Le langage Java Script

Les bases

SOMMAIRE

Avant-propos

- Côté client, sur un navigateur, il permet de dynamiser le contenu.

- JavaScript est :
  - Un langage de script multiplateforme
  - Orienté objet
  - Non typé
  - Basé sur le prototypage et non sur les classes
  - Sensible à la casse
  - Interprété ou compilé à la volée

- ES6 introduit le mot-clé classe mais ce n'est qu'une syntaxe pour faciliter l'utilisation du prototypage.

- Le langage est en pleine évolution, depuis deux ans on peut dire révolution

Avant-propos

- Le JavaScript est envoyé avec la page HTML au client (le navigateur web).

- Le JavaScript sera interprété par le navigateur (tout comme l'HTML et le CSS).

Avant-propos : Ajouter un script

- Deux façons d'ajouter un script dans une page HTML :
  - Directement entre les balises script :

```
<script>
    console.log("code JavaScript");
</script>
```

  - Depuis un fichier .js chargé dans une page HTML :

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

- Le second choix est conseillé pour respecter une séparation entre les langages et ainsi avoir une meilleure lisibilité.

- Il est souvent conseillé de placer certains scripts en pied de page :
  - Le code est chargé de façon synchrone, ce qui veut dire que le navigateur bloque le chargement de la page pendant qu'il charge le script.

Avant-propos : Ajouter un script

- Les différents scripts, qu'ils soient dans des fichiers ou non, se partagent un espace globale.
- Ils peuvent communiquer entre eux et se partager des variables ou des fonctions.
- Il faut donc garder une bonne vision globale de son site web et de l'ensemble des script qui le compose.
- Le développement Java Script n'est pas simple, il est simplifié par un très grand nombre de framework

Avant-propos : Mode strict

- Variante de JavaScript avec des restrictions

- Transforme des erreurs silencieuses en erreurs explicites
  - Votre code devra être parfait sinon il ne fonctionnera pas du tout
  - Code plus rapide
  - Empêche l'utilisation de mots réservés (mots-clés que JS garde pour plus tard)

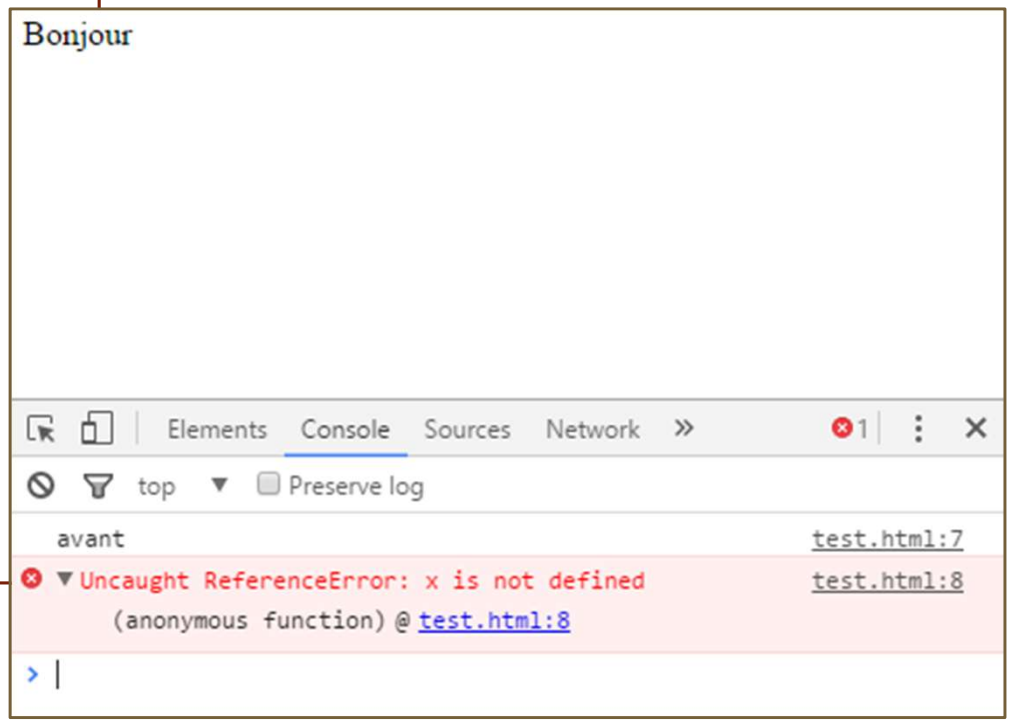
```
//Script entier en mode strict
'use strict';

/* ... instructions ... */
```

```
function strict(){
  // Fonction en mode strict
  'use strict';
}
```

Avant-propos : Mode strict

- Exemple : il manque le var

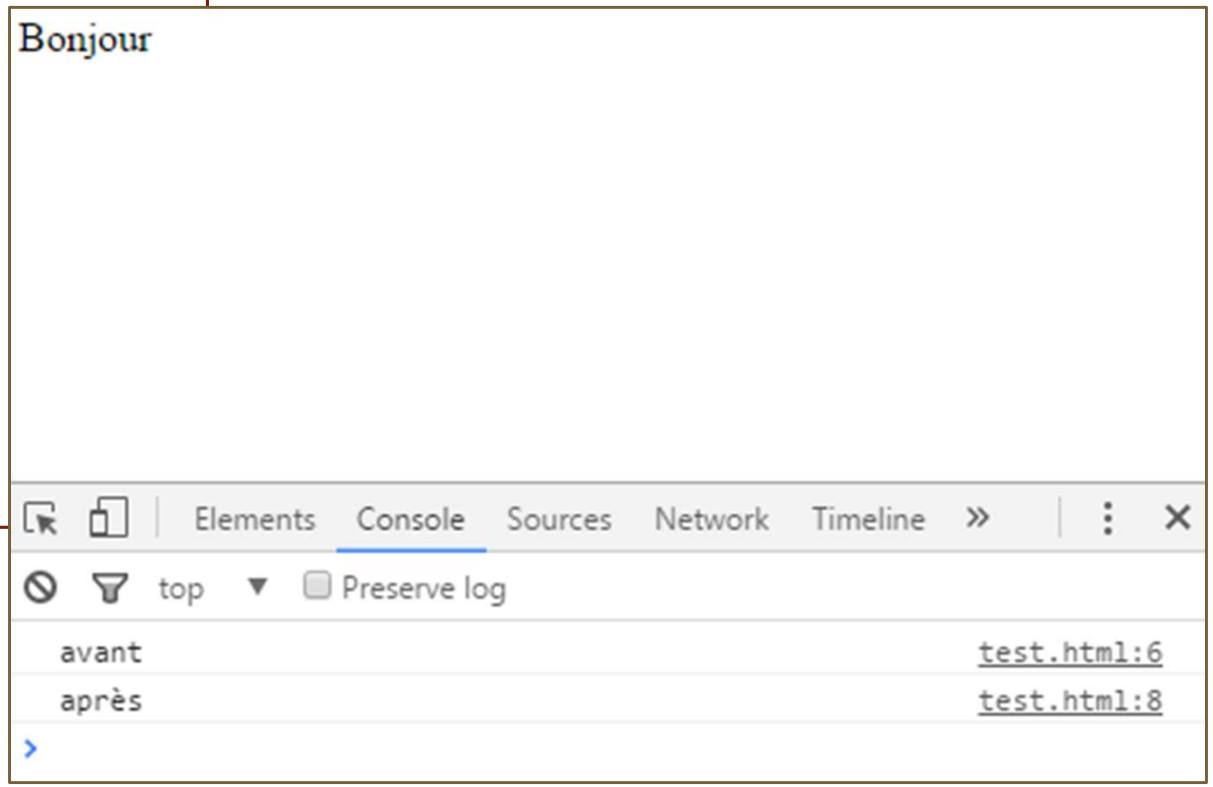
```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      'use strict';
      console.log("avant");
      x = 3.5; // Partira en erreur en mode strict
      // Il faut un var x = 3.5;
      console.log("apr\350s");
    </script>
  </head>

  <body>Bonjour</body>

</html>
```



Avant-propos : Mode NON strict

- Exemple : le var n'est pas obligatoire

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      console.log("avant");
      x = 3.5; // Ne dira rien
      console.log("apr\350s");
    </script>
  </head>

  <body>Bonjour</body>

</html>
```



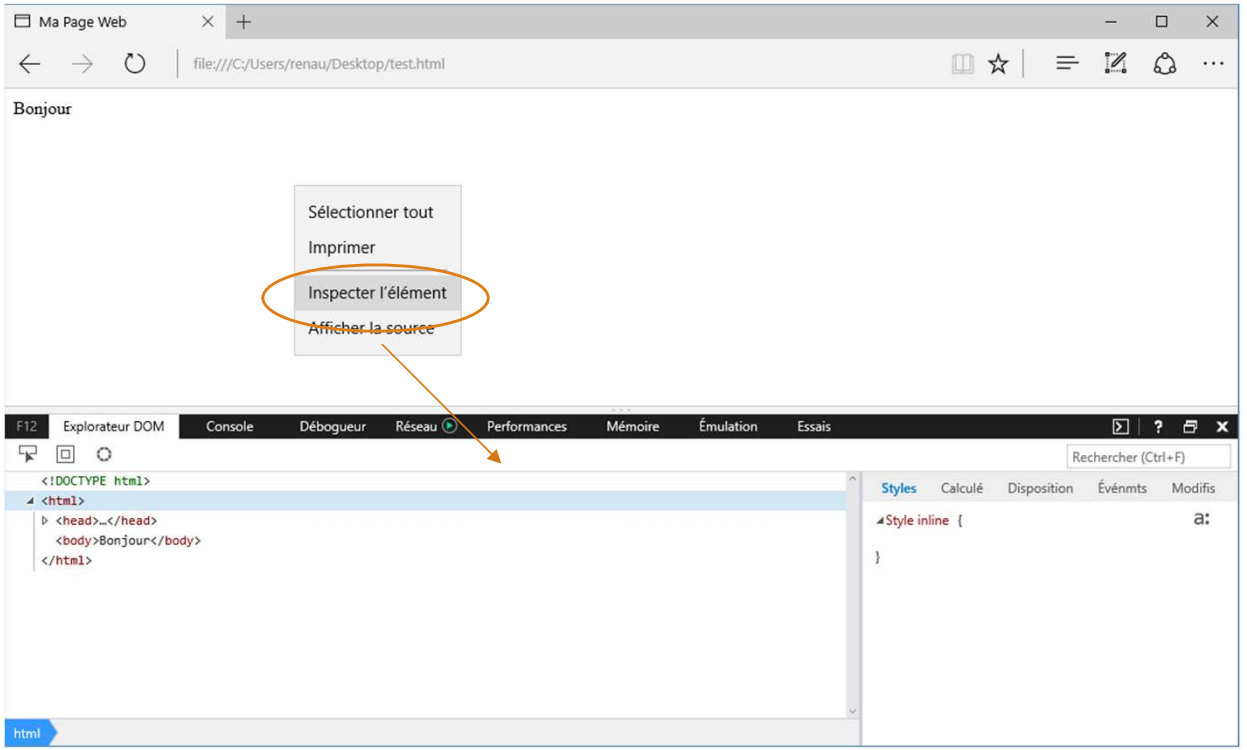
Avant-propos : JSLint / JSHint

- JSLint :
  - Outil d'analyse qualité de code JavaScript.
  - Optimise le code
  - Façon de coder plus propre et plus sûr.
- JSHint se base sur JSLint en étant moins exigeant.
- Pour utiliser JSLint ou JSHint, il faut passer par un module externe (ex : plugin JSHint sous Eclipse) qui analysera le code.
  - http://www.jslint.com/
- D'autres outils :
  - Atom - https://atom.io/
  - Notepad++ - https://notepad-plus-plus.org/fr/
  - Brackets - http://brackets.io/
  - Microsoft Visual Studio Express pour le Web
  - http://blog.reybango.com/the-big-list-of-javascript-css-and-html-development-tools-libraries-projects-and-books/

Avant-propos : Navigateurs

- Pour développer vous aurez besoin de votre navigateur web

- Sous IE
  - Touche F12
  - Ou clic droit puis inspecter l'élément

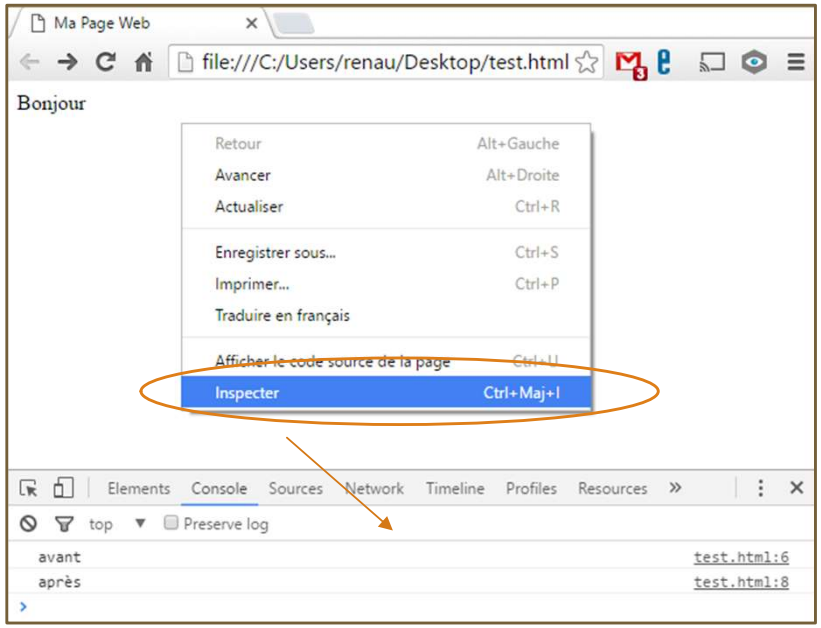


Avant-propos : Navigateurs

- Sous Chrome

- Vous pouvez aussi compléter l'intelligence de votre navigateur avec des plugins dédiés aux Java Script

- Vous pourrez ainsi débuguer ou naviguer plus simplement dans votre code.



Avant-propos : Syntaxe / Nommage

- Les **commentaires** :
  - // commentaire sur une ligne
  - /* commentaire sur plusieurs lignes */
- Les instructions se terminent par « **;** »
  - ce n'est pas obligatoire mais **très fortement encouragé**.
- { } définissent un **bloc** de code
- ( ) servent dans l'appel et la définition des **fonctions**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      console.log('avant');
      console.log('apr\350s');
      // Une fonction
      function faireQQc(uneValeur) {
        if (0 == uneValeur) {
          alert('Bonjour');
        };
      };
    </script>
  </head>

  <body>Bonjour</body>

</html>
```

Avant-propos : Syntaxe / Nommage

- Nommage :
  - JavaScript **est sensible** à la casse (différence entre majuscules et minuscules).
- Nom des fonctions et des variables en camelCase.
  - var le**N**om**D**e**M**a**V**ariable
  - function ma**F**onction**Q**ui**F**ait**QQC**() { … }
- Une constante ou une variable globale sera en majuscule
  - MA_CONSTANTE.
- Une variable au pluriel indique généralement un tableau.
  - mesClients ⇔ mesClients[0] ⇔ premier élément du tableau

Avant-propos : Console

- Vous pouvez envoyer n'importe quel message au navigateur dans sa console via la fonction log() de la variable globale console (avec un *e*)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      console.log('avant');
    </script>
  </head>

  <body>Bonjour</body>

</html>
```

Avant-propos : Console

- La plus part des navigateurs acceptent différents niveaux de trace :

```
console.log("message simple");
console.warn("message d'avertissement");
console.error("message d'erreur");
console.info("message d'info");
```

Avant-propos : Débuguer

- Vous pouvez placer dans votre code le mot clef **debugger** afin de placer un point d'arrêt dans votre code

```
var x = 15 * 5;
debugger;
document.getElementById("demo").innerHTML = x;
```

- Le point d'arrêt sera pris en compte par le navigateur et vous pourrez
  - Évaluer vos variables
  - Faire du pas à pas
  - …

Les bases : Les variables

- Le JavaScript **est non typé**, ce qui veut dire qu'une variable peut avoir n'importe quel type de valeur **et en changer** durant l'exécution du script.

- Il est possible de définir une variable de trois manières :

```
variableGlobale = 12;
var variableLocale = "chaîne de caractères";
this.variableObjet = 12.5;
```

- Sans le mot clé **var**, où que se trouve la variable, elle sera globale au moment de sa définition.

- Avec l'utilisation de var :
  - En dehors d'une fonction, la variable sera globale à toutes les fonctions du même espace.
  - Dans une fonction elle sera locale à la fonction

Les bases : Les variables

- Une variable définie avec **var** dans une fonction est accessible partout dans la fonction après la définition de celle-ci.

```
function uneFonction() {
    var j = "varJ";
    for (var loop = 0; loop < 1; loop++) {
        var variableFor = "variable dans for";
    }
    console.log(j, loop, variableFor );
}
uneFonction();
```

- Affiche :

```
varJ 1 variable dans for
```

Les bases : Les variables

- Une variable définie avec **var** en dehors d'une fonction est accessible par celle-ci, si ces deux conditions sont respectées :
  - La fonction et la variable sont définies dans le même espace.
  - La fonction est appelée après la définition de la variable.

- Une variable globale est accessible partout une fois définie.

```
var vglobal = 'vg0'
globale1 = 'vg1';

function uneFonction() {
    console.log(vglobal, globale1, globale3); // affiche vg0 vg1 vg3
    globale2 = 'vg2';
    console.log(globale4); //erreur car non défini avant l'appel de fonction
}

globale3 = 'vg3'; //remontée de variable, "hoisting"
uneFonction();
globale4 = 'vg4';
console.log(globale2); // affiche : vg2
```

Les bases : Les variables

- Pour éviter les erreurs et garder une cohérence, il est conseillé de limiter les variables globales car il y a « pollution de l'espace de nom ».
  - ⇔Utilisez var tout le temps
- Plus le code est important et complexe, plus il y a de chances que deux variables se portent le même nom.
- Il est conseillé de définir toutes les variables locales au début de la fonction.

```
function uneFonction() {
    var a=12, b=13, boucle=0;
    /* instructions */
}
```

- ES6 introduit pour les variables les mots-clés :
  - const : Ne sont pas des constantes sur des valeurs mais sur des références. Comme avec le mot-clé var en ES5, les variables définies avec const sont locales au bloc.
  - let : Identique au var d'ES5 mais limité à son bloc.

Les bases : Les types

- JavaScript est dit non typé car les variables acceptent n'importe quel type, ce qui ne veut pas dire que les types n'existent pas.
- Il existe 5 types primitifs en JavaScript ES5 + 1 en ES6

Types primitifs	Description	Exemple
number	Entier ou flottant compris entre $-(2^{53}-1)$ et $2^{53}-1$. Il existe 3 symboles : +Infinity, -Infinity, et NaN. NaN pour Not an Number lors d'une conversion ratée.	var a = 12; var b = 0.5;
boolean	Variable ayant pour valeur vrai ou faux.	var a = true; var b = false;
string	Chaîne de caractères. " ou ""	var a = "un texte"; var b = 'un texte';
null	Variable ayant la valeur null.	var a = null;
undefined	Aucune valeur affectée.	var a;
Symbol	Donnée unique et non modifiable.	var a = Symbol();

- Il existe un autre type, les objets : Object.
  - Exemple var a = {};

Les bases : Les opérateurs

Operateurs mathématiques	Description	Exemple
+	Addition	a = 1 + 2;
-	Soustraction	a = 3 – 2:
/	Division	a = 3 / 2; Donne 1.5
*	Multiplication	a = 3 * 2;
%	Modulo. Reste d'une division d'entiers.	a = 5 % 3; Donne 2
++	Incrémentation. ++i pré-incrémentation (a lieu immédiatement) i++ post-incrémentation (a lieu après l'instruction)	++i; j++;
--	Décrémentation --i pré-décrémentation (a lieu immédiatement) i-- post-décrémentation (a lieu après l'instruction)	--i; j--;
-	Négation	a = 1; b = -a;

+	Plus unaire. Généralement pour une conversion Implicite.	a = "12"; // String
		b = +a; // Number

Les bases : Les opérateurs

Operateurs comparaisons	Description	Exemple

==	Égalité. Renvoie TRUE si même valeur	12 == 12; // vrai
		12 == "12"; // vrai

!=	Inégalité. Renvoie TRUE si les valeurs sont différentes	12 != 13; // vrai
===	Égalité stricte. Renvoie TRUE si mêmes valeurs et mêmes types.	12 === "12"; // faux

!==	Inégalité stricte. Renvoie True si les valeurs sont différentes OU les types sont différents	12 !== "12"; // true
		13 !== 14; // true

>	Supérieur	13>10; // true
		10>10; // false

>=	Supérieur ou égal	10>=10; // true
		9>=10; //false

<	Inférieur	9 < 9; // false
<=	Inférieur ou égal	9<=9; // true

Les bases : Les opérateurs

Operateurs logiques	Description	Exemple
&&	ET logique. Expression1 && expression2 donne vrai si les deux expressions sont vraies	true && false => false
||	Ou logique. Expression1 && expression2 donne vrai si l'une des expressions est vraie	true || false => true
!	Non logique. Inverse la valeur d'une expression booléenne	A = true; !a => false

Les bases : Les opérateurs

Operateurs binaires	Description
a & b	Opérateur ET sur chaque bits des deux valeurs
a | b	Opérateur ou sur chaque bits des deux valeurs
a ^ b	Égalité stricte. Renvoie true si mêmes valeurs et mêmes types.
~ a	Inégalité stricte. Renvoie true si les valeurs sont différentes OU les types sont différents
a << n	Décalage des bits de a vers la gauche de n fois
a >> n	Décalage des bits de a vers la droite de n fois
a >>> b	Décalage des bits de a vers la gauche de n fois avec complément de 0 à gauche.

Les bases : Les opérateurs


```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      var a = 8;  // 01000
      var b = 12; // 01100
      console.log('a&b='+(a&b)); // 8  = 01000
      console.log('a|b='+(a|b)); // 12 = 01100
      console.log('a^b='+(a^b)); // 4  = 00100
      console.log('~a='+(~a));   // -9 =-01001
      console.log('a<<2='+(a<<2));   // 32 = 100000
      console.log('a>>2='+(a>>2));   // 2 = 10
      console.log('a>>>2='+(a>>>2)); // 2 = 10
    </script>
  </head>

  <body>Bonjour</body>
</html>
```

Les bases : Les opérateurs

Operateurs unaires	Description	Exemple
delete	Supprime un objet, une propriété d'un objet ou un tableau	delete objet;
typeof	Retourne une chaîne de caractères indiquant le type de l'élément.	typeof true; Retourne "boolean"
void	Indique qu'une expression doit être évaluée sans valeur de retour, retourne undefined	void(0)

Les bases : Les opérateurs

Operateurs d'affectations	Raccourcis de :
x = y	Affectation
x += y	x = x + y
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x **=y	x = x ** y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

Les bases : Les opérateurs

Type d'opérateur	Opérateurs individuels
membre	. []
appel/création d'instance	() new
négation/incrémentation	! ~ - + ++ -- typeof void delete
multiplication/division	* / %
addition/soustraction	+ -
décalage binaire	<< >> >>>
relationnel	< <= > >= in instanceof
égalité	== != === !==
ET binaire	&
OU exclusif binaire	^
OU binaire	|
ET logique	&&
OU logique	||
conditionnel	?:
assignation	= += -= *= /= %= <<= >>= >>>= &= ^= |=
virgule	,

TP01

- V0 : Réalisez une première page Web en HTML 5 qui affiche simplement en JavaScript dans la console : « Bonjour tout le monde ! »
- V1 :Placez le texte « Bonjour tout le monde ! » dans une variable
- V2 : Placez votre code JavaScript dans un fichier JS importé par la page HTML

Les bases : Les tableaux

- JavaScript étant non typé, les tableaux acceptent tous les types de valeurs.

```
var tab0 = [ "a", "b", null ];
var tab1 = new Array( 0, "texte", 3.0 );
var tab2 = [ "a", tab0, tab1 ];
```

- Chaque élément d'un tableau a un indice, ce qui veut dire un numéro désignant sa position.

```
console.log(tab0[0]); // affiche: a
console.log(tab1[2]); // affiche: 3.0
```

- Le premier élément se trouve à l'indice 0.

Les bases : Les tableaux

- Modifier une valeur dans un tableau :

```
tab0[0] = 12;
```

- Récupérer une valeur d'un tableau :

```
console.log(tab0[0]); // affiche: 12
```

- Vous pouvez connaître la taille d'un tableau grâce à sa propriété : length

```
console.log(tab0.length); // affiche 3
```

Les bases : Les tableaux

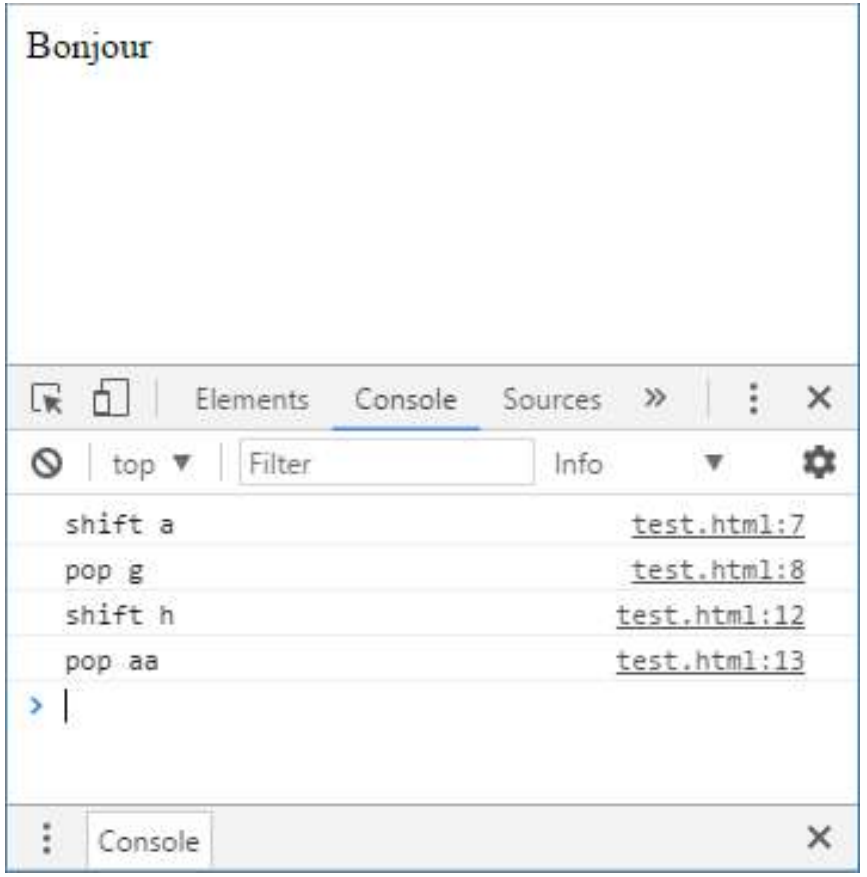
- Exemple de fonctions sur les tableaux :

Fonctions sur les tableaux	Description
sort	Trie le tableau. sort() ne fonctionne qu'avec les chaines, sinon faire usage de sort(function(a,b) {…}) pour les autres types.
shift	Retourne le premier élément et le supprime du tableau
push	Ajoute un élément à la fin du tableau
pop	Retourne le dernier élément et le supprime du tableau
join	Les éléments sont mis à la suite des autres dans une chaîne de caractères
unshift	Ajoute un élément en début de tableau
reverse	Inverse les éléments

```
var fruit = ["pommes", "bananes", "Cerises"];
fruit.sort(); // ["Cerises", "bananes", "pommes"];
```

Les bases : Les tableaux


```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      var tab = ['a', 'b', 'c', 'd', 'e', 'f', 'g'];
      console.log('shift '+tab.shift()); // a
      console.log('pop '+tab.pop()); // g
      tab.push('h');
      tab.unshift('aa');
      tab.reverse();
      console.log('shift '+tab.shift()); // h
      console.log('pop '+tab.pop()); // aa
    </script>
  </head>

  <body>Bonjour</body>
</html>
```



Les bases : Les conditions if


```
if (conditionA) {
  // instructions à exécuter si la conditionA est vraie
}
```

```
if (conditionB ) {
  // instructions à exécuter si la conditionB est vraie
} else {
  // instructions à exécuter si la conditionB est fausse
}
```

```
if (conditionC) {
      // instructions à exécuter si la conditionC est vraie
} else if (conditionD) {
      // instructions à exécuter si la conditionC est fausse  et si la
      conditionD est vraie
}
```

Les bases : Les conditions if


```
if (conditionC) {
    // instructions à exécuter si la conditionC est vraie
} else if( condition D ) {
    // instructions à exécuter si la conditionC est fausse
    // et si la conditionD est vraie
} else {
    // instructions à exécuter si les conditionC
    // et conditionD sont fausses
}
```

Les bases : Les conditions if

- Vous pouvez utiliser les valeurs **null** ou **undefined** comme test, mais attention :

```
[if (a)] - a EST null
[if (a !== null)] - a n'est PAS null
[if (a == null)] - a est null
[if (a === undefined)] - a est undefined
```

```
// Ici a est undefined (en ===), et null (en ==)
var a;
if (a) {
        console.log("[if (a)] - a n'est PAS null");
} else {
        // C'est ici que l'on passe
        console.log("[if (a)] - a EST null");
}

if (a !== null) {
        // C'est ici que l'on passe, car a est undefined
        console.log("[if (a !== null)] - a n'est PAS null");
} else {
        console.log("[if (a !== null)] - a EST undefined");
}

if (a == null) {
        // C'est ici que l'on passe, car a est null ou undefined si on n'utilise pas ===
        console.log("[if (a == null)] - a est null");
} else {
        console.log("[if (a == null)] - a n'est PAS null");
}

if (a === undefined) {
        // C'est ici que l'on passe
        console.log("[if (a === undefined)] - a est undefined");
} else {
        console.log("[if (a === undefined)] - a n'est PAS undefined");
}
```

Les bases : Les conditions sur les variables

- En JavaScript il est possible d'utiliser une variable autre qu'un booléen comme condition.

- Le test se portera sur le type, la valeur ou l'existence de la variable.

Valeur	Équivalence en booléen
0	false
1	true
-1	true
'texte'	true
""	false
null	false
undefined	false
NaN	false

Les bases : Les conditions switch

- La condition switch test s'il y a égalité entre une variable avec différentes valeurs.

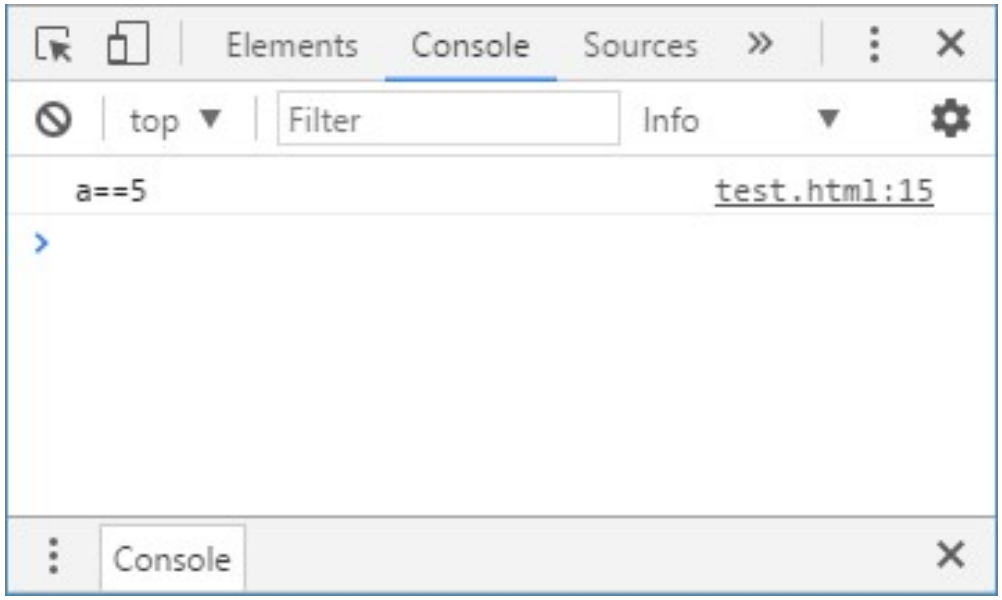
- Syntaxes :

```
switch( variable ) {
  case valeur1:    //  est-ce que la valeur de la variable est égale à valeur1 ?
    console.log(" instruction à executer si la variable est égale à la valeur1 ");
    break; // sort du switch après avoir executé les instructions
  case valeur2:    //  est-ce que la valeur de la variable est égale à valeur2 ?
    console.log(" instruction à executer si la variable est égale à la valeur2 ");
    break;  // sort du switch après avoir executé les instructions
  default:
    console.log(" instructions à exécuter sinon ");
}
```

- Attention à ne pas oublier le break !

Les bases : Les conditions switch

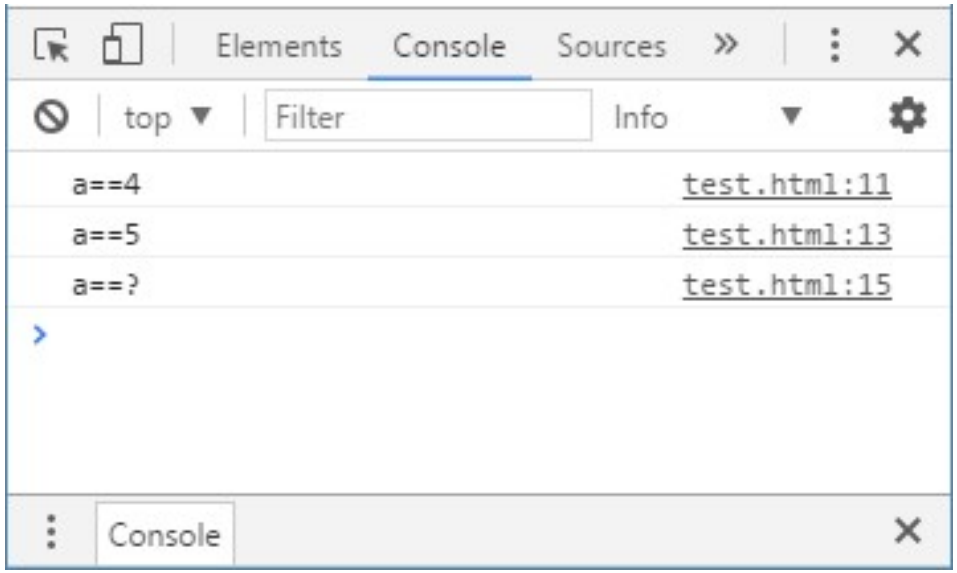

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      var a = 5;
      switch(a) {
        case 2:
          console.log("a==2");
          break;
        case 4:
          console.log("a==4");
          break;
        case 5:
          console.log("a==5");
          break;
        default:
          console.log("a==?");
      }
      </script>
  </head>

  <body>Bonjour</body>
</html>
```



Les bases : Les conditions switch

- L'instruction break permet de sortir du switch.
- Si le break est omis, les instructions seront toutes lues jusqu'à qu'un break soit atteint ou que la fin du switch soit atteinte.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
    <script>
      var a = 4;
      switch(a) {
        case 2:
          console.log("a==2");
        case 4:
          console.log("a==4");
        case 5:
          console.log("a==5");
        default:
          console.log("a==?");
      }
      </script>
  </head>

  <body>Bonjour</body>
</html>
```



Les bases : La condition ternaire

- La condition ternaire correspond à un : if (…) {…} else {…} ne contenant qu'une instruction dans chaque bloc.
- Syntaxe : condition ? Instruction si vraie : instruction si faux ;
- Les deux codes suivants sont équivalents :

```
var a = 12;
var b = 13;

12 > 13 ? console.log("vrai") : console.log("faux");
```

```
var a = 12;
var b = 13;

if( 12 > 13 ) {
    console.log("vrai");
} else {
    console.log("faux");
}
```

Les bases : La condition ternaire

- La condition ternaire est souvent utilisé pour fournir une valeur à une variable à partir d'une condition sans passer par un if/else.

```
var a = true || false ? 15 : 30;
console.log(a);
```

```
var a;
if (true || false) {
  a = 15;
} else {
  a = 30;
}
console.log(a);
```

Les bases : Les boucles - for

- **Définition** : une boucle contient une série d'instructions exécutées jusqu'à ce qu'un résultat particulier soit obtenu ou qu'une condition prédéterminée soit remplie.

- Les boucles permettent de réutiliser des séries d'instructions et permettent ainsi de limiter le nombre d'instructions.

- Syntaxe :

for(**initialisation** ; **condition avant** itération ; **instruction après** itération) {

    // instructions à répeter

}

- Exemple :

```
for( var cpt=0; cpt<4; cpt++ ) {
    console.log("Compteur: "+cpt);
}
```

Les bases : Les boucles - for … in

- La boucle for… in permet de faire des itérations sur un tableau ou un objet
  - Attention : cle = index (ce n'est pas la valeur de la case)
- Exemples :

```
var tab = [ "a", "b", null];

for (var cle in tab) {
    console.log( 'valeur : ' + tab[cle]);
}
```

Résultat :

```
valeur: a
valeur: b
valeur: null
```

```
var obj = {
    a: "texte",
    b: 12.5,
    c: function() { console.log('rien'); }
};

for (var val in obj) {
    console.log( 'valeur : ' + obj[val]);
}
```

Résultat :

```
valeur : texte
valeur : 12.5
valeur : function () { console.log('rien'); }
```

Les bases : Les boucles - while et do…while

- while(condition d'itération) { // instructions }

```
var cpt=0;
while (cpt<4) {
  console.log("Compteur: "+cpt);
  cpt++;
}
```

- do { // instructions } while(condition d'itération)

```
var cpt=0;
do {
  console.log("Compteur: "+cpt);
  cpt++;
} while (cpt<4);
```

- While : exécuté **0** ou n fois
- Do while : exécuté **1** ou n fois

Les bases : break et continue

- L'instruction break permet aussi de sortir des boucles.

```
for (var i=0 ; i<6 ; i++) {
  if (i == 3) {
    break;
  }
  console.log("Compteur: "+i);
}
```

affiche :

```
Compteur: 0
Compteur: 1
Compteur: 2
```

- L'instruction continue passe directement à l'itération suivante.

```
for (var i=0 ; i<6 ; i++) {
  if (i == 3) {
    continue;
  }
  console.log("Compteur: "+i);
}
```

affiche :

```
Compteur: 0
Compteur: 1
Compteur: 2
Compteur: 4
Compteur: 5
```

- **continue** et **break** s'appliquent sur la boucle où ils se trouvent.

Les bases : break et continue (le retour du goto)

- Une étiquette est un repère dans le code.
- Il est possible d'utiliser des étiquettes avec les instructions break et continue pour sortir de plusieurs boucles à la fois.

```
sortie:for (var h=0 ; h<2 ; h++) {
    for (var i=0 ; i<3 ; i++) {
        if (i == 2) {
            break sortie;
        }
        console.log("Compteur: "+i);
    }
}
```

affiche :

```
Compteur: 0
Compteur: 1
```

```
sortie:for (var h=0 ; h<2 ; h++) {
    for (var i=0 ; i<3 ; i++) {
        if (i == 1) {
            continue sortie;
        }
        console.log("Compteur: "+i+" "+h);
    }
}
```

affiche :

```
Compteur: 0 0
Compteur: 0 1
```

TP02

- Dans un fichier à part, réalisez un code en JavaScript qui
  - Déclare un tableau de 10 cases
  - Via une boucle, l'initialise de la manière suivante :
    - Première case : valeur 2
    - Seconde case : valeur 4
    - Troisième case : valeur 6
    - …
    - Dernière case : valeur 20
  - Affiche le tableau dans la console
  - Affiche la somme suivante sur la console (une case sur deux)
    - tab[0]+tab[2]+tab[4] …

Les bases : Les fonctions

- Une fonction (**function**) permet de mettre en commun du code afin d'éviter de le copier/coller partout où on en a besoin.
- Elle porte un nom clair et explicite, en commençant par une minuscule.
  - ***function*** **c**alculer**T**va(unChiffre) { … }
- Elle représente un ensemble d'instructions réunis dans un même bloc ({}).
- Elle peut avoir des valeurs en entrées, appelées **paramètres**.
- Elle renvoie ou non un résultat via le mot clef **return**.

Les bases : Les fonctions

- La signature (ou le prototypage) d'une fonction respecte le schéma :
  - function nomDeLaFonction([params]) { … }
- **function** (à l'anglaise) et pas ~~fonction~~
- La signature n'indique pas si la fonction renvoie ou non une valeur.
- La signature n'indique pas non plus si la fonction renvoie une exception

```
function faireA() {
    // instructions
}

function faireB(a, b) {
    // instructions
}

function faireC() {
    // instructions
    return "chaine de caractères";
}
```

Les bases : Les fonctions

- Les paramètres d'une fonction ne sont jamais obligatoire
  - Un fonction peur avoir 3 paramètres ⇔on peut l'appeler avec 0, 1, 2, 3 paramètres

```
function faire(p1, p2, p3) {
    // instructions
}

faire();
faire(1);
faire(1, 2);
faire(1, 2, 3);
```

Les bases : Les fonctions

- Tout comme les variables, les fonctions définies dans un script sont dans l'espace globale de nom.
- Il y a pollution de cet espace globale.

- En JavaScript, il est possible de définir des fonctions dans des fonctions.
  - Ces fonctions deviennent des fonctions locales et évitent la pollution de l'espace global.

```
function fctGlobale() {
     function fctLocale(){
          console.log('une fonction');
     };

     fctLocale(); // affiche : 'une fonction'
}

fctGlobale();
```

Les bases : Les fonctions

- Une fonction peut-être anonyme, c'est-à-dire ne pas avoir de nom.

- En Java Script il est permis de définir une fonction dans une variable :
  - Ci-dessus nous avons une fonction anonyme, mais la variable qui la contient peut être appelée comme une fonction.

```
var fct = function () {
    var a = "une variable";
    console.log(a);
}

fct(); //affiche "une variable"
```

Les bases : Les fonctions

- Une fonction anonyme peut être utilisée de plusieurs manière :
- Les parenthèses rouges indiquent une expression à exécuter et les bleus lancent l'exécution.

```
(function() {
    console.log("appel d'une fonction anonyme");
})();
```

- Ceci est équivalent à :

```
function uneFonction() {
    console.log("appel d'une fonction");
} uneFonction();
```

Les bases : Les fonctions

- Il est possible de fournir une fonction comme paramètre à une autre fonction :

```
function uneFonction(fct) {
    fct();
};

function affiche() {
    console.log("une fonction");
};

uneFonction(affiche);
```

```
function uneAutreFonction(fct) {
    fct(12);
};

function afficheValeur(val) {
    console.log(val);
};

uneAutreFonction(afficheValeur);
```

Les bases : Closure (fermeture)

- Une *fermeture* est une fonction qui capture des informations à un instant T.

- L'écriture ci-dessus fait usage d'une référence à la variable « a » dans la fonction « affiche ».

```
var a = 12;
function affiche() {
    console.log(a);
};
```

- La variable « a » sera affichée sur la console à chaque appel de « affiche », mais *a* est une variable globale.

Les bases : Closure (fermeture)

- Comme vu précédemment, il est possible de fournir une fonction en argument d'une autre.
- Nous pouvons donc faire :

```
function creerFonction() {
  var a = 12;
  function affiche() {
    console.log(a);
  };
  return affiche;
}

// maFonction contient une closure
// Elle contient son propre environnement (variable a + fonction affiche)
var maFonction = creerFonction();
// Appel de la closure
maFonction();
```

Les bases : Closure (fermeture)

- Une fermeture permet d'associer des données (l'environnement) avec une fonction qui agit sur ces données.

- On peut faire un parallèle avec la programmation orientée objet car les objets permettent d'associer des données (les propriétés) avec des méthodes.

- On peut ainsi utiliser une fermeture pour tout endroit où on utiliserait un objet qui n’a qu’une seule méthode.

Les bases : Closure (fermeture)

- Exemple

```
function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size + 'px';
  };
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);

document.getElementById('size-12').onclick = size12;
document.getElementById('size-14').onclick = size14;
document.getElementById('size-16').onclick = size16;
```

TP03

- Dans un fichier à part, réalisez un code en JavaScript qui
  - Déclare un tableau et l'initialise de la manière suivante : [34, -2, 6, 9, 34, 25, -10, 43]
  - Déclare une fonction *chercherMin* qui prendra comme paramètre un tableau. Dans la fonction :
    - Vérifie que la tableau n'est pas null (sinon affiche une erreur)
    - Retourne la plus petite valeur trouvée dans le tableau
  - Affichez le résultat de l'appel de chercherMin en lui passant le tableau comme paramètre

TP03 - suite

- Ajoutez une méthode chercherIndexMaxAbs qui va chercher **l'index de la case** d'un tableau possédant le chiffre le plus grand en valeur absolue
  - Vous pouvez faire usage de la fonction abs() de l'objet Math Javascript
    - Math.abs(-5) <=> 5

Les objets

- Un objet est une structure qui encapsule des attributs et des fonctions (appelées méthodes).
  - Il permet une réutilisation simple du code.
- Dans d'autres langages, les objets ont un schéma de constructions ⇔ les classes.
  - Lorsque nous créons un objet en suivant ce schéma, nous parlons d'instance de classe (objet et instance sont donc synonymes).
- Il n'y a pas de classes en JavaScript, mais des instances d'objets.
- Les éléments qui ne sont pas de type primitifs sont des objets en JavaScript, fonctions comprises.
- Les objets sont transmis par référence :
  - S'ils sont modifiés dans une fonction, cette modification **est sauvegardée** dans l'objet.

Les objets - Attributs

- Exemple d'un objet simple et de la définition de ses attributs :

```
var obj = {
    attribut0: "a",
    attribut1: 2.5,
    attribut2: 16
};
```

- Les objets définis ainsi, sont appelés objets littéraux.

- Chaque élément qui compose l'objet est séparé par une virgule.
  - Le dernier élément n'a pas obligation de se terminer par une virgule

- Ils peuvent être utilisé comme tableau associatif (parfois appelé table de hachage ou dictionnaire).

Les objets - Attributs

- Pour accéder aux éléments d'un objet :
  - Usage des clefs (écriture lourde)

```
console.log(obj['attribut0']); // affiche: a
obj['attribut0'] = 154;
console.log(obj['attribut0']); // affiche: 154
```

  - Il est possible d'utiliser la notation pointée sur un objet (écriture conseillée):

```
console.log(obj.attribut1); // affiche: 2.5
obj.attribut1 = "texte";
console.log(obj.attribut1); // affiche: texte
```

Les objets - Méthodes

- Une méthode est simplement un attribut de type function

- Définir une méthode à un objet :

```
var obj = {
    attribut0: "a",
    attribut1: 2.5,
    methode0: function() {
        console.log("méthode sans paramètre");
    },
    methode1: function(param0) {
        console.log("méthode avec paramètre");
        console.log(param0);
    }
};

obj.methode0();
obj.methode1('une valeur');
```

Les objets - this

- A l'intérieur d'un objet, une méthode peut utiliser le mot clé this qui fait référence à l'objet courant ou utiliser le nom de l'objet pour atteindre un champ.

```
var obj = {
     attribut: "a",
     methode0: function() {
          console.log("méthode0 appelée");
     },
     methode1: function() {
          console.log("méthode1 appelée");
     },
     methode2: function() {
          console.log("méthode2 appelée");
          this.methode0();       // appel de la méthode avec le mot clé this
          obj.methode1();        // appel de la méthode avec le nom de l'objet
          console.log(obj.attribut); // appel de l'attribut avec le nom de l'objet
     }
};

obj.methode2();
```

Les objets - this

- this n'est utilisable qu'a l'intérieur d'une définition d'un objet.
- this représente l'instance courante.
- Il est conseillé d'utiliser le mot clé this.

```
var obj = {
     attribut: "a",
     methode0: function() {
          console.log("méthode0 appelée");
     },
     methode1: function() {
          console.log("méthode1 appelée");
     },
     methode2: function() {
          console.log("méthode2 appelée");
          this.methode0();
          this.methode1();
          console.log(this.attribut);
     }
};
// il n'est pas possible d'écrire this ici
obj.methode2();
```

Les objets avec constructeur

- En JavaScript les fonctions sont aussi des objets et peuvent être utilisées comme des classes.
- Dans ce cas, la fonction sert de constructeur.
- Par convention, les fonctions servant de classe commence par une majuscule.
- Pour appeler le constructeur on fera usage du mot clef **new**

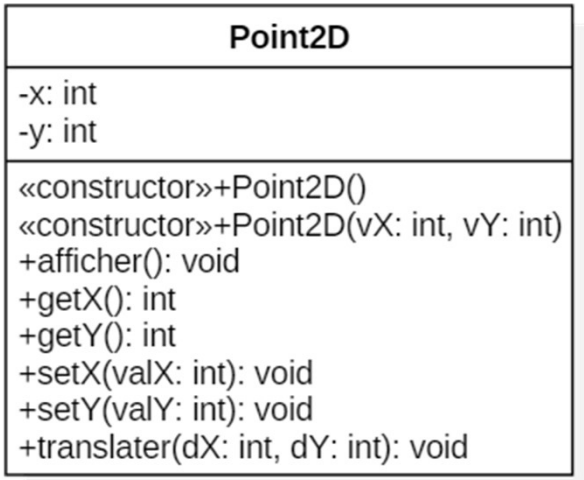
```
function MaClasse() {
    console.log("initialisation");
    this.attribut0 =  0;
    this.attribut1 = 'a';

    this.methode = function() {
        console.log("une méthode");
    };
}
// appel via constructeur
var obj = new MaClasse();
obj.methode();
```

Les objets avec constructeur

- **<u>Attention</u>** : quand on fait usage d'une fonction comme constructeur :
  - Les attributs sont **dans** la fonction constructeur
    - On utilise '=' et plus ':'
  - Les méthodes sont **dans** la fonction constructeur
    - On utilise '=' et plus ':'
  - Chaque élément est séparé par un ';' et non plus une ','

TP04

- Réalisez dans un code JavaScript à part un objet Point2D
  - Il aura les attributs x, y
  - Un constructeur qui prend des valeurs pour x et y
    - Attention, elles peuvent être ‘undefined’
  - Les méthodes
    - afficher() : affiche [x,y] sur la console
    - translater(dX, dY) :On est ici sur une translation vectorielle, la valeur de x devient x+dX et la valeur de y se transforme en y+dY.
    - getX, getY : retournent les valeurs des attributs
    - setX, setY : modifient les valeurs
  - Instanciez des Point2D et appelez vos méthodes.



Les objets du JavaScript

- Vous avez quelques objets déjà présent dans le JavaScript de base
  - window
  - string
  - date
  - math
- Il y en a d'autre :
  - https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux

Les principaux objets - Window

- L'objet Window
  - window.open : ouvre une nouvelle fenêtre

```
window.open("fichier.htm","fenetre1","width=310,height=400,left=0,top=0");
```

  - window.location.href : URL de la page

```
window.location.href = "fichier2.htm";
```

  - window.location.**reload**() : recharge la page
  - window.history.**back**() : revenir à la page précédente
  - window.history.**forward**() : aller à la page suivante
  - window.**document** : obtient le document courant

- window est une variable globale
  - Crée en même temps que la fenêtre
  - Appelable de n'importe quel endroit (page, fichier JS séparé)

Les principaux objets - Date

- L'objet Date

```
var uneDate = new Date();
```

  - uneDate.**getDate**() : retourne le jour du mois
  - uneDate.**getMonth**() : retourne le mois
  - uneDate.**getFullYear**() : retourne l'année complète
  - uneDate.**getHours**() : retourne l'heure
  - uneDate.**getMinutes**() : retourne les minutes
  - uneDate.**getSeconds**() : retourne les secondes

- Chaque méthode ci-dessus a un équivalent setXXX pour définir une valeur
  - uneDate.**setDate**(18) : définir le jour du mois
  - uneDate.**setFullYear**(1995) : définit l'année
  - etc.

Les principaux objets - String

L'objet String

```
var uneChaine = "Formation";
```

- uneChaine.**length** : longueur de la chaîne
- uneChaine.**indexOf**("chaine") : position d'une chaîne dans uneChaine
- uneChaine.**replace**(expReg, "nouveau") : remplacer un contenu dans la chaîne par une autre
- uneChaine.**substr**(début, nbCar) : extraire un sous-ensemble de la chaîne
- uneChaine.**substring**(début, fin) : extraire un sous-ensemble de la chaîne
- uneChaine.**split**("car") : coupe une chaîne pour en faire un tableau

Les exceptions - Error

- Une exception : Cas exceptionnel ou erreur dans le programme.
- Il est possible de générer une exception lorsque le fonctionnement du programme est compromis.
- Quand une exception est levée, les instructions qui suivent ne sont plus exécutées.
- Il est possible de proposer un moyen d'éviter la fin brutale du programme en proposant une solution, dans ce cas nous "attrapons l'exception".
- Les exceptions sont représentées par un l'objet Error qui possède des attributs
  - **name** : le nom de la classe de l'erreur
  - **message** : le message de l'erreur
  - **fileName** : le nom du fichier où l'erreur s'est produite
  - **lineNumber** : le numéro de la ligne où l'erreur s'est produite
  - **stack** : la pile d'appels

Les exceptions

- Interception globale :
  - JavaScript propose un moyen d'attraper toutes les exceptions

```
function afficheErreur (errorMsg, urlScript, lineNumber, colonne, errorObj) {
    console.log('Message: ' + errorMsg + ' Script: ' + urlScript + ' Ligne: ' + lineNumber+ ' Colonne: ' +
    colonne + ' StackTrace: ' +  errorObj);
}

window.onerror = afficheErreur;
```

  - Cette solution est acceptable lors des phases de **développement**.

Les exceptions - Ratrapper

- Pour attraper une erreur, il faut placer le code à risque dans un bloc **try**{…}.
- Le bloc **catch** sera exécuté en cas d'erreur.
- Le bloc **finally** sera exécuté dans tous les cas.

```
try {
    console.log(d); //Cette variable est inexistante
} catch(error) {
    console.log("Une exception a été attrapée");
    console.log("Nom de l'exception :" + error.name);
    console.log("Message de l'exception :" + error.message);
} finally {
    console.log("S'affichera avec ou sans exception");
}
```

Les exceptions - Lever

- Pour lancer/lever une exception :
  - **throw** new Error("Message de mon Exception");

```
try {
    throw new Error("test"); // exception lancée
} catch(error) {
    console.log("Une exception a été attrapée");
    console.log("Nom de l'exception :" + error.name);
    console.log("Message de l'exception :" + error.message);
} finally {
    console.log("S'affichera avec ou sans exception");
}
```

- Exception levée ⇔return, la méthode, le flux d'instruction est cassé

TP05

- Reprenez le code du tp4, levez une exception quand x ou y sont négatifs

- Testez en appelant vos méthode setX et setY avec des valeurs négatives

les évènements HTML

- La plus part des éléments HTML supportent des évènements

- Chaque famille d'élément à ses évènements.

- Par exemple sur un élément de formulaire :
  - **onblur** : lors de la perte de focus
  - **onchange** : lors du changement d'état
  - **onclick** : lors du clic de souris

- http://www.w3schools.com/tags/ref_eventattributes.asp

les évènements HTML

- Vous pouvez attacher du code Java script directement à un évènement HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ma Page Web</title>
  </head>

  <body>
    <form name="contactForm"
          onsubmit="console.log('soumission'); return false;">

        <input type="text" name="lo" onblur="faireQQc(this);"/>
        <button type="submit">Ok</button>
    </form>
  </body>

  <script>
    function faireQQc(unElm) {
      unElm.value = 18;
    }
  </script>
</html>
```

Navigation dans le DOM

- **D**ocument **O**bject **M**odel

- Le DOM fournit une API pour contrôler un navigateur et accéder au contenu d'une page web:
  - Trouver et paramétrer des éléments d'une page
  - Gérer les évènements de contrôles d'une page
  - Modifier les styles associés à des éléments
  - Serializer et deserializer une page comme un document XML
  - Valider et mettre à jour des pages web

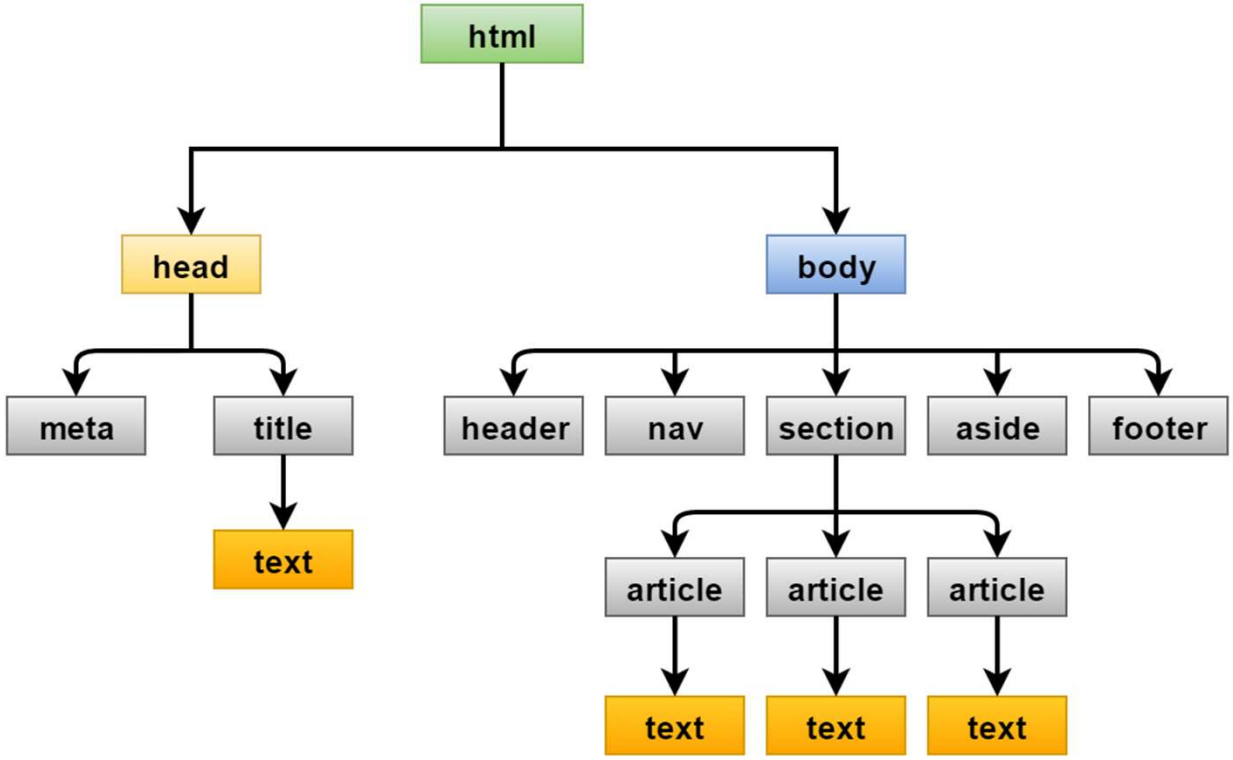
HTML5 : Le DOM

- Exemple de page HTML :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Titre de ma page</title>
    </head>

    <body>
        <header></header>
        <nav></nav>

        <section>
            <article>Texte de l'article 1</article>
            <article>Texte de l'article 2</article>
            <article>Texte de l'article 3</article>
        </section>

        <aside></aside>
        <footer></footer>
    </body>
</html>
```

HTML5 : Le DOM

- Représentation du DOM de la page HTML :



DOM simplifié (seul les éléments HTML sont représentés)

Trouver des éléments dans le DOM

- Soit le formulaire suivant:

```
<form name="contactForm">
  <input type="text" name="nameBox" id="nameBoxId" />
</form>
```

- Référencez le formulaire d'une des façons suivantes:

```
document.forms[0]   // collection qui commence à zéro
document.forms["contactForm"] // utilisation du name
document.forms.contactForm // utilisation du name
document.contactForm // utilisation du name
```

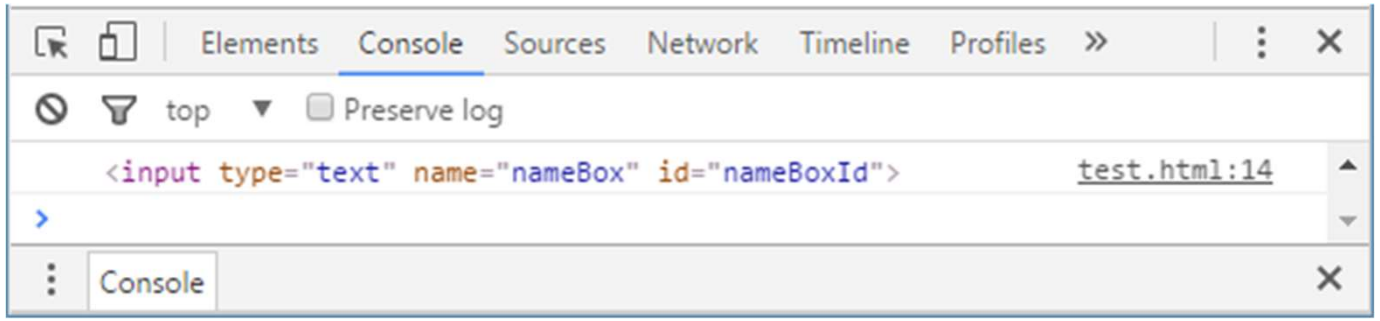
- document : représente la page web où vous êtes.

Trouver des éléments dans le DOM

- Référencez un élément du formulaire comme ci-dessous :

```
document.forms.contactForm.elements[0]
document.forms.contactForm.elements["nameBox"]
document.forms.contactForm.nameBox
document.contactForm.nameBox
document.getElementById("nameBoxId")
```

- Ses méthodes vous retournent un objet qui représente le champ de saisie



Trouver des éléments dans le DOM

- Pour récupérer / modifier la valeur d'un champ de saisie :
  - Faire usage de son **attribut** value
- Pour récupérer / modifier le nom d'un champ de saisie :
  - Faire usage de son **attribut** name
- Pour récupérer / modifier le type d'un champ de saisie :
  - Faire usage de son **attribut** type

```
document.forms.contactForm.elements[0].value = 18;
console.log(document.forms.contactForm.elements["nameBox"].name);
console.log(document.forms.contactForm.nameBox.type);
console.log(document.contactForm.nameBox.name);
console.log(document.getElementById("nameBoxId").value);
```

Modifier des éléments dans le DOM

- Pour modifier un élément d'une page:
  - Créez un nouvel objet contenant la nouvelle donnée.
  - Trouvez l'élément parent qui doit contenir la nouvelle donnée.
  - Ajoutez, insérez ou remplacez la valeur de l'élément par la nouvelle donnée.

```
<ul id="VenueList">
  <li>Room A</li>
  <li>Room B</li>
</ul>
```

```
<script>
  // On cible l'élément via son id
  var list = document.getElementById("VenueList");
  // on fabrique un nouvel élément
  var newItem = document.createElement("li");
  // On ajoute dans cet élément du texte
  newItem.textContent = "Room C";
  // On ajoute l'élément à son parent
  list.appendChild(newItem);
</script>
```

Modifier des éléments dans le DOM

- Il existe plusieurs méthodes pour créer un nouvel objet:
  - document.createElement(tagname)
  - document.createTextNode(string)
  - document.createAttribute(name, value)
  - document.createDocumentFragment()

- Après avoir créé un objet, il est nécessaire de l'ajouter au DOM:
  - appendChild(newNode)
  - insertBefore(newNode, existingNode)
  - replaceChild(newNode, existingNode)
  - replaceData(offset, length, string)

Modifier des éléments dans le DOM

- Pour supprimer un élément ou un attribut:
  - Trouvez l'élément parent.
  - Utilisez removeChild ou removeAttribute pour supprimer la donnée.

```
// La cible, ma liste
var liste = document.getElementById("VenueList");
// Le premier element de la liste
var elm = liste.firstChild; // ou liste.firstElementChild
// removeChild(node)
liste.removeChild(elm);

//removeAttribute(attributeName)
liste.removeAttribute("id");

//removeAttributeNode(node)
liste.removeAttribute(liste.attributes[0]);
```

```
<ul id="VenueList">
  <li>Room A</li>
  <li>Room B</li>
</ul>
```

Manipulation d'objets DOM (1/2)

- document.createTextNode
  - Créé un nœud texte à insérer dans un élément HTML
- document.createElement
  - Créé un objet DOM sans l'ajouter dans le HTML
  - Exemple : document.createElement('div') créé une div
- element.appendChild
  - Ajoute un élément en tant que fils d'un autre élément

Manipulation d'objets DOM (2/2)

- element.createAttribute
  - Créé un attribut sur un élément
  - Exemple :

```
var align = document.createAttribute('align');
align.nodeValue = "right";
var element = document.getElementById('identifier');
element.setAttributeNode(align);
```

Modification du style

- Sur tout objet DOM il est possible de modifier le style à l'aide de sa propriété « style »

- La notation des propriétés se fait en Camel Case
  - CSS : background-color → JS: objet.style.backgroundColor

- Exemple :

```
var element = document.getElementById('identifier');
element.style.border = '1px solid red';
element.style.borderColor = 'red';
element.style.backgroundColor = 'red';
element.style.fontWeight = 'bold';
element.className = 'someClass';
```

TP06

- Créez une page web avec un formulaire qui possède un champ de type *text* et un *button* submit
  - L'action du formulaire ira sur « # »
  - Lors du clic sur le bouton
    - Vérifiez que le champ *text* n'est pas vide (utilisez la méthode trim() des String)
      - Supprimez le formulaire de la page et affichez « bonjour » + le contenu du champ *text*

- Question : peut-on se passer du formulaire ?

Gérer les évènements dans le DOM

- Le DOM définit les événements qui peuvent être déclenchées par le navigateur ou par l'utilisateur:

```
var helpIcon = document.getElementById("helpIcon");
helpIcon.onmouseover =
  function() {
      window.alert('Some help text');
  };
```

- Vous pouvez également définir des écouteurs d'événements qui s'exécutent lorsqu'un événement se déclenche:
  - Utile si le même événement a besoin de déclencher des actions multiples

Gérer les évènements dans le DOM

- Pour ajouter un écouteur d'événements:

```
var showHelpText =  function() {
      window.alert('Some help text');
}
helpIcon.addEventListener("mouseover", showHelpText, false);
```

- Pour supprimer un écouteur d'événements:

```
helpIcon.removeEventListener("mouseover", showHelpText, false);
```

TP07

- Prenez le site web de l'énoncé, ajoutez des validations pour les formulaires des pages
  - Login
    - Les deux champs doivent être non vides
  - Historique des opérations
    - La date de début doit être avant la date de fin
  - Virement
    - Les deux comptes sélectionnés doivent être différents
    - Le montant doit être un chiffre > 0

Bibliographie

- Le mode strict : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Strict_mode
- Configuration de JSHint : http://jshint.com/docs/
- Tout sur JS: https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/
- ECMAScript 2015 (ES6) : http://www.ecma-international.org/ecma-262/6.0/
- Document : https://developer.mozilla.org/fr/docs/Web/API/Document