ Les valeurs null et undefined sont égales.

Types de données JavaScript

- Le type **Boolean** :
  - ✓ Deux valeurs possibles : **true** (vrai) et **false** (faux).
- Exemples :
  - ✓ 5 == 5 --> true
  - ✓ 4 == 5 --> false
  - ✓ **typeof** false --> 'boolean'

Types de données JavaScript

- Valeurs convertibles en **true** :
  - ✓ **true**.
  - ✓ Toute chaîne non vide.
  - ✓ Toute valeur numérique non nulle (même infinie).
  - ✓ Tout objet.

Types de données JavaScript

- Valeurs convertibles en **false** :
  - ✓ **false**.
  - ✓ La chaîne vide : **""**.
  - ✓ **0** et **NaN**.
  - ✓ **null**.
  - ✓ **undefined**.

Types de données JavaScript

- Fonction de conversion vers une valeur booléenne : **Boolean**()
  - ✓ **Boolean**(0 == 0) --> true
  - ✓ **Boolean**(0) --> false
  - ✓ **Boolean**("0") --> true
  - ✓ **Boolean**("") --> false

Types de données JavaScript

- Le type **Number** :
  - ✓ Codage virgule flottante IEEE 754.
  - ✓ Entiers et réels à la fois.
  - ✓ Valeurs spéciales : **NaN** (Not a Number), **Infinity** (+∞), **-Infinity** (-∞).

Types de données JavaScript

- Le type **Number** :
  - ✓ Plus petite valeur positive : **Number.MIN_VALUE** (5e-324).
  - ✓ Plus grande valeur positive : **Number.MAX_VALUE** (1.7976931348623157e+308)

Types de données JavaScript

- Le type **Number** (fonctions) :
  - ✓ **isFinite**(valeur) : retourne **true** si la valeur est comprise entre la borne inf et la borne sup.
  - ✓ **isNaN**(valeur) : retourne **true** si la valeur est non convertible à un nombre.

Types de données JavaScript

- Le type **Number** (fonctions) :
  - ✓ **parseInt**(valeur, base) : convertie une valeur en un entier dans la base indiquée.
  - ✓ **parseFloat**(valeur) : convertie une valeur en un réel.
  - ✓ **Number**(valeur) : convertie une valeur en un numérique.

Types de données JavaScript

- Littéraux numériques :
  - ✓ **var** a = 70 --> décimal, 70
  - ✓ **var** a = 070 --> octal, 56
  - ✓ **var** a = 0x1f --> hexadécimal, 31
  - ✓ **var** a = 7.0 --> décimal, 7
  - ✓ **var** a = 1.2 --> réel, 1.2
  - ✓ **var** a = 1.378e2 --> réel, 137.8

Types de données JavaScript

- Exemples **isNaN** :
  - ✓ **isNaN**(NaN) --> true
  - ✓ **isNaN**(5) --> false
  - ✓ **isNaN**("5") --> false
    // car convertible en 5
  - ✓ **isNaN**(true) --> false
    // car convertible en 1

Types de données JavaScript

- Exemples **isFinite** et **Number** :
  - ✓ **isFinite**(1/0) --> flase
  - ✓ **isFinite**(125) --> true
  - ✓ **isFinite**(-1/0) --> false
  - ✓ **Number**(true) --> 1
  - ✓ **Number**("12") --> 12
  - ✓ **Number**("zero") --> NaN

Types de données JavaScript

- Exemples **parseInt** :
  - ✓ **parseInt**("1234ab") --> 1234
  - ✓ **parseInt**("") --> NaN
  - ✓ **parseInt**("021") --> 17
  - ✓ **parseInt**(12.34) --> 12
  - ✓ **parseInt**("0xab") --> NaN
  - ✓ **parseInt**("0xaf", 16) --> 175

Types de données JavaScript

- Exemples **parseFloat** :
  - ✓ **parseFloat**("1234ab") --> 1234
  - ✓ **parseFloat**("0xA") --> 0
  - ✓ **parseFloat**(12.34.75) --> 12.34
  - ✓ **parseInt**(0958.85) --> 958.85

Types de données JavaScript

- Le type **String** :
  - ✓ Séquence de caractères unicode sur 16-bits.
  - ✓ Valeur écrite entre deux quotte « ' » ou deux guillemets « " ».

Types de données JavaScript

- Littéraux caractères :
  - ✓ \n (New line)
  - ✓ \t (Tabulation)
  - ✓ \b (Backspace)
  - ✓ \r (Carriage return)
  - ✓ \f (Form feed)
  - ✓ \\ (Backslash)

Types de données JavaScript

- Littéraux caractères :
  - ✓ \' (Single quote)
  - ✓ \" (Double quote)
  - ✓ \xnn caractère représenté par un code hexadécimal (n is 0-F)
  - ✓ \unnn caractère unicode représenté par un code hexadécimal (n is 0-F)

Types de données JavaScript

- Propriété (attribut) **length** : calcul le nombre de caractères dans une chaîne.
- Fonction **String**(valeur) : convertie une valeur donnée en une chaîne.
- Méthode **toString**(base) : convertie une valeur numérique en une chaîne.

Types de données JavaScript

- Exemples :
  - ✓ **var** a = "bonjour";
  - ✓ **typeof** a --> 'string'
  - ✓ a.**length** --> 7
  - ✓ a = 10;
  - ✓ **String**(a) --> "10"
  - ✓ a.**toString**(2) --> "1010"

Opérateurs de JavaScript

- Opérateurs arithmétiques :
  - ✓ **+** : addition.
  - ✓ **-** : opposé et soustraction.
  - ✓ ***** : multiplication.
  - ✓ **/** et **%** : division et reste.
  - ✓ **++** : incrémentation.
  - ✓ **--** : décrémentation.

Opérateurs de JavaScript

- Opérateurs bit à bit :
  - ✓ ~ : complément à 1.
  - ✓ & | ^ : et, ou, ou exclusif bit à bit.
  - ✓ << : décalage à gauche.
  - ✓ >> : décalage à droite.
  - ✓ >>> : décalage à droite non signé.

Opérateurs de JavaScript

- Opérateurs logiques (booléens) :
  - ✓ ! : non logique.
  - ✓ && : et logique.
  - ✓ || : ou logique.

Opérateurs de JavaScript

- Opérateurs relationnels :
  - ✓ < : inférieur à.
  - ✓ <= : inférieur ou égal à.
  - ✓ > : supérieur à.
  - ✓ >= : supérieur ou égal à.
  - ✓ == : égal à.
  - ✓ != : différent de.

Opérateurs de JavaScript

- Opérateurs relationnels :
  - ✓ === : identiquement égal à.
  - ✓ !== : identiquement différent de.

Opérateurs de JavaScript

- Opérateur conditionnel : **?:**
- Opérateurs d'affectation : simple (=) et composé (**+= -= *= /= %= <<= >>= >>>=** …)
- Opérateur de séquence : **,**

Instructions JavaScript

- Instructions conditionnelles :
  - ✓ **if … else**, **switch**
- Instructions de boucles :
  - **for**, **while**, **do … while**
- Instructions **break** et **continue**

Instructions JavaScript

- Instruction **for**… **in**

  for(**var** *property* **in** *object*)

  statement

- Exemple :
  - ✓ **for**(**var** p **in** window) **alert**(p)

Instructions JavaScript

- Instruction **with**

  with(*object*)

  statement

- Exemple sans **with** :

```
alert(Math.sqrt(3));
alert(Math.sin(1.57);
alert(Math.rand());
```

Instructions JavaScript

- Exemple avec **with** :

```
with(Math)
{
    alert(sqrt(3));
    alert(sin(1.57));
    alert(rand());
}
```

Fonctions

- Déclaration d'une fonction JavaScript :

```
function Name(argument1, …,
    argumentN)
{
    // code
}
```

Fonctions

- Exemple :

```
function saluer(nom)
{
    alert("Bonjour, " + nom);
}
// appel
    saluer("Amine");
```

Fonctions

- Une fonction JavaScript peut retourner une valeur :

```
function somme(nb1, nb2) {
    return nb1 + nb2;
}
// appel
    var a = somme(10, 20); --> 30
```

Fonctions

- Toute fonction JavaScript possède une propriété **arguments**.
- **arguments** est un tableau qui mémorise les valeurs des arguments d'une fonction.
- **arguments.length** : nombre des arguments de la fonction.

Fonctions

```
function add() {
    var n = arguments.length;
    if(n == 0) alert("rien");
    else if(n == 1)
        return arguments[0];
    else if(n == 2) return
    arguments[0] + arguments[1]; }
```

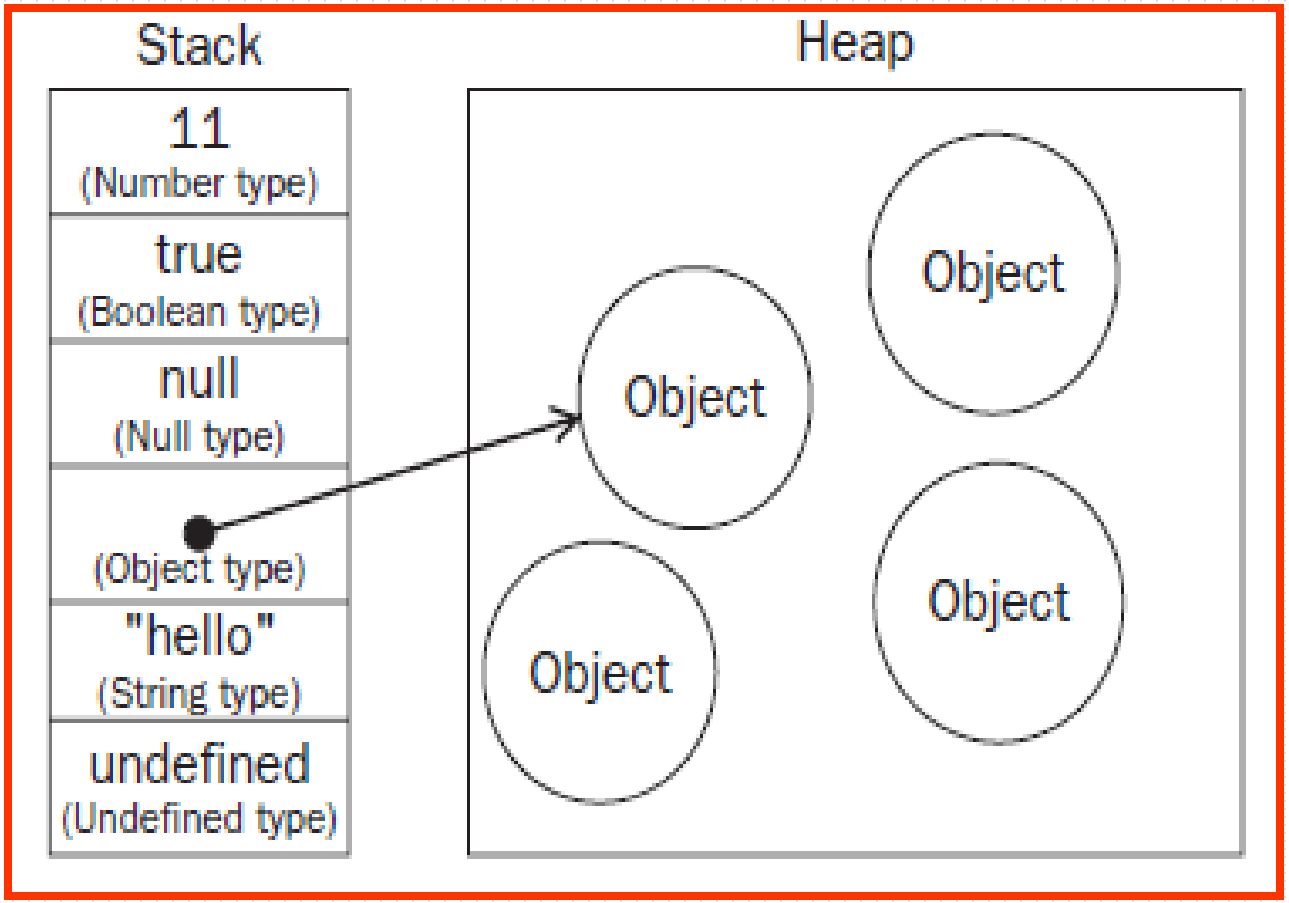
Fonctions

- Appel de la fonction *add* :
  - *add*(); // "rien";
  - *add*(3); // 3
  - *add*(2, 3); // 5
  - *add*(1, 2, 3); // *undefined*

Variables et mémoire

- En JavaScript, la valeur d'une variable peut être de deux catégories :
  - ✓ **Primitive**.
  - ✓ **Référence**.

Variables et mémoire



Variables et mémoire

- Valeurs primitives :
  - ✓ stockées dans la **pile** (**stack**).
  - ✓ accessibles séquentiellement.

Variables et mémoire

- Valeurs références :
  - ✓ stockées dans le **tas** : zone de la mémoire pour stocker les objets.
  - ✓ non accessibles d'une façon séquentielle.
  - ✓ agissent comme des pointeurs vers des objets.

Variables et mémoire

- Valeurs références :
  - ✓ sont crées en utilisant l'opérateur **new**.
  - ✓ Syntaxe :

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

  - ✓ Exemple :

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

Variables et mémoire

- Valeurs références :
  - ✓ peuvent avoir de propriétés (attributs) définies d'une manière dynamique.
  - ✓ Exemple :

```
var livre = new Object();
livre.auteur = "T. Corman";
```

Variables et mémoire

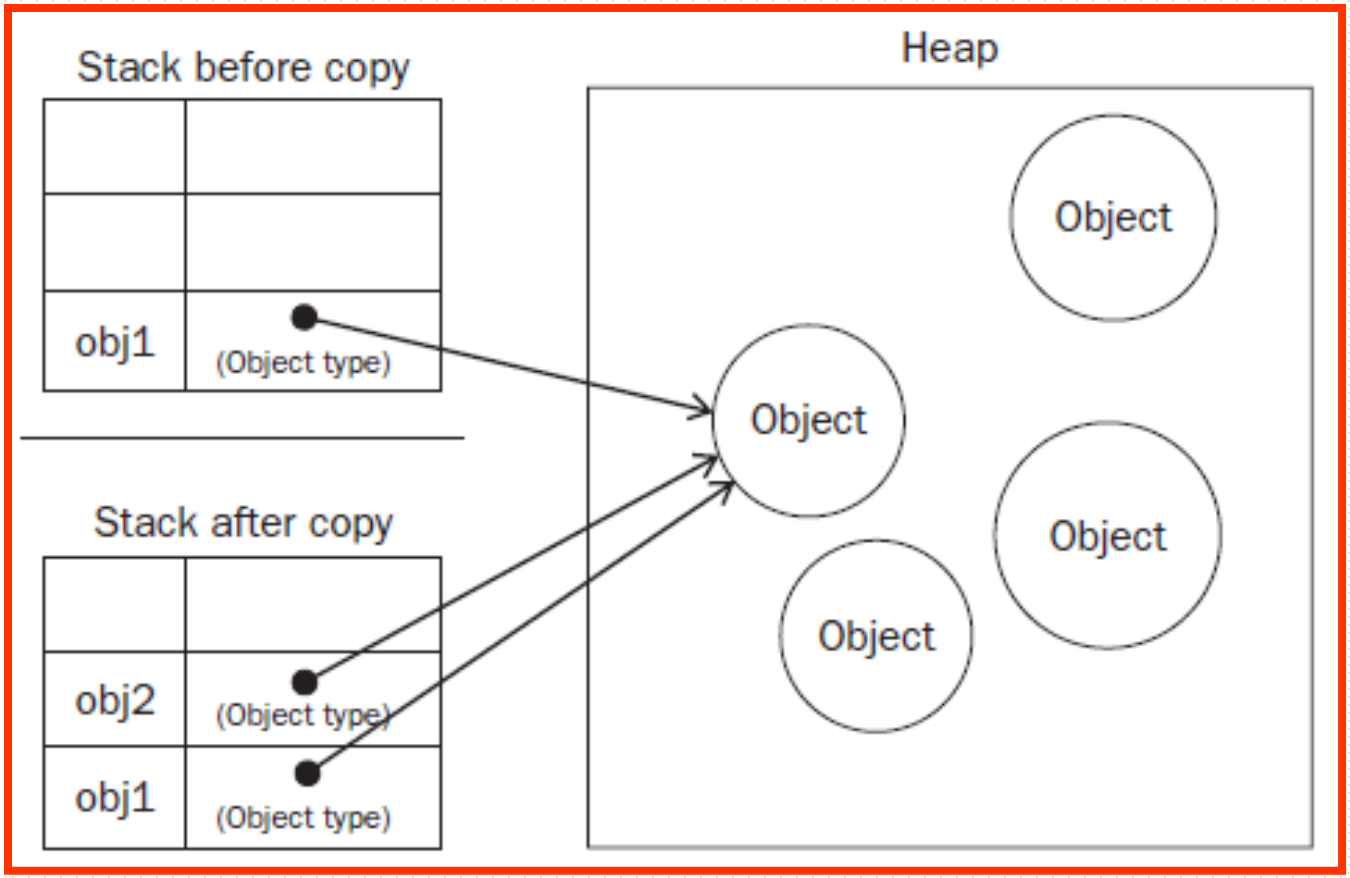
- ✓ Exemple (suite) :

```
livre.titre = "Introduction à
   l'algorithmique";
livre.prix = 450;
alert(livre.prix) --> 450
```

Variables et mémoire

- **Copie des valeurs** :
  - ✓ Cas de valeurs primitives : copie de valeur simple.
  - ✓ Cas de valeurs références : copie de référence. Les deux variables pointent sur le même objet (elles auront les mêmes propriétés).

Variables et mémoire



Variables et mémoire

- Exemple 1 :

```
var num1 = 17;
var num2 = num1; // copie
alert(num2); --> 17
```

Variables et mémoire

- Exemple 2 :

```
var obj1 = new Object();
var obj2 = obj1; // copie
obj1.nom = "ajax";
alert(obj2.nom) --> "ajax"
```

Variables et mémoire

- **Passage des arguments** :
  - ✓ Passage par valeur.
  - ✓ Les valeurs des arguments seront copiées dans des variables locales.
  - ✓ La valeur d'un argument n'est pas modifiée après l'appel de la fonction.

Variables et mémoire

- Exemple 1 :

```
function add10(num) {
    num += 10;  return num; }
var n = 10;
var r = add10(n);
alert(r); --> 20
alert(n); --> 10
```

Variables et mémoire

- Exemple 2 :

```
function setNom(obj) {
    obj.nom = 'Mars';
    obj = new Object();
    obj.nom = 'Neptun'; }
var p = new Object();
setNom(p);  alert(p.nom); --> 'Mars'
```

Variables et mémoire

- Portée des variables :
  - ✓ Variables **globales** : déclarées avant un bloc ou une fonction et hors celle-ci.
  - ✓ Variables **locales** : déclarées au premier niveau à l'intérieur d'un bloc ou une fonction.

Variables et mémoire

- Exemple 1 :

```
var color = 'red';    // globale
function changeColor(newcolor)
{  color = newcolor;  }
changeColor('yellow'); // appel
alert(color); --> 'yellow'
```

Variables et mémoire

- Exemple 2 :

```
function add(x, y)
{  var r = x + y;
   return r; // locale  }
var z = add(2, 3);
alert(z);  // 5
alert(r);  // erreur
```

Types références prédéfinis

- Les types références prédéfinis de JavaScript :
  - ✓ **Object**.
  - ✓ **Array**.
  - ✓ **Date**.
  - ✓ **RegExp**.
  - ✓ **Function**.

Type Object

- Déclaration d'une variable de type Object :

```
var obj1 = new Object();
var obj2 = { };
var obj3 = {
        nom = 'Saad';
        age = 25 };
```

Type Object

- Accès à une propriété d'une variable de type Object :

```
var obj = {
        nom = 'Saad';
        age = 25  };
alert(obj.nom); --> 'Saad'
alert(obj['age']); --> 25
```

Type Array

- Propriétés :
  - ✓ **Array** = tableau (associatif).
  - ✓ Premier élément d'indice 0.
  - ✓ Les éléments peuvent être de types différents.
  - ✓ La taille du tableau est dynamique (non statique).

Type Array

- Déclaration :

```
// tableau vide
var colors = new Array();
// tableau à 5 éléments non initié
var colors = new Array(5);
// tableau à 3 éléments initialisé
var numbres = new Array(1, 5, 8);
```

Type Array

- Déclaration :

```
var tab1 = Array(1, true, 'zigzag');
var tab2 = [5, 'Casa', 22000];
var tab3 = []; // vide
```

Type Array

- Accès aux éléments d'un tableau :

```
var p = ["said", 25, "oujda", true];
alert(p[0]); --> 'said'
p[1] = 35;
alert(p[1]); --> 35
if(p[3]) alert('marié'); --> 'marié'
alert(p[4]); --> undefined
```

Méthodes de l'objet Array

- Méthodes de conversion :
  - ✓ **toString**() : convertie le contenu d'un tableau en une chaîne. Les valeurs des éléments sont séparées par des virgules.
  - ✓ **valueOf**() : identique à toString() lorsqu'elles sont appliquées à un tableau.

Méthodes de l'objet Array

- Exemples :

```
var colors = ['red', 'blue', 'green'];
alert(colors.toString());
// affiche 'red,blue,green'
var numbers = [4, 5, 8];
alert(numbers.valueOf());
// affiche '4,5,8'
```

Méthodes de l'objet Array

- Méthodes de conversion :
  - ✓ **join**(str) : identique à **toString**(), sauf que les valeurs des éléments sont séparées par la chaîne indiquée par l'argument str.
- ➢ Ces trois méthodes ne sont pas à effet de bord.

Méthodes de l'objet Array

- Exemple :

```
var colors = ['red', 'blue', 'green'];
alert(colors.join('#'));
// affiche 'red#blue#green'
alert(colors.join('||'));
// affiche 'red||blue||green'
```

Méthodes de l'objet Array

- Méthodes de pile (LIFO) :
  - ✓ **push**(liste) : emplie de nouveaux éléments à la fin d'un tableau.
  - ✓ **pop**() : dépile le dernier élément d'un tableau.
- ➢ Ces deux méthodes sont à effet de bord.

Méthodes de l'objet Array

- Exemple :

```
var primes = [2, 3, 5];
primes.push(7, 11);
primes.toString(); --> '2,3,5,7,11'
var elem = primes.pop();
primes.toString(); --> '2,3,5,7'
alert(elem); --> 11
```

Méthodes de l'objet Array

- Méthodes de file (FIFO) :
  - ✓ **unshift**(liste) : insère de nouveaux éléments au début d'un tableau.
  - ✓ **shift**() : supprime le premier élément d'un tableau.
- ➢ Ces deux méthodes sont à effet de bord.

Méthodes de l'objet Array

- Exemple :

```
var primes = [5, 7, 11];
primes.unshift(2, 3);
primes.toString(); --> '2,3,5,7,11'
var elem = primes.shift();
primes.toString(); --> '3,5,7,11'
alert(elem); --> 2
```

Méthodes de l'objet Array

- Méthodes de ré-ordonnancement :
  - ✓ **reverse**() : inverse l'ordre des éléments d'un tableau.
  - ✓ **sort**() : trie un tableau dans l'ordre croissant.
- ➢ Ces deux méthodes sont à effet de bord.

Type Array

- Exemple :

```
var primes = [11, 7, 3, 2, 5];
primes.reverse();
primes.toString(); --> '5,2,3,7,11'
primes.sort();
primes.toString(); --> '2,3,5,7,11'
```

Méthodes de l'objet Array

- Méthodes de manipulation :
  - ✓ **concat**(liste) : concatène une liste à la fin d'un tableau.
  - ✓ **slice**(deb, fin) : extrait le sous-tableau d'un tableau à partir de la position indiquée par « deb » jusqu'à la position « fin ».

Type Array

- Exemple :

```
var primes = [2, 3, 5];
var q = primes.concat(7, 11);
alert(q); --> '2,3,5,7,11'
alert(primes); --> '2,3,5'
// concat est sans effet de bord
```

Type Array

- Exemple :

```
var primes = [2, 3, 5, 7, 11];
var q = primes.slice(1, 3);
alert(q); --> '3,5,7'
alert(primes); --> '2,3,5,7,11'
// slice est sans effet de bord
```

Méthodes de l'objet Array

- La méthode **splice** :
  - ✓ **splice**(deb, n) : supprime « n » éléments à partir de la position indiquée par l'indice « deb ».
  - ✓ **splice**(deb, 0, liste) : insère une liste à partir de la position indiquée par l'indice « deb ».

Méthodes de l'objet Array

- La méthode **splice** :
  - ✓ **splice**(deb, n, liste) : supprime « n » éléments à partir de la position indiquée par l'indice « deb » et insère les éléments de la liste donnée à partir de la position « deb » (un effet de remplacement).

Méthodes de l'objet Array

- Exemple :

```
var colors = ['red', 'cyan', 'green'];
colors.splice(1, 1);
alert(colors); --> 'red,green'
colors.splice(1, 0, 'yellow');
alert(colors); --> 'red,yellow,green'
```

Méthodes de l'objet Array

- Exemple (suite) :

```
colors.splice(1, 1, 'blue');
alert(colors); --> 'red, blue,green'
```

Type Date

- Le type **Date** :
  - ✓ Permet de représenter des valeurs pour les dates.
  - ✓ Regroupe l'information sur la date et l'heure.
  - ✓ Dispose de plusieurs méthodes pour la gestion des dates et des heures.

Type Date

- Création d'un objet de type **Date** :
  - ✓ `var d = new Date();`
  - ✓ La variable « d » est un objet de type « Date ».
  - ✓ Son contenu est la date et l'heure courantes (système).

Type Date

- Création d'un objet de type **Date** :
  - ✓ **var** d = **new Date**(2010, 10, 7);
  - ✓ Le contenu de « d » est la date du "7 Novembre 2010".
  - ✓ Le **constructeur** « Date » possède plusieurs syntaxes pour créer un objet de type « Date ».

Quelques méthodes de Date

- Méthodes « getters » :
  - ✓ **valueOf**() et **getTime**() : renvoient le temps en millisecondes.
  - ✓ **getFullYear**() : renvoie l'année sur 4 chiffres.
  - ✓ **getMonth**() : retourne le mois sous forme d'un nombre (0…11).

Quelques méthodes de Date

- Méthodes « getters » :
  - ✓ **getDate**() : retourne le jour sous forme d'un nombre (1…31).
  - ✓ **getDay**() : retourne le nom du jour sous forme d'un nombre (0…6).

Quelques méthodes de Date

- Méthodes « getters » :
  - ✓ **getHours**() : retourne l'heure sous forme d'un nombre (0…23).
  - ✓ **getMinutes**() : retourne le nombre de minutes (0…59).
  - ✓ **getSeconds**() : retourne le nombre de secondes (0…59).

Quelques méthodes de Date

- Méthodes « setters » :
  - ✓ **setTime**(ms).
  - ✓ **setYearFull**(annee).
  - ✓ **setMonth**(mois).
  - ✓ **setDate**(date).
  - ✓ **setDay**(jour).
  - ✓ **setHours**(heure).

Quelques méthodes de Date

- Méthodes « setters » :
  - ✓ **setMinutes**(minutes).
  - ✓ **setSeconds**(secondes).

Type RegExp

- Le type **RegExp** permet de créer des **expressions régulières** comme celle du langage **Perl**.
- Syntaxe Perl :
  - ✓ **var** er = /*pattern*/*flags*;
- *pattern* (motif) : expression régulière simple ou complexe.

Type RegExp

- *flags* : précisions sur l'expression régulière.
- Trois valeurs de flags :
  - ✓ **g** (mode global) : le motif sera appliqué sur toute la chaîne au lieu de s'arrêter après la 1ère correspondance.

Type RegExp

- ✓ **i** (mode sensibilité à la casse) : indique d'ignorer la casse dans le motif lors de la recherche d'une correspondance.
- ✓ **m** (mode multi-lignes) : indique que la recherche du motif sera continuée après atteindre un saut de ligne.

Type RegExp

- Exemples :

```
var er1 = /at/g;
// toutes les instances de "at"
var er = /[bc]at/i;
// 1ère instance de "bat" ou "cat"
```

Type RegExp

- **Méta-caractères** :
  - ✓ [ ] : intervalle.
  - ✓ ( ) : regroupement.
  - ✓ { } : définition.
  - ✓ . : n'importe quel caractère.
  - ✓ $: se termine par.
  - ✓ ^ : commence par.

Type RegExp

- **Méta-caractères** :
  - ✓ * : 0, 1 ou plusieurs occurrences.
  - ✓ + : , 1 ou plusieurs occurrences.
  - ✓ ? : 0 ou 1 occurrence.
  - ✓ | : réunion.
  - ✓ \ : préfixe pour un méta-caractère à reconnaître.

Type RegExp

- **Constructeur RegExp :**
  - ✓ **var** er = **new RegExp**(*pattern*, *flags*).
- Exemple :

```
var er1 = new RegExp("[at]+", "g");
```

Type RegExp

- Propriétés d'un objet RegExp :
  - ✓ **global** : booléen indiquant si le flag 'g' est considéré.
  - ✓ **igonreCase** : booléen indiquant si le flag 'i' est considéré.
  - ✓ **multiline** : booléen indiquant si le flag 'm' est considéré.

Type RegExp

- Propriétés d'un objet RegExp :
  - ✓ **source** : chaîne du motif de l'expression régulière.
  - ✓ **lastIndex** : Entier indiquant la position où commencer la recherche, 0 par défaut.
- Les 3 premières : *false* par défaut.

Type RegExp

- Exemples :

```
var er = /\[bc\]at/i;
alert(er.global);          // false
alert(er.ignoreCase);      // true
alert(er.multiline);       // false
alert(er.lastIndex);       // 0
alert(er.source);          // "\[bc\]at"
```

Type RegExp

- Méthodes d'un objet RegExp :
  - ✓ **test(**texte**)** : renvoie *true*, si le texte contient un mot accepté par l'expression régulière.
  - ✓ **exec(**texte**)** : renvoie un tableau contentant les mots reconnus par l'expression régulière.

Type RegExp

- Exemples :

```
var r = /j.*t/, texte = "JavaScript";
alert(r.test(texte)); // false
r = /j.*t/i;
alert(r.test(texte)); // true
var t = r.exec(texte));
alert(t[0]); // "JavaScript"
```

L'objet Math

- Il s'agit d'un objet (et non pas d'un type d'objet).
- Il regroupe des propriétés et des méthodes pour faire des calculs mathématiques usuels.
- L'objet Math est **extensible** : on peut l'enrichir par des fonctions.

Propriétés de l'objet Math

- **E** : valeur de e.
- **LN10** : valeur $ln(10)$.
- **LN2** : valeur $ln(2)$.
- **LOG2E** : valeur $ln_2(e)$.
- **LOG10E** : valeur $ln_{10}(e)$.
- **PI** : valeur de π.
- **SQRT2** et **SQRT1_2** : $\sqrt{2}$ et $\sqrt{(1/2)}$.

Méthodes de l'objet Math

- **max** : maximum d'un tableau.
- **min** : minimum d'un tableau.
- **floor** : partie entière inférieure.
- **ceil** : partie entière supérieure.
- **round** : arrondi à l'entier le plus proche.
- **rand** : réel aléatoire entre 0 et 1.

Méthodes de l'objet Math

- **sin**, **cos**, **tan** : sinus, cosinus et tangente.
- **asin**, **acos**, **atan** : arc sinus, arc cosinus et arc tangente.
- **atan2**(y, x) : arc sinus de l'angle y/x.
- **sqrt** : racine carrée.

Méthodes de l'objet Math

- **abs** : valeur absolue.
- **exp** : exponentielle.
- **log** : logarithme népérien (base *e*).
- **pow**(x, n) : x puissance n (x^n).

Types primitifs comme objets

- Constructeurs :
  - ✓ **Number**();
  - ✓ **Boolean**();
  - ✓ **String**();

Types primitifs comme objets

- Méthodes des objets **Number** :
  - ✓ **toFixed**(n) : spécifie le nombre de chiffres après la virgule.
  - ✓ **toExponential**(exp) : convertie en la notation exponentielle.
  - ✓ **toPrecision**(n) : combine les deux.

Types primitifs comme objets

- Méthodes des objets **String** :
  - ✓ **charAt**(i) : caractère d'indice i.
  - ✓ **charCodeAt**(i) : code du caractère d'indice i.
  - ✓ **Concat**(…) : concaténation.
  - ✓ **slice**, **substr** et **substring** : extraction de sous-chaînes.

Types primitifs comme objets

- Méthodes des objets **String** :
  - ✓ **indexOf**(ch, pos) : indice de la 1ère occurrence de ch après la position pos.
  - ✓ **lastIndexOf**(ch, pos) : indice de la dernière occurrence de ch après la position pos.

Types primitifs comme objets

- Méthodes des objets **String** :
  - ✓ **toUpperCase**() : mettre les caractères au majuscule.
  - ✓ **toLowerCase**() : mettre les caractères au minuscule.
  - ✓ **localCompare**(ch) : compare une chaîne à une autre (1, 0, -1).

Types primitifs comme objets

- Méthodes des objets **String** :
  - ✓ **fromCharCode**(C1, …, Cn) : forme une chaîne à partir des codes ASCII donnés.

Types primitifs comme objets

- Méthodes HTML des objets **String** :
  - ✓ **anchor**(name) : <**a name** = "name">string</**a**>
  - ✓ **big**() : <**big**>string</**big**>
  - ✓ **bold**() : <**b**>string</**b**>
  - ✓ **fixed**() : <**tt**>string</**tt**>

Types primitifs comme objets

- Méthodes HTML des objets **String** :
  - ✓ **fontcolor**(color) : <**font color** = "color">string</**font**>
  - ✓ **fontsize**(size) : <**font size** = "size">string</**font**>
  - ✓ **italics**() : <**i**>string</**i**>

Types primitifs comme objets

- Méthodes HTML des objets **String** :
  - ✓ **link**(url) : <**a href** = "url">string</**a**>
  - ✓ **small**() : <**small**>string</**small**>
  - ✓ **strike**() : <**strike**>string</**strike**>
  - ✓ **sub**() : <**sub**>string</**sub**>
  - ✓ **sup**() : <**sup**>string</**sup**>

POO en JavaScript

- Créer des objets personnalisés :
  - ✓ Créer une nouvelle instance de **Object**.
  - ✓ Ajouter des propriétés et des méthodes à cette instance.

POO en JavaScript

- Exemple :

```
var person = new Object();
person.name = "Alice";
person.age = 45;
person.job = "Webmaster";
person.getName() = function() {
    return this.name; }
```

POO en JavaScript

- Inconvénient :
  - ✓ La création de plusieurs objets de même interface nécessite une très grande duplication de code.
- Solution :
  - ✓ Utiliser une approche d'usine (un fabriqua).

POO en JavaScript

- Exemple :

```
function createPerson(name, age,
    job) {
    var pers = new Object();
    pers.name = name;
    pers.age = age;
    pers.job = job;
```

POO en JavaScript

- Exemple (suite) :

```
pers.getName() = function()
{
        return this.name;
};
return pers;
}
```

POO en JavaScript

- Exemple (suite) :

```
var ali = createPerson("Ali", 39, "Doctor");
var amine = createPerson("Amine", 29, "C++ Devlopper");
var yahia = createPerson("Yahia", 35, "Teacher");
```

POO en JavaScript

- Avantage : la création de plusieurs objets de même interface est facilité.
- Inconvénient : impossibilité d'identifier le type des objets crées.
- Solution : utiliser un constructeur.

POO en JavaScript

- Exemple :

```
function Person(name, age, job)
{
    this.name = name;
    this.age = age;
    this.job = job;
```

POO en JavaScript

- Exemple (suite) :

```
this.getName() = function()
{
    return this.name;
};
}
```

POO en JavaScript

- Exemple (suite) :

```
var ali = new Person("Ali", 39,
"Doctor");
var amine = new Person("Amine",
29, "C++ Devlopper");
var yahia = new Person("Yahia",
35, "Teacher");
```

POO en JavaScript

- Comparaison des approches (usine/constructeur) :
  - ✓ Pas d'objet crée d'une manière explicite.
  - ✓ Les propriétés et méthodes sont assignées directement à **this**.
  - ✓ Il n'y a pas d'instruction **return**.

POO en JavaScript

- étapes d'appel d'un constructeur :
  - ✓ Création d'un nouveau objet.
  - ✓ Faire pointer le pointeur « **this** » vers ce nouveau objet.
  - ✓ Exécution du code de constructeur : ajout des propriétés et méthodes.

POO en JavaScript

- étapes d'appel d'un constructeur :
  - ✓ Retourner l'objet crée.
- propriété **constructor** :
  - ✓ obj.**constructor** : renvoie le constructeur qui a servi pour créer l'objet « obj ».

POO en JavaScript

- Opérateur **instanceOf** :
  - ✓ obj **instanceOf** class : renvoie *true* si l'objet « obj » est une instance de la classe « class » et *false* sinon.
- Exemple :

```
ali instanceOf Person; // true
```

POO en JavaScript

- **Prototypage** :
  - ✓ Chaque fonction possède une propriété « **prototype** ».
  - ✓ C'est un objet contenant des propriétés et méthodes qui seront disponibles pour chaque instance d'une classe spécifique.

POO en JavaScript

- Exemple :

```
function Person() { }
// prototypage
Person.prototype.name = "Paul";
Person.prototype.age = 55;
Person.prototype.job = "Director";
```

POO en JavaScript

- Exemple (suite) :

```
Person.prototype.getName =
    function()
{ return this.name; }
// Appel
var pers = new Person();
alert(pers.getName()); // "paul"
```

POO en JavaScript

- Exemple (suite) :

```
// mise à jour
pers.name = "Jack";
alert(pers.getName()); // "Jack";
pers.age = 36;
pers.jor = "Officier";
```

BOM : Browser Object Model

- Le navigateur mémorise son interprétation du code HTML sous la forme d'une structure d'objets Javascript, appelée **Modèle Objet Document** ou **DOM**.
- Dans le DOM, chaque élément HTML devient un objet.

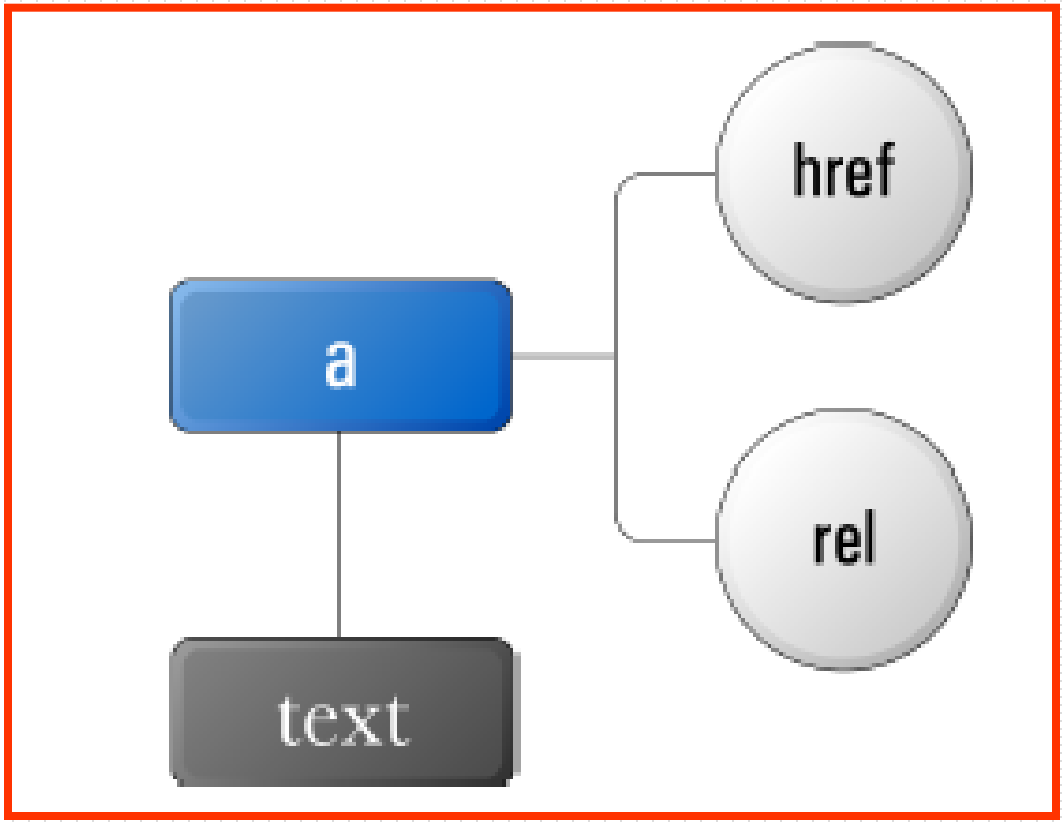
BOM : Browser Object Model

- Le **DOM** a une structure d'arbre.
- Le nœud racine de cet arbre est l'élément **document**.
- Le premier fils du nœud racine est l'élément **html**.
- Ce dernier possède deux fils : **head** et **body**.

BOM : Browser Object Model

- ✓ Dans le **DOM** :
  - ✓ un contenu textuel est un **nœud de texte**.
  - ✓ Un attribut est un **nœud d'attribut**.

BOM : Browser Object Model



BOM : Méthodes d'accès

- Trouver un élément par son **id** :
  - **document.getElementById**("id").
  - La valeur **null** est retournée si l'élément n'est pas trouvé.
  - La propriété **nodeName** renvoie le nom propre de l'élément.

BOM : Méthodes d'accès

✓ Exemple :

```
var elem =
    document.getElementById("p1");
if(elem != null)
{
    alert(elem.nodeName);
}
```

BOM : Méthodes d'accès

- Trouver un élément par son **nom** :
  - **document.getElementsByTagName**("name").
  - Le résultat retourné est la liste de tous les éléments de nom donné.

DOM : Document Object Model

- Exemple (suite) :

```
// mise à jour
pers.name = "Jack";
alert(pers.getName()); // "Jack";
pers.age = 36;
pers.jor = "Officier";
```

Les événements

- Exemple (suite) :

```
// mise à jour
pers.name = "Jack";
alert(pers.getName()); // "Jack";
pers.age = 36;
pers.jor = "Officier";
```

Scripts pour formulaires

- Exemple (suite) :

```
// mise à jour
pers.name = "Jack";
alert(pers.getName()); // "Jack";
pers.age = 36;
pers.jor = "Officier";
```