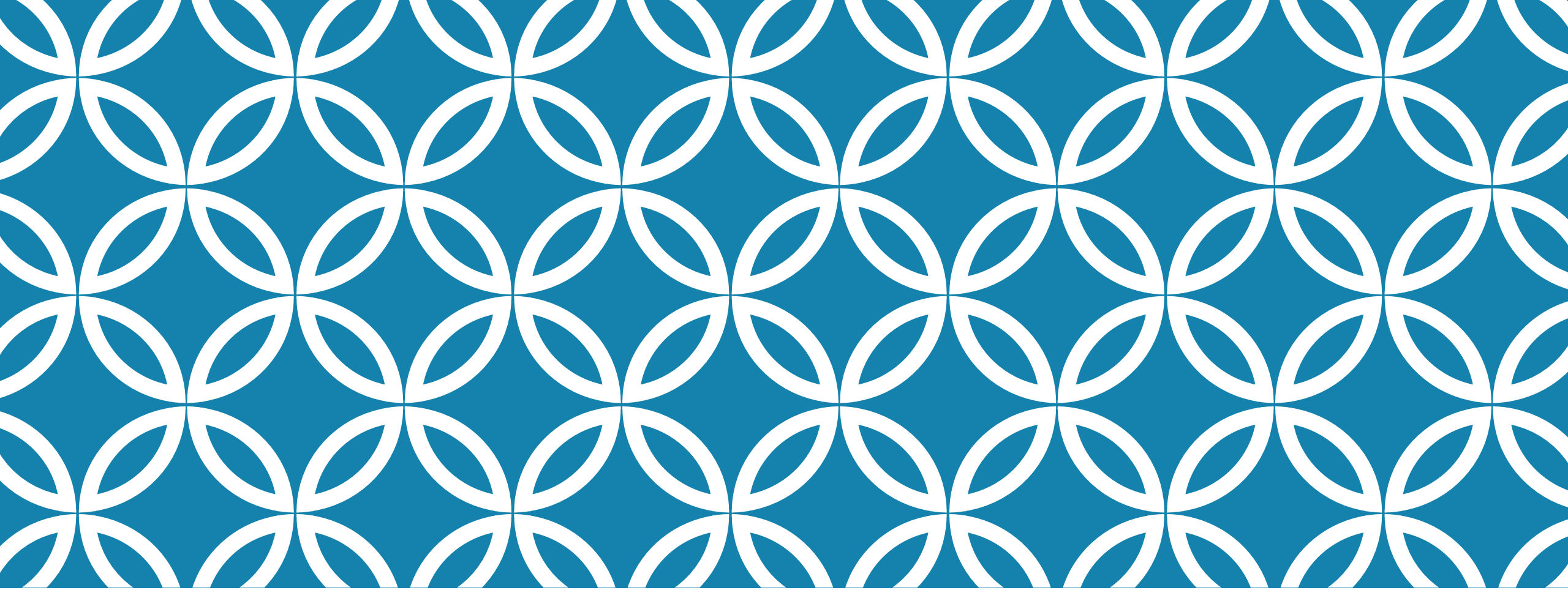


INTRODUCTION AU WEB
JAVASCRIPT

Par Jean-Pierre Lozi
Basé sur les cours d'Andrea Tettamanzi et Philippe Renevier

JAVASCRIPT: C'EST QUOI ?

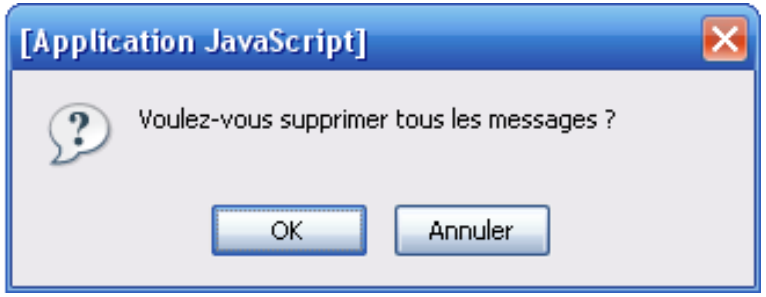
- Langage de script pour les pages web interactives
- Côté client : les scripts sont exécutés par le navigateur de l'utilisateur
- HTML pour le contenu, CSS pour la présentation, JavaScript pour programmer…
  - **…côté client seulement !**
  - …c'est-à-dire sur la page web récupérée par l'utilisateur (mais toujous même page récupérée)
- **Que peut-on faire en Javascript ?**
  - Puissance d'un langage de programmation complet, notamment structures conditionnelles (if/then/else), et boucles pour les répétitions (for, while).
  - Possibilité de changer le contenu de la page (modifier le HTML), les styles (modifier le CSS), d'afficher des boîtes de dialogue (messages d'alerte), de récupérer les événements clavier et souris (et réagir en conséquence), de dessiner…

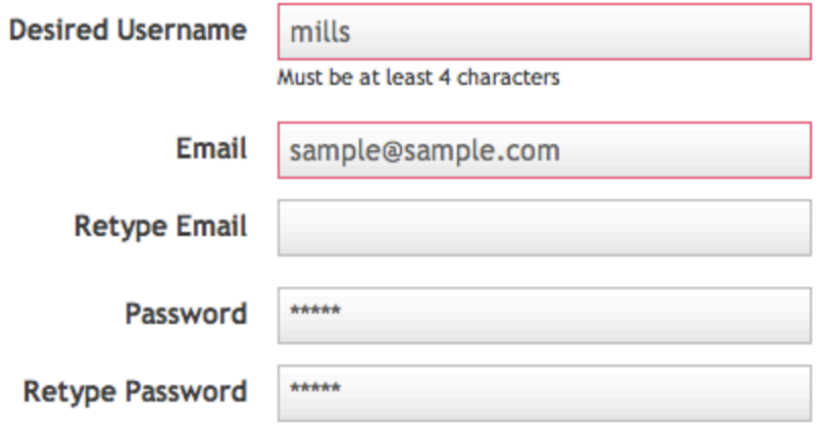
JAVASCRIPT: C'EST QUOI ?

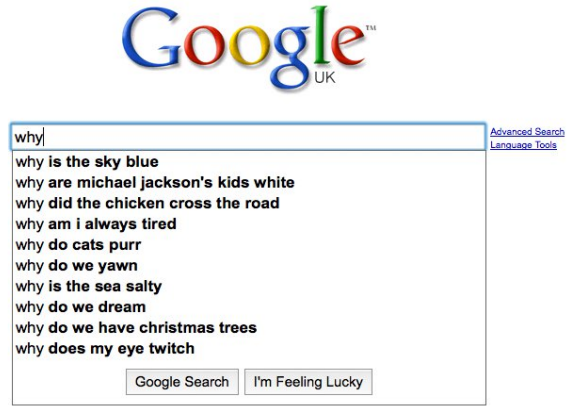
- **Différence avec la programmation côté serveur (PHP par exemple) ?**
  - Ne peut pas délivrer du contenu différent selon ce qu'on lui demande
  - Par exemple, une recherche Google : résultats différents envoyés selon votre requête. Un programme tourne sur le serveur, récupère votre requête, trouve les résultats, et vous envoie une page avec un contenu différent selon ce que vous avez demandé.
    Impossible en JavaScript, où on a toujours la même chose sur la page, et le programme effectue des actions locales selon vos intéractions avec la page (clavier / souris / boutons…)
  - Autre exemple : session sur votre banque en ligne, côté serveur ! Le serveur vous identifie et selon qui vous êtes, envoie des pages différentes, avec les infos vous correspondant. Impossible en JavaScript !
- « En gros », la différence entre les deux :
  - **Programmation côté client :** pour les actions sur une page qui ne demandent pas de recharger la page (un jeu sur la page par exemple, un menu animé, une carte interactive un peu complexe…)
  - **Programmation côté serveur :** pour les actions qui nécessitent de charger une nouvelle page générée dynamiquement (par exemple, récupérer et interpréter les résultats d'un formulaire, afficher une page de résultats de recherche Google, afficher les messages d'un fil sur un forum, vos opérations bancaires…)

JAVASCRIPT: C'EST QUOI ?

- Quelques exemples courants d'utilisation de JavaScript :
  - Afficher des boîtes de dialogue pour confirmer une action (un peu passé de mode)
  - Vérifier en temps réel (sans soumettre) que les informations sont valides lorsqu'on remplit les champs d'un formulaire (le mot de passe a le bon nombre de caractères, téléphone = que des chiffres…)
  - Suggestions en temps réel dans votre moteur de recherche comme le fait Google !
  - Objets dynamiques un peu complexes qui répondent aux mouvements de souris (ou frappes clavier) : un menu déroulant dynamique, un graphique qui montre l'évolution de quelque chose en temps réel (cours de la bourse par exemple), etc.
  - **Vu qu'on peut dessiner et récupérer éléments clavier et souris, on peut programmer des jeux !**



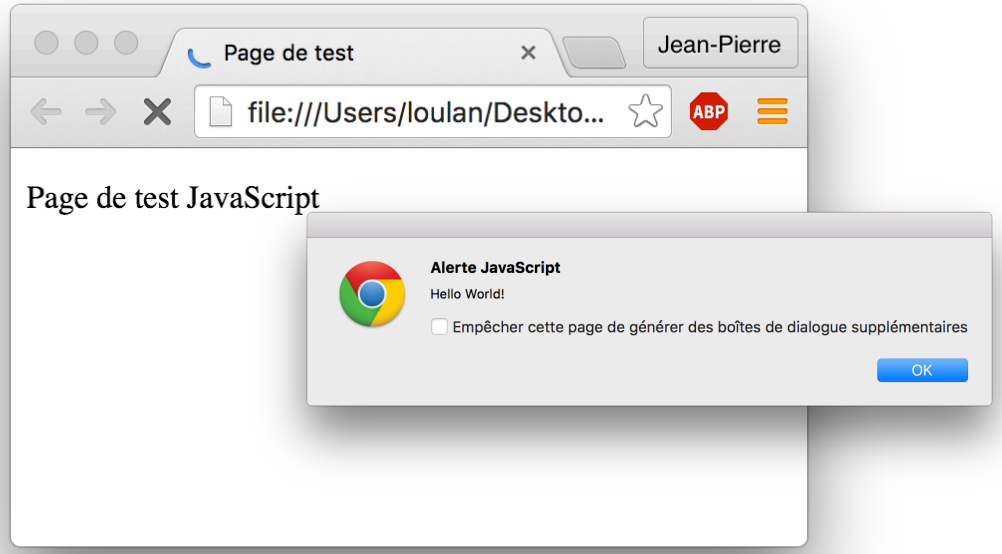




JAVASCRIPT: À QUOI ÇA RESSEMBLE ?

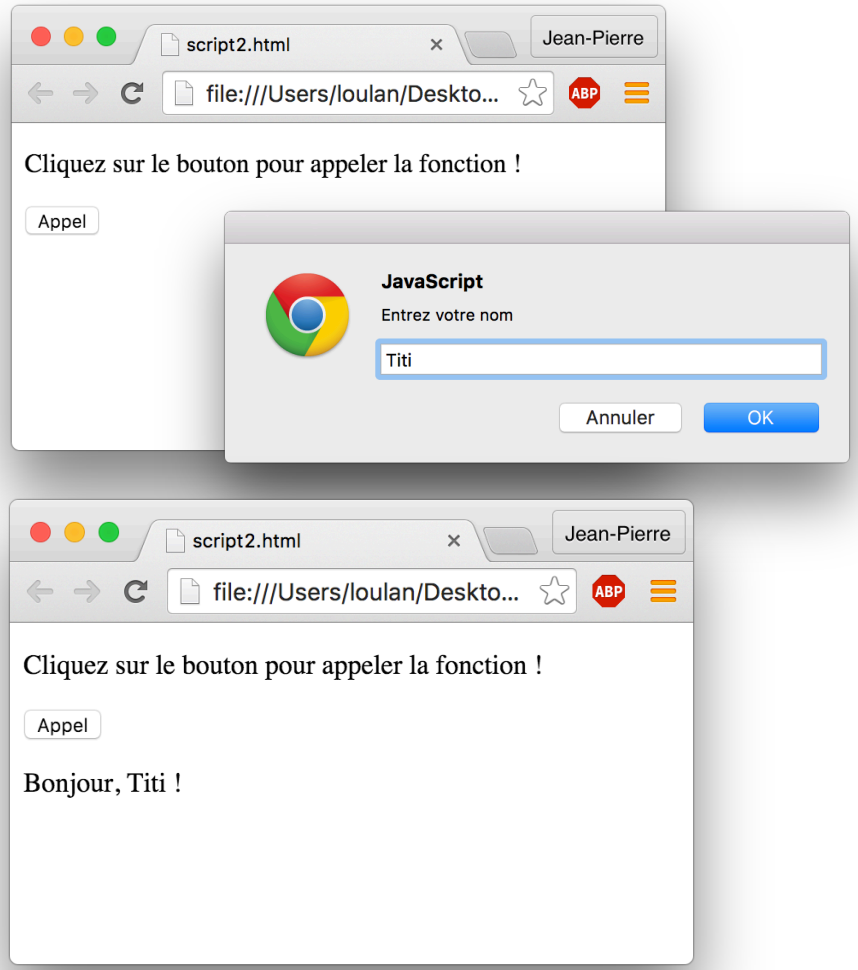
```
<!DOCTYPE html>
<html>
  <head>
    <title>Page de test</title>
  </head>
  <body>
    <p>Page de test JavaScript</p>
    <script type="text/javascript">
      alert("Hello World!");
    </script>
  </body>
</html>
```

Optionnel en HTML 5



JAVASCRIPT: À QUOI ÇA RESSEMBLE ?

```
<!DOCTYPE html>
<html>
<body>
<p>Cliquez sur le bouton pour appeler la fonction !</p>
<button onclick="maFonction()">Appel</button>
<p id="demo"></p>
<script>
function maFonction()
{
  var personne;
  personne = prompt("Entrez votre nom", "Toto");

  if (personne != null)
  {
    document.getElementById("demo").innerHTML =
      "Bonjour, " + personne + " !";
  }
}
</script>
</body>
</html>
```



INSÉRER DU JAVASCRIPT DANS UNE PAGE WEB

- Les script sont dans les balises `<script>...</script>`
- Avant HTML 5, il fallait utiliser `<script type="text/javascript">...</script>`. Paramètre `type` plus nécessaire maintenant, vous pouvez l'omettre.
- Possible de mettre des balises script multiples, dans le `header` ou le `body` (sans importance)
- Au lieu d'ajouter le code directement dans la page, possible de le mettre dans un fichier séparé (généralement avec l'extension `.js`) et de l'inclure avec:
  `<script src="chemin/vers/fichier.js"></script>`
- Pour éditer vos scripts : n'importe quel éditeur de texte, comme pour du HTML ou du CSS
  - Gedit, Emacs, etc.

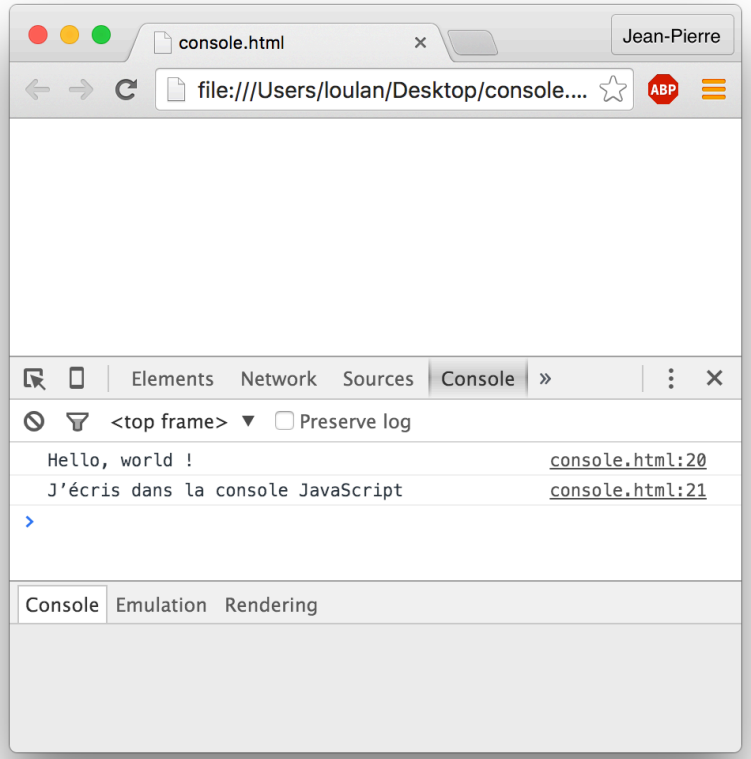
JAVASCRIPT : ÉLÉMENTS DE BASE

- Toutes les instructions doivent être séparées par des points-virgule (semicolon en Anglais) :

```
console.log("Hello, world !");
console.log("J'écris dans la console JavaScript");
alert("Voici une alerte !");
```

- `console.log()` permet d'écrire dans la console JavaScript, pratique pour déboguer vos programmes (afficher la valeur d'une variable, ou voir si on est entré dans un `if`, par exemple)

- Pour afficher la console JavaScript : sous Chrome/Chromium : `Menu Affichage > Options pour les développeurs > Console JavaScript`, sous Firefox : `Menu Outils > Développement web > Console web...`



JAVASCRIPT : ÉLÉMENTS DE BASE

- **Commentaires** dans le code : soit entre /* et */ (lignes multiples possibles), comme en CSS
- Soit après // jusqu'à un retour à la ligne
- Rappel : tout ce qui est entre commentaire ignoré lors de l'exécution du code. Utile pour documenter le code, ou pour en « enlever » temporairement une partie quand vous travaillez dessus
- Même chose qu'en PHP, C++ ou Java par exemple…
- Exemple :

```
/* Je peux revenir
à la ligne facilement
comme ça pour les longs commentaires */
console.log("Hello World!"); // Commentaire court
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Les **variables** : permettent de stocker des valeurs. Rappellez-vous :

```
var personne;
personne = prompt("Entrez votre nom", "Toto");
```

- Lorsque vous créez une variable, elle n'a pas de valeur par défaut (`NULL`)
- On peut lui assigner un valeur avec = (affectation)
- On peut stocker des choses diverses dans une variable, comme des chaînes de caractère (`"Hello world !"`), des nombres, des collections d'éléments (tableaux…)
- La valeur d'une variable peut changer pendant l'exécution d'un programme.
- Nom d'une variable :
  - Majuscules et minuscules importantes ! (`person` et `Person` seront deux variables différentes...)
  - Lorsqu'on déclare une nouvelle variable elle doit avoir un nouveau nom
  - Un nom de variable doit commencer avec une lettre, `$`, ou `_`
  - Les noms de variable ne peuvent contenir que des lettres, des chiffres, `$`, ou `_`

JAVASCRIPT : ÉLÉMENTS DE BASE

- Pour déclarer (créer) une variable, on utilise var, par exemple :

```
var ageDuCapitaine;
```

- Généralement une bonne idée de donner à vos variables une valeur de départ, par ex. :

```
var ageDuCapitaine = 64;
```

- Une fois que vous avez une variable, vous pouvez directement l'utiliser avec son nom :

```
var ageDuCapitaine = 64;
console.log(ageDuCapitaine); // Affichera 64 dans la console
```

- Vous pouvez modifier la valeur de la variable après sa création :

```
var ageDuCapitaine = 64;
console.log(ageDuCapitaine); // Affichera 64 dans la console
ageDuCapitaine = 32;
console.log(ageDuCapitaine); // Affichera 32 dans la console
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Les variables peuvent contenir des entiers ou des flottants :

```
var nombreDePoissons = 5;
var ageExactDuCapitaine = 64.3;
```

- Une fois que vous avez des variables, vous pouvez calculer avec !

```
var litresDEau = 5;
var litresDEauAjoutes = 1.5;

litresDEau = litresDEau + litresDEauAjoutes;
/* litresDEau vaut maintenant 6.5 */
```

Entiers automatiquement convertis en flottants quand nécessaire !

JAVASCRIPT : ÉLÉMENTS DE BASE

- **Opérateurs arithmétiques** :

Exemple	Nom	Résultat
`-a`	Négation	Opposé de `a`
`a + b`	Addition	Somme de `a` et de `b`
`a - b`	Soustraction	Différence entre `a` et `b`
`a * b`	Multiplication	Produit de `a` et de `b`
`a / b`	Division	Quotient de `a` et `b`
`a % b`	Modulo	Reste de la division euclidienne de `a` par `b`

- Existent également `+=`, `-=`, `*=`, `/=`, `%=` qui combinent opération et affectation

```
litresDEau = litresDEau + litresDEauAjoutes;
/* Identique à : litresDEau += litresDEauAjoutes; */
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Les variables peuvent contenir des chaînes de caractères :

```
var nomDuCapitaine = "Nemo"; // Valide également :
var surnomDuCapitaine = 'Ahab';
```

- Si vous voulez utiliser des guillemets dans une chaîne de caractère définie avec des guillemets, ou des apostrophes dans une chaîne de caractères définie avec des apostrophes, il vous faudra les échapper avec un backslash (\) :

```
var message1 = "Le nom d'usage du capitaine est \"Ahab\".";
var message2 = 'Le nom d\'usage du capitaine est "Ahab".';
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Vous pouvez **concaténer** des chaînes de caractères avec +
- « Concaténer » veut seulement dire créer une nouvelle chaîne qui est formées de plusieurs chaînes accolées entre elles (terme courant en informatique, souvenez-vous-en !)
- Par exemple :

```
var prenom = "Robert";
var nomComplet = nomDeFamille + " Nemo";
document.write(nomComplet); // Ecrit dans la page "Robert Nemo"
```

- += marche aussi avec les chaînes de caractères, même principe (sauf qu'ici concatène avec soi-même au lieu d'ajouter à soi-même) :

```
var nom = "Robert";
nom += " Nemo"; // Equivalent à nom = nom + " Nemo";
document.write(nom); // Ecrit dans la page "Robert Nemo"
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Possible d'utiliser des **fonctions** : des morceaux de code réutilisable
- Les fonctions peuvent prendre des **arguments (ou paramètres)**, et renvoyer une **valeur de retour**
- On en a déjà vu certaines offertes par JavaScript : `console.log("Mon message")` (un argument), ou encore `prompt("Entrez votre nom", "Toto")` (deux arguments, une valeur de retour)
- On a vu comment écrire une fonction sans paramètre dans l'un des deux exemples. Rappel : avec le mot-clé `function`. On peut écrire par exemple :

```
function maFonction() {
console.log("Fonction appellée"); }
```

- On peut ensuite appeler la fonction autant de fois qu'on veut avec :

```
maFonction();
```

- On peut aussi faire en sorte que la fonction soit appellée lorsqu'on clique sur un élément HTML en y ajoutant le paramètre `onclick="maFonction()"`

JAVASCRIPT : ÉLÉMENTS DE BASE

- Exemple de définitions de fonctions avec des arguments :

```
function appelleMarin(nom) {
    console.log("Ohé, " + nom +  " !");
}

appelleMarin("Némo"); // Ecrit "Ohé, Némo !"


function someCarree(a, b) {
    console.log((a + b) * (a + b));
}

someCarree(1, 3); // affiche 16
someCarree(5, 7); // affiche 144
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Vous pouvez passer des variables à des fonctions. Les variables n'ont pas besoin d'avoir le même nom que les arguments de la fonction :

```
function ajouteUn(nombre){
    var nouveauNombre = nombre + 1;
    console.log("On a : " + nouveauNombre);
}

// On déclare des variables
var nombreDePoissons = 8;
var nombreDeTortues = 4;

// On les utilise dans des fonctions
ajouteUn(nombreDePoissons); // Affiche "On a : 9"
ajouteUn(nombreDeTortues); // Affiche "On a : 5"
```

JAVASCRIPT : ÉLÉMENTS DE BASE

- Une fonction peut également renvoyer une valeur :

```
function carre(nombre) {
    return nombre * nombre;
}

var carreDeCinq = carre(5);

console.log(carreDeCinq); // Ecrit "25" dans la console
console.log(carre(4)); // Ecrit "16" dans la console
```

- Attention, `return` sort tout de suite de la fonction !

```
function carre(nombre) {
    return 0;
    console.log(nombre * nombre); // Ne sera jamais exécuté
}
```

JAVASCRIPT : ÉLÉMENTS DE BASE

Petit résumé des fonctions qu'on a vues jusqu'à présent :

- `console.log(chaine)` : affiche chaine dans la console JavaScript
- `document.write(chaine)` : écrit la chaine dans la page
  - Là où se trouve le script si exécuté avant que page entièrement chargée
  - **Remplace le contenu de la page par la chaîne sinon. À éviter !**
- `document.getElementById(un_id)` : renvoie l'élément dans la page HTML dont l'identifiant est `un_id`. Si on a stocké l'élément dans une variable element, on peut ensuite modifier son contenu avec `element.innerHTML = …;` (utile pour écrire dans une balise quelconque qui a un identifiant dans la page)
- `alert(chaine)` : affiche une boîte de dialogue avec la chaîne (et un bouton « OK »)
- `prompt(chaine, par_defaut)` : demande une valeur avec le message chaine, la valeur par défaut de la chaine sera `par_defaut`. Renvoie la valeur qu'a entré l'utilisateur, ou `NULL` s'il a cliqué sur « Annuler »

INTRO À JAVASCRIPT : TOUT COMPRIS ?

- Dans le script ci-dessous, trouvez commentaire(s), variable(s), opérateur(s), fonction(s), argument(s), et valeur(s) de retour :

```
function calculePourboire(total) {
    var pourcentagePourboire = 0.12; // Peut être modifié
    return (total * pourcentagePourboire);
}

var sommeDueAvantPourboire = parseInt(prompt("Somme due ?", "0"));

var pourboire = calculePourboire(sommeDueAvantPourboire);
var recu = 'Repas: ' + sommeDueAvantPourboire
           + ' Pourboire: ' + pourboire
           + ' Total: ' + (sommeDueAvantPourboire + pourboire);

alert(recu);
```

- **Que fait le script ?**

PORTÉE DES VARIABLES

- Portée d'une variable : « d'où à où » l'ordinateur s'en souvient dans le code
- Une variable déclarée hors d'une fonction a une **portée globale** et on peut y accéder de partout, y compris dans des fonctions :

```
var langage = "JavaScript"; // Portée globale

function jAdore() {
    console.log ("J'adore " + langage + "."); // Fonctionnera
}

jAdore();
```

- On parle tout simplement d'une **variable globale** (concept courant dans la plupart des langages de programmation)

PORTÉE DES VARIABLES

- Une variable déclarée dans une fonction a une **portée locale** et on ne peut y accéder que de l'intérieur d'une fonction :

```
function jAdore() {
    var langage= "JavaScript"; // Portée locale
    console.log ("J'adore " + langage + "."); // Fonctionnera
}
jAdore();

console.log ("J'adore " + langage + "."); // Ne fonctionnera pas
```

- On parle tout simplement d'une **variable locale** (également un concept courant)

BOOLÉENS

- On a vu que les variables pouvaient contenir des entiers, des nombres flottants, et des chaînes de caractères… elles peuvent aussi contenir une **valeur booléenne** (i.e., vrai/`true` ou faux/`false`)
- Par exemple, on peut définir :

```
var javaScriptCEstBien = true;
var basicCEstBien = false;
```

- Si on essaie d'utiliser une variable qui n'est pas un booléen en tant que booléen, JavaScript essaiera de deviner ce qu'on veut dire (conversion automatique)
- **Le nombre 0, la chaîne vide "", et NULL sont considérés comme étant false, tout le reste est considéré comme étant true**

BOOLÉENS

- Utilité des booléens ? Par exemple pour décider si un morceau de code doit être exécuté ou pas selon une condition… On a déjà vu dans un exemple **la structure conditionnelle if** :

```
if (condition) {
    // Instructions à exécuter si condition vaut true
}
```

- Par exemple :

```
var bananes = 5;
if (bananes > 0) {
    console.log ("Vous avez des bananes.");
}
```

- Ou encore :

```
var bananes = 5;
var condition = (bananes > 0);
if (condition) {
    console.log ("Vous avez des bananes.");
}
```

BOOLÉENS

- Opérateurs qui renvoient des variables booléennes :

Exemple	Nom	Résultat
`a == b`	Egal	Renvoie true ssi `a` est égal à `b` (peuvent être de types différents, e.g. `"0"` et `0`)
`a === b`	Identique	Renvoie true ssi `a` est identique à `b` (même valeur et même type)
`a != b`	Différent	« Contraire » de `==`
`a !== b`	Non identique	« Contraire » de `!==`
`a < b`	Strictement inférieur à	`true` si `a` est strictement inférieur à `b`
`a > b`	Strictement supérieur à	`true` si `a` est strictement supérieur à `b`
`a <= b`	Inférieur ou égal à	`true` si `a` est inférieur ou égal à `b`
`a >= b`	Supérieur ou égal à	`true` si `a` est supérieur ou égal à `b`

BOOLÉENS

ATTENTION : NE PAS CONFONDRE = et == !

(une erreur classique !)

LA STRUCTURE CONDITIONNELLE IF

- On a vu `if` pour exécuter du code si une condition est vérifiée. On peut ajouter un « sinon », appellé `else`, pour exécuter du code différent si la condition n'est pas vérifiée.

```
var age = 25;

if (age >= 18) {
    console.log ("Whoo, vous pouvez conduire !");
}
else {
    console.log ("Désolé, vous pourrez seulement conduire dans "
                 + (18 - age) + "ans.");
}
```

LA STRUCTURE CONDITIONNELLE IF

- Pour des conditions multiples, possible d'utiliser `else if` :

```
var age = 20;

if (age >= 24) {
    console.log("Vous pouvez voter et devenir sénateur !");
}
else if (age >= 18) {
    console.log("Vous pouvez voter.");
}
else if (age >= 16) {
    console.log("Vous ne pouvez pas voter mais vous pouvez"
                + "travailler sous certaines conditions.");
}
else {
    console.log("Vous pouvez... aller à l'école ?");
}
```

LA STRUCTURE CONDITIONNELLE IF

- Possible de supprimer les accolades lorsqu'on n'a qu'une seule instruction :

```
var age = 20;

if (age >= 24)
    console.log("Vous pouvez voter et devenir sénateur !");
else if (age >= 18) {
    console.log("Vous pouvez voter.");
    console.log("Félicitations !");
}
else if (age >= 16)
    console.log("Vous ne pouvez pas voter mais vous pouvez"
                + "travailler sous certaines conditions.");
else
    console.log("Vous pouvez... aller à l'école ?");
```

LA STRUCTURE CONDITIONNELLE IF

Opérateurs logiques sur des booléens :

- **Opérateur ET : `a && b`**
  `true` si `a` et `b` tous les deux vrais, `false` sinon. Si `a` est `false`, JavaScript n'essaie même pas d'évaluer `b`, car il sait que l'expression ne pourra être vraie de toute façon : renvoie directement `false`.
- **Opérateur OU : `a || b`**
  `true` si soit `a` soit `b` vrai, `false` sinon. Si `a` est `true`, JavaScript n'essaie même pas d'évaluer `b`, car il sait il sait que l'expression ne pourra être fausse de toute façon : renvoie directement `true`.
- **Opérateur NON : `!a`**
  Renvoie `true` si `a` vaut `false` et inversement.

LES BOUCLES

- Boucles utilisées pour répéter du code plusieurs fois
- Utile pour ne pas réécrire la même chose de nombreuses fois, et souvent nécessaire car le nombre de fois qu'on veut répéter le code peut dépendre d'un paramètre dont la valeur ne peut être prédite
- **Boucle `while` (« tant que ») : répète le code tant que la condition entre parenthèses est vraie**

```
var bananes = 10;
while (bananes >= 1) {
    console.log ("J'ai " + bananes + " bananes et j'en mange une...");
    bananes--;
}
console.log ("Je n'ai plus de bananes.");
```

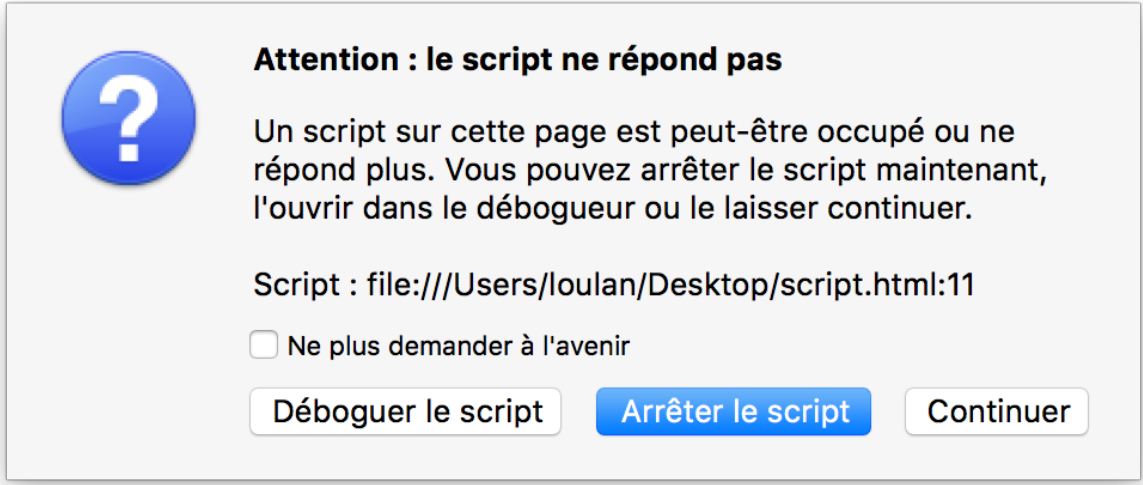
- Au passage, `bananes--` décrémente (= soustrait 1 à) la variable bananes, équivalent à `bananes = bananes - 1` ou `bananes -= 1`. De la même manière, `bananes++` incrémente (ajoute 1 à) la variable `bananes`.

LES BOUCLES

- À votre avis, quel est le problème de ce script ?

```
var bananes = 10;

while (bananes >= 1) {
    console.log ("J'ai " + bananes
                 + " bananes et j'en mange une...")
}
```

LES BOUCLES



- À votre avis, quel est le problème de ce script ?

```
var bananes = 10;

while (bananes >= 1) {
    console.log ("J'ai " + bananes
                 + " bananes et j'en mange une...")
}
```

- **Boucle infinie !** Il faut toujours au moins que quelque chose « change » dans votre boucle pour qu'elle ait une chance de se terminer…
- C'est nécessaire mais pas suffisant ! Par exemple, si vous incrémentez la variable bananes au lieu de la décrémenter, vous « changerez » quelque chose, mais quand même une boucle infinie…
- Page qui ne répond pas, peut-être un message d'erreur au bout d'un moment…

LES BOUCLES

- **Boucle `for` (« pour ») : généralement, répète le code pour un nombre de fois donné, e.g.**

```
for (var i = 0 ; i < 100 ; i++)
    console.log (i);

console.log ("J'ai compté de 0 à 99 !");
```

- Une boucle for a 3 expressions séparées par des points-virgule :

```
for (à exécuter au départ ;
     test : on reste dans la boucle ? (effectué aussi avant d'entrer) ;
     à exécuter à la fin de chaque itération)
```

- On peut mettre plusieurs instructions séparées par des virgules pour chaque expressions… Du coup on peut faire des choses plus compliquées :

```
for (var i = 0, j = 0 ; i + j < 100 ; i++, j += 2)
    document.write(i + " " + j + "<br />");
```

LES BOUCLES

- **Boucle `for` (« pour ») : généralement, répète le code pour un nombre de fois donné, e.g. :**

```
for (var i = 0 ; i < 100 ; i++)
    console.log (i);

console.log ("J'ai compté de 0 à 99 !");
```

Au passage, remarquez que pour les `for` aussi on peut supprimer les accolades si on n'a qu'une seule instruction ! Marche pour les `if`, else `if`, `else`, `while`, `for`… à peu près pour n'importe quel bloc sauf les fonctions !

LES BOUCLES

- Vous pouvez bien sûr ajouter des instructions et utiliser des opérateurs logiques dans une boucle, par exemple :

```
for (var i = 1; i <= 50; i++) {

    if (i % 3 == 0 && i % 5 == 0)
        console.log ("FizzBuzz");
    else if (i % 3 == 0)
        console.log ("Fizz");
    else if (i % 5 == 0)
        console.log ("Buzz");
    else
        console.log (i);
}
```

- **D'ailleurs que fait ce code ?** (c'est le fameux exercice du « Fizz Buzz » utilisé dans les entretiens d'embauche…)

BREAK, CONTINUE, RETURN

- `break` sort d'une boucle prématurément :

```
function premier_multiple_de_7(debut, fin) {
    for (var i = debut; i <= fin; i++) {
        console.log("Je teste " + i); // Arrêt au premier multiple de 7
        if (i % 7 == 0) {
            console.log("Trouvé ! " + i);
            break; // Sort de la boucle
        }
    }
}
```

- Attention, si vous avez une boucle dans une autre, **`break` ne sort que de la boucle intérieure, pas de la boucle extérieure !**

BREAK, CONTINUE, RETURN

- `continue` passe à l'itération suivante :

```
function nombres_non_multiples_de_7(debut, fin) {
    for (var i = debut; i <= fin; i++) {
        if (i % 7 == 0) continue;
        text += i + "n'est pas divisible par 7.<br />";
    }
    return text;
}
```

- On a vu que `return` renvoie une valeur de retour dans une fonction et n'exécute pas le code ensuite. On peut en fait utiliser return dans n'importe quelle fonction pour en sortir immédiatement. Si la fonction n'a pas de valeur de retour on utilise simplement « `return;` », sans spécifier de valeur.
  - **Permet notamment de sortir de plusieurs boucles imbriquées !**

BREAK, CONTINUE, RETURN

- Exemple : sortir de plusieurs boucles avec `return`

```
function trouve_et_ecris_deux_facteurs (x) {
    for (var i = 0; i <= x; ++i) {
        for (var j = 0; j <= x; ++j) {
            if(i * j == x) {
                document.write(i + " x " +
                               j + " =" + "x<br />");
                return;
            }
        }
    }
}
```

- **Que fait ce code ?**

BOUCLES : UN EXEMPLE D'ALGORITHME SIMPLE…

- Afficher tous les nombres premiers entre 0 et 100 :

```
function estPremier(num) {
    if (num < 2) return false;
    for (var i = 2; i < num; i++) {
        if(num % i == 0)
            return false;
    }
    return true;
}

for(var i = 0; i < 100; i++) {
    if(estPremier(i))
        console.log(i);
}
```

TABLEAUX

- Les tableaux sont des listes ordonnées de valeurs. Par exemple :

```
var tableau = [element0, element1, ...];
```

Attention: cela n'a rien à voir avec des tableaux HTML, il s'agit de listes de valeurs qu'on manipule en programmant, pas quelque chose de graphique !

- Vous pouvez mettre n'importe quel type de valeur dans un tableau, et des cases différentes d'un tableau peuvent avoir des valeurs différentes :

```
var couleursArcEnCiel = ["Rouge", "Orange", "Jaune", "Vert", "Bleu",
                         "Indigo", "Violet"];
var numerosDuLoto = [33, 2, 34, 18, 17, 45];
var nomAnneeNaissance = ["Antoine Martin", 1984];
```

- Possible de connaître le nombre d'éléments d'un tableau avec la propriété `length` :

```
console.log(couleursArcEnCiel.length); // Ecrit 7
```

TABLEAUX

- On peut accéder à une case d'un tableau en mettant son indice entre crochets. Par exemple, pour modifier un élément :

```
nomAnneeNaissance[1] = 1985;
```

- Pour lire un élément :

```
console.log(nomAnneeNaissance[1]);
```

- Les tableaux n'ont pas une taille fixe. Vous pouvez utiliser `push()` pour ajouter un élément à la fin d'un tableau :

```
var langages = ["Java", "Javascript", "C++"];
langages.push("Fortran");
/* Maintenant, langages contient ["Java", "Javascript", "C++", "Fortran"]. */
```

- Pour récupérer le dernier élément d'un tableau et l'enlever : fonction `pop()` qui fait l'inverse

```
l = langages.pop();
/* l vaut "Fortran" et langages contient ["Java", "Javascript", "C++"]. */
```

TABLEAUX : UN EXEMPLE D'ALGORITHME SIMPLE…

- Tableaux en deux dimensions possible :

```
var matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

console.log(matrix[1][1]) // Affiche l'élément central
```

- Et de la même manière, en n dimensions :

```
var 3DMatrix = [
    [ [1, 2],
      [3, 4] ],
    [ [5, 6],
      [7, 8] ]
];
```

TABLEAUX : UN EXEMPLE D'ALGORITHME SIMPLE…

- Calculer le produit scalaire de deux vecteurs :

```
function produit_scalaire(a, b) {
    var n = 0;

    if (a.length != b.length)
        return "Erreur : vecteurs de taille différente.";

    for (var i = 0; i < a.length; i++)
        n += a[i] * b[i];

    return n;
}

console.log(produit_scalaire([1,2,3,4,5], [6,7,8,9,10]));
```

OBJETS

- Les objets permettent de stocker un ensemble de valeurs :

```
var objectName = {
    propertyName: propertyValue,
    propertyName: propertyValue,
    …
};
```

- Par exemple :

```
var infosUtilisateur = {
    villeNaissance: "Trifouillis-les-Oies",
    cheveux: "Bruns",
    aime: ["tricoter", "coder"], // Un tableau
    anniversaire: {jour: 10, mois: 8} // Un objet dans un objet
};
```

OBJETS

- Pour récupérer/modifier/ajouter une propriété : notation en point

```
var mesInfos = {
    villeNaissance: "Trifouillis-les-Oies",
    cheveux: "Bruns"
};

var maVilleNaissance = mesInfos.villeNaissance;
mesInfos.villeNaissance = "Antibes";
mesInfos.age = 50;
```

- Possible également d'utiliser la notation avec crochets (comme pour les tableaux)

```
var mesCheveux = mesInfos["cheveux"];
mesInfos["cheveux"] = "Blonds";
mesInfos["sexe"] = "M";
```

- Possible de supprimer des propriétés avec `delete` :

```
delete mesInfos.age;
delete mesInfos["sexe"];
```

OBJETS

- Puisque les tableaux peuvent contenir n'importe quel type d'élément, ils peuvent aussi contenir des objets… Par exemple :

```
var mesCours = [
    {intitule: "Topologie", heures: 4},
    {intitule: "Javascript", heures: 3.5}
];

for (var i = 0; i < mesCours.length; i++) {
    var monCours = mesCours[i];
    console.log("J'ai " + monCours.heures
                + " heures de " + monCours.intitule);
}
```

OBJETS

- Comme tout autre type de variable, un objet peut être passé en paramètre d'une fonction :

```
var winnie = {
    type: "Ours",
    age: 3,
    aime: ["miel", "jouer"],
    anniversaire: {jour: 7, mois: 6}
}

function decrireAnimal(animal) {
    console.log("Cet animal a " + animal.age
                + " ans. Il s'agit d'un " + animal.type + ".");
}

decrireAnimal(winnie);
```

OBJETS

- En plus des propriétés, les objets peuvent contenir des fonctions :

```
var marvinLeRobot = {
    age: 5,
    couleur: "Gris",
    bipe: function() {
        console.log("Bip");
    },
    marcheVers: function(direction) {
        console.log("Je marche direction " + direction+ ".");
    },
};

marvinLeRobot.bipe();
marvinLeRobot.marcheVers("ouest");
```

RAPPEL : L'ARBRE « DOM »

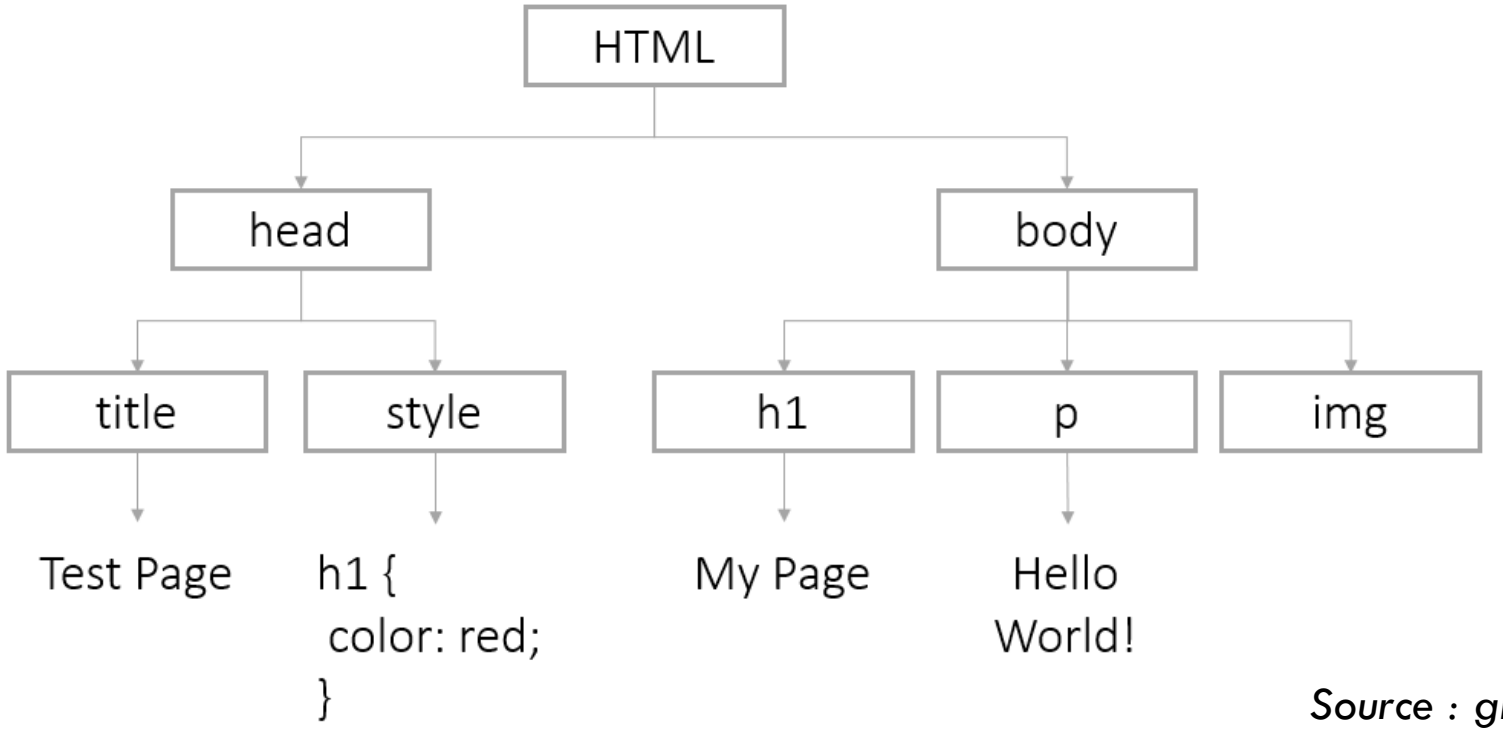
- Un document HTML est un « arbre » de balises, appellé « DOM » pour Document-Object Model. Par exemple :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Test page</title>
        <style>
            h1 { color: red; }
        </style>
    </head>
    <body>
        <h1>My Page</h1>
        <p>Hello World!</p>
        <img src="logo.jpg" alt="Logo" />
    </body>
</html>
```

Source : girldevelopit.github.io

RAPPEL : L'ARBRE « DOM »

- Un document HTML est un « arbre » de balises. Par exemple :



Source : girldevelopit.github.io

ACCÈS AUX ÉLÉMENTS DE LA PAGE

- Le navigateur crée automatiquement un objet « Document » pour stocker l'arbre DOM de la page. Pour modifier une page :
  - Récupérer l'arbre DOM et le stocker dans une variable
  - Utiliser des méthodes pour le manipuler
- On a déjà vu comment accéder à un élément par son identifiant (rappel : une seule balise avec un identifiant donné par page) :

```
document.getElementById(id);
```

- Par exemple, pour récupérer cette balise :

```
<img id="logo" src="logo.png" alt="Logo" />
```

- On peut utiliser :

```
var imgLogo = document.getElementById("logo");
```

ACCÈS AUX ÉLÉMENTS DE LA PAGE

- On peut également accéder aux éléments par type de balise :

```
document.getElementsByTagName(nomDeBalise);
```

- Les résultats seront stockés dans un tableau. Par exemple, si on a deux éléments `li` dans une page :

```
<ul>
    <li>Marguerite</li>
    <li>Tulipe</li>
</ul>
```

- On pourra y accéder de la manière suivante :

```
var listItems = document.getElementsByTagName("li");

for (var i =0; i < listItems.length; i++) {
    console.log("J'ai trouvé l'élément li : " + listItems[i]);
}
```

ACCÈS AUX ÉLÉMENTS DE LA PAGE

- De la même manière, possible de récupérer tous les éléments d'une classe donnée :

```
document.getElementsByClassName(nomDeClasse);
```

Bien sûr, les résultat sont stockés dans un tableau car il peut y en avoir plusieurs.

- On peut aussi récupérer le premier élément trouvé qui est identifié par un sélecteur CSS :

```
document.querySelector(selecteur);
```

- Ou, pour récupérer tous les éléments identifiés par un sélecteur CSS :

```
document.querySelectorAll(selecteur);
```

- Par exemple, pour changer la couleur de fond de toutes les cellules de classe important :

```
var importants = document.querySelectorAll("td.important");

for (var i = 0; i < importants.length; i++) {
    importants[i].style.backgroundColor = "yellow";
}
```

ACCÈS AUX ÉLÉMENTS DE LA PAGE

- Attention, petite subtilité : les scripts sont exécutés au fur et à mesure que la page est chargée
- **Si vous essayez de récupérer un élément déclaré plus tard dans la page cela ne fonctionnera pas si votre code est exécuté plus tôt dans la page !**
- Par contre, aucun problème si c'est du code exécuté lorsque l'utilisateur clique sur un bouton on répond à une action, par exemple, car ceci se produira une fois que la page est entièrement chargée
- Sinon, bien faire attention de mettre le script après l'élément auquel vous essayez d'accéder, ou de le mettre dans une fonction appelée par l'événement onload du body, qui signifie que la page a été entièrement chargée, par exemple :

```
<body onload="myFunction()">
```

MODIFICATION DES ATTRIBUTS

- On peut modifier les attributs d'un élément du DOM avec la notation en point. Par exemple, si on a l'image suivante :

```
<img id="animal" src="images/singe.png" alt="animal"/>
```

- On peut récupérer et stocker l'URL de l'image dans une variable, puis la modifier avec :

```
var photo = document.getElementById("animal");
var ancienneImage = photo.src;
photo.src = "images/serpent.png";
```

- Possible également d'utiliser `getAttribute()` et `setAttribute()` :

```
var photo = document.getElementById("animal");
var ancienneImage = photo.getAttribute("src");
photo.setAttribute("src", "images/serpent.png");
```

MODIFICATION DES STYLES CSS

- On peut modifier le style CSS d'un élément avec la propriété `style`.
- Par exemple, pour générer ce style CSS :

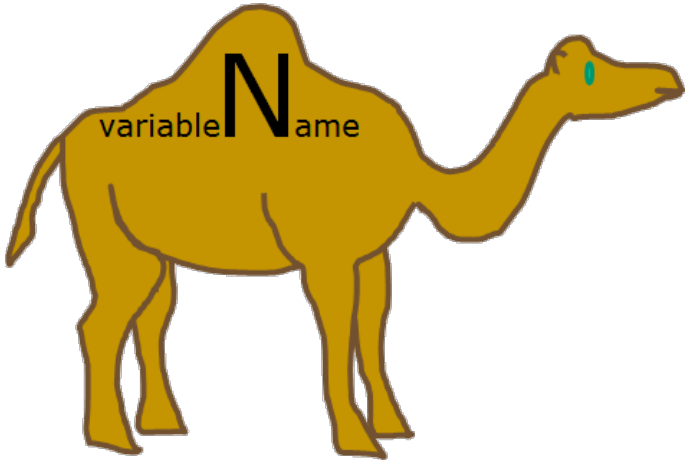
```
body {
    color: red;
}
```

- On pourra utiliser :

```
var pageNode = document.getElementsByTagName("body")[0];
pageNode.style.color = "red";
```

- **Pourquoi a-t-on besoin de [0] à la fin de la première ligne ?**

MODIFICATION DES STYLES CSS



- Attention : pour les propriétés CSS avec des tirets, il faut les transformer en camelCase en JavaScript, i.e., on enlève les tirets et chaque nouveau mot commence par une majuscule
- Par exemple, `background-color` devient `backgroundColor`, `border-right-color` devient `borderRightColor`...
- Par exemple, pour générer ce CSS :

```
body {
    background-color: pink;
    padding-top: 10px;
}
```

- On pourra utiliser:

```
var pageNode = document.getElementsByTagName("body")[0];
pageNode.style.backgroundColor = "pink";
pageNode.style.paddingTop = "10px";
```

MODIFICATION DU CONTENU DES BALISES

- On a déjà vu que pour modifier le HTML contenu dans une balise, on peut utiliser la propriété `innerHTML`. Par exemple :

```
var pageNode = document.getElementsByTagName('body')[0];
pageNode.innerHTML = "<h1>Wow !</h1> <p>J'ai remplacé la page"
                   + " en entier !</p>";
```

- On peut bien sûr ajouter du HTML à la fin de celui étant déjà présent dans la balise grâce à l'opérateur += :

```
pageNode.innerHTML += "...j'ajoute cette phrase en bas de la page";
```

- Bien sûr, comme on l'avait vu, cela fonctionne avec d'autres éléments que `body`, e.g. :

```
<p id="attention"></p>
```

- On peut faire :

```
var warningParagraph = document.getElementById("attention");
warningParagraph.innerHTML = "Danger !";
```

CRÉER DE NOUVEAUX ÉLÉMENTS

- On peut créer de nouveaux éléments dans la page (il s'agit de « nœuds » de l'arbre DOM)
- Ou y ajouter du texte (on peut avoir du texte hors balises dans le `body` par exemple, pas tout le texte est forcément dans du `h1`, `p`, etc.)
- Pour créer un nouvel élément, on peut utiliser :

```
maBalise = document.createElement(nomBalise);
```

- Puis, pour l'ajouter en (dernier) fils d'un élément existant :

```
elementExistant.appendChild(maBalise);
```

- On peut aussi ajouter du texte à la fin d'un élément avec :

```
elementExistant.appendText("Texte ajouté à la fin");
```

CRÉER DE NOUVEAUX ÉLÉMENTS

- Exemple : ajout d'une image et d'un paragraphe à la fin d'une page

```
var page = document.getElementsByTagName("body")[0];
var nouvelleImage = document.createElement("img");

nouvelleImage.src = "images/logo.png";
nouvelleImage.style.border = "1px solid black";

page.appendChild(nouvelleImage);

var nouveauParagraphe = document.createElement("p");
var texteDuParagraphe =
    document.createTextNode("Ceci est mon nouveau paragraphe.");
nouveauParagraphe.appendChild(texteDuParagraphe);

page.appendChild(nouveauParagraphe);
```

ÉLÉMENTS: CLASSES

- On peut accéder à la liste des classes d'un élément avec `.classList`.
- Le résultat est un élément de type `DOMTokenList`, qui a plusieurs opérations disponibles
- `.length` pour la longueur de la liste, et `.item(n)` pour accéder à un élément de la liste
- `.add("cl1", "cl2")` et `.remove("cl1", "cl2")` pour ajouter/retirer classes
- `.contains("class")` pour tester si la liste contient une classe
- `.toggle("class")` ajoute la classe si elle n'était pas dans la liste l'enlève sinon.

ÉVÉNEMENTS

- Comme on l'a vu, on peut spécifier une fonction qui est appelée lorsqu'un événement se produit sur une page. Un événement = un clic de la souris, la souris entre ou sort d'un élément, la souris se déplace, frappe clavier, etc.
- Par exemple, on peut écrire :

```
function maFonction() {
    // Fonction à exécuter lors d'un clic de la souris sur element
};

element.onclick = maFonction;
```

- Ou encore, tout simplement :

```
element.onclick = function () {
    // Fonction à exécuter lors d'un clic de la souris sur element
};
```

ÉVÉNEMENTS

De nombreux événements existent. Parmi les plus courants :

- **`onclick`** : l'utilisateur clique sur un élément
- **`onmouseenter`** : le pointeur de la souris entre dans un élément
- **`onmouseleave`** : le pointeur de la souris sort d'un élément
- **`onmousemove`** : le pointeur de la souris bouge (sur un élément)
- **`onkeydown`** : l'utilisateur appuie sur une touche (on a aussi **`onmousedown`**)
- **`onkeyup`** : l'utilisateur relâche une touche (on a aussi **`onmouseup`**)
- **`onkeypress`** : l'utilisateur appuie sur puis relâche une touche

ÉVÉNEMENTS

- Il y a de nombreux éléments autres que les éléments clavier/souris. Par exemple:
  - **`onwheel`** = utilisation de la molette de la souris
  - **`onresize`** = la fenêtre a été redimensionnée
  - **`onscroll`** = l'utilisateur fait défiler la page
  - **`oncut`**/**`oncopy`**/**`onpaste`** = l'utilisateur coupe/copie/colle le contenu de l'élément
  - **`animationend`** = une animation CSS est terminée
  - **`onoffline`** = lorsque le navigateur se met à travailler hors ligne
  - **`ontouchstart`** = l'utilisateur touche l'écran (d'un téléphone, d'une tablette)
- **Presque 100 événements au total !**

ÉVÉNEMENTS

- Comme on l'a déjà vu, on peut associer une fonction à un événement directement en HTML

```
<button onclick="disBonjour()">Cliquez moi !</button>
```

- Avec par exemple :

```
function disBonjour() { alert("Bonjour !"); }
```

- Les fonctions appellées par les événements peuvent prendre un paramètre, qui contient des informations sur l'événement. Par exemple :

```
function moved(event) {
    var posX = event.clientX;
    var posY = event.clientY;
    document.getElementById("divPosition").innerHTML =
        "Coordonnées de la souris :" + "X = " + posX + ", Y = " + posY;
}
/* On suppose qu'on a un div avec identifiant divPosition
   qui contiendra les coordonnées de la souris. */
```

LE MOT-CLÉ « THIS »

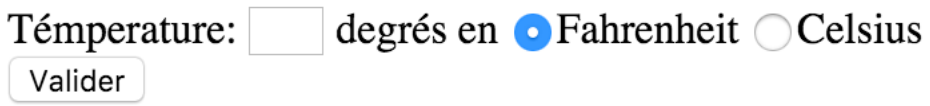
- Le mot-clé `this` fait référence à l'élément avec lequel l'utilisateur vient d'intéragir :

```
element.onmouseover = function() {
    console.log(this);
};

element.onclick = function() {
    this.style.backgroundColor = "red";
    this.innerHTML = "On m'a cliqué !";
};
```

FORMULAIRES

- On peut utiliser des formulaires HTML pour récupérer des informations des utilisateurs qu'on pourra utiliser dans des fonctions. Par exemple :

```
<form id="temperatureForm">
    <label for="temp">Témperature:</label>
    <input type="text" id="temp" size="3" /> degrés en
    <input type="radio" name="tempFormat" value="F" checked />Fahrenheit
    <input type="radio" name="tempFormat" value="C" />Celsius
    <br />
    <input id="tempSubmitButton" type="submit" value="Valider" />
</form>
```



FORMULAIRES

- On peut récupérer les données avec la propriété `value` :

```
var temperature = document.getElementById("temp").value;
console.log (temperature);
```

Peut être fait à n'importe quel moment, essayer par exemple `onblur` (lorsqu'un élément du formulaire n'est plus « sélectionné » (perte du « focus »)

- Pour trouver quel bouton radio a été cliqué, on peut par exemple utiliser un tableau :

```
var radios = document.getElementsByName("tempFormat");

for (var i = 0, length = radios.length; i < length; i++) {
    if (radios[i].checked) {
        var radioButtonValue = radios[i].value;
        // Un seul bouton radio peut être sélectionné à la fois,
        // donc inutile de continuer à boucler.
        break;
    }
}
```

JS AVANCÉ : PROGRAMMATION FONCTIONNELLE

- Au lieu d'utiliser une boucle sur les tableaux, programmation fonctionnelle possible
- Avec ces fonctions : `forEach()`, `map()`, `every()`, `some()`, `filter()`
- On leur passe en paramètre une fonction, qui est appliquée à chaque élément du tableau
- Cette fonction peut prendre trois paramètres : l'élément, son index, et le tableau
- Mais souvent on n'utilise qu'un seul paramètre, ou les deux premiers. Exemple :

```
demoP = document.getElementById("demo");
var numbers = [4, 9, 16, 25];
function myFunction(item, index) {
      demoP.innerHTML = demoP.innerHTML + "index[" + index
                        + "]: " + item + "<br />";
}
numbers.forEach(myFunction);
```

index[0]: 4
index[1]: 9
index[2]: 16
index[3]: 25

JS AVANCÉ : PROGRAMMATION FONCTIONNELLE

- On peut aussi utiliser une fonction anonyme :

```
elements.forEach(function(element) { element.innerHTML = ""; });
```

- Différence entre les différentes fonctions :
- `forEach()` applique une fonction sans valeur de retour à chaque élément du tableau
- `map()` applique une fonction `f()` à chaque élément du tableau, chaque élément `e` est remplacé par `f(e)`. Le résultat est donc un tableau de la même longueur :

```
myArray = [1,2,3,4];
myArray.map(function (element) { return element * element; });
// À la fin, myArray vaut [1, 4, 9, 16]
```

JS AVANCÉ : PROGRAMMATION FONCTIONNELLE

- `filter()` applique une fonction à chaque élément `e` du tableau, et garde l'élément dans le tableau ssi `f(e)` vaut `true`.

```
a = [5, 4, 3, 2, 1];
smallvalues = a.filter(function(x) { return x < 3 }); // [2, 1]
everyother = a.filter(function(x, i) { return i % 2 == 0 }); // [5, 3, 1]
```

- `every()` applique une fonction à chaque élément e du tableau, et renvoie true ssi `f()` a renvoyé `true` pour **tous** les éléments

- `some()` applique une fonction à chaque élément e du tableau, et renvoie true ssi `f()` a renvoyé `true` pour **au moins un** les éléments

JS AVANCÉ : PROGRAMMATION FONCTIONNELLE

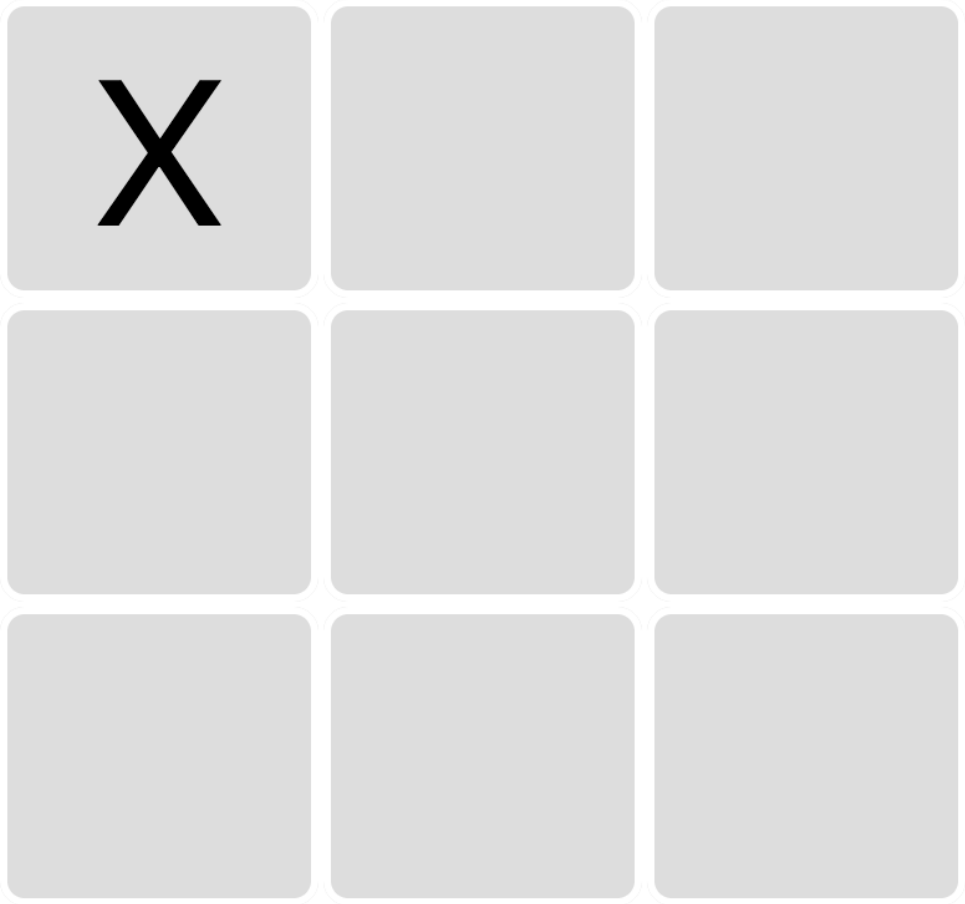
```
var ages = [32, 33, 16, 40];

function checkAdult(age) { return age >= 18; }

document.getElementById("demo").innerHTML = ages.every(checkAdult);
document.getElementById("demo").innerHTML += ", " + ages.some(checkAdult);
```

Que contiendra l'élément demo ?

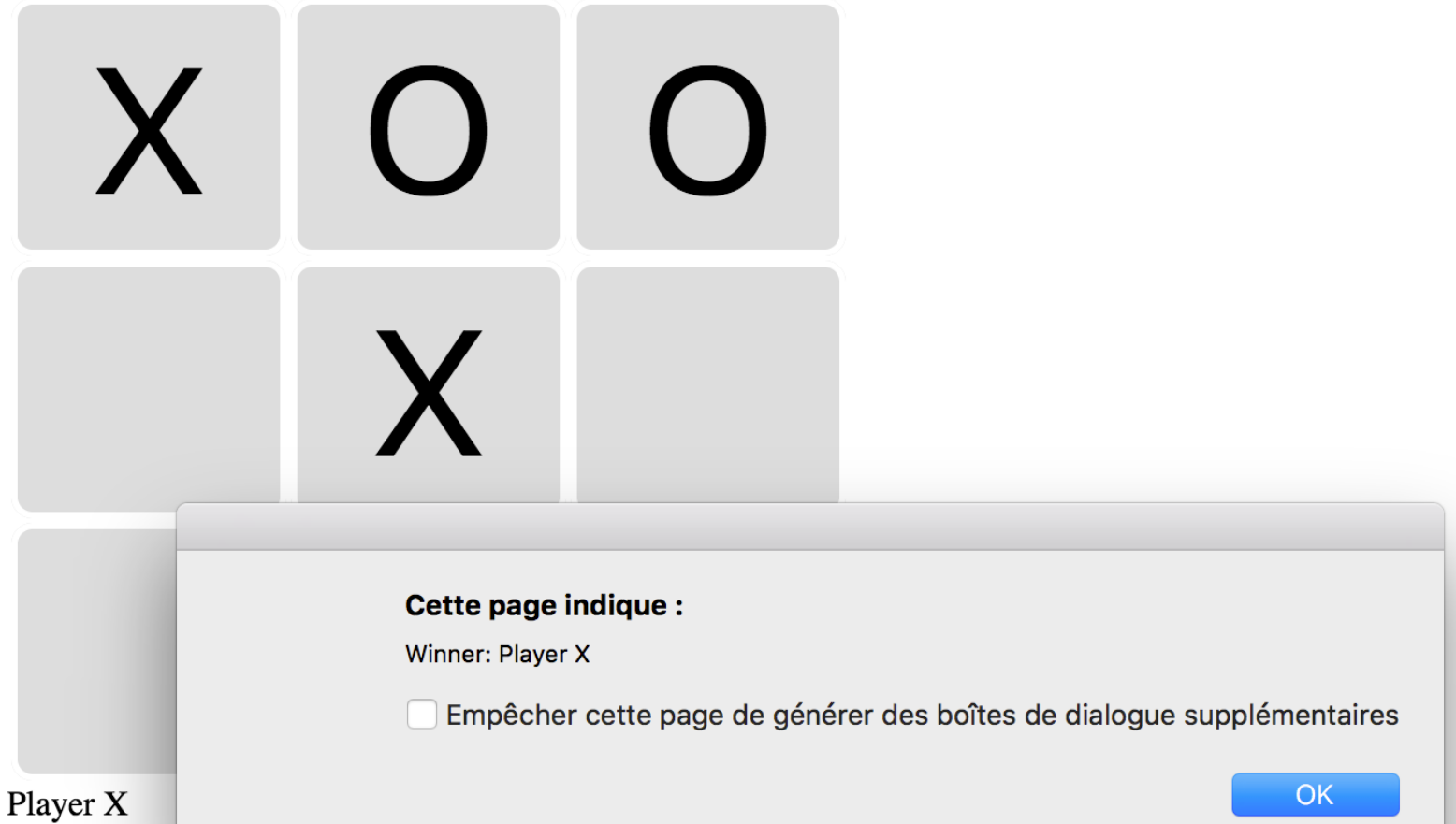
UN EXEMPLE : JEU DE TIC-TAC-TOE

Player X

UN EXEMPLE : JEU DE TIC-TAC-TOE



UN EXEMPLE : JEU DE TIC-TAC-TOE



UN EXEMPLE : JEU DE TIC-TAC-TOE

```
<html>

<head>
<title>Tic-tac-toe</title>
<link rel="stylesheet" type="text/css" href="tictactoe.css" />
<script type="text/javascript" src="tictactoe.js"></script>
</head>

<body onload="init()">
<div id="tictactoe"></div>
<div id="turn">Player X</span>
</body>

</html>
```

UN EXEMPLE : JEU DE TIC-TAC-TOE

```
table
{
    border-width: 0px;
}

td
{
    background-color: #dddddd;
    border: 3px solid white;
    font-size: 80px;
    font-family: Verdana, sans-serif;
    color: #000000;
    border-radius: 10px 10px 10px 10px;
}
```

UN EXEMPLE : JEU DE TIC-TAC-TOE

```
/* Source: https://github.com/vasanthk/tic-tac-toe-js */
var N_SIZE = 3,         // Taille de la grille
    EMPTY = " ", // Contenu d'une case vide
    boxes = [],         // Tableau JS qui contient comme élément chaque td
                        // de la grille
    turn = "X",         // Vaut "X" ou "O"
    moves;              // Nombre de mouvements
```

UN EXEMPLE : JEU DE TIC-TAC-TOE

```
function init() {
    var board = document.createElement('table');
    for (var i = 0; i < N_SIZE; i++) {
        var row = document.createElement('tr');
        board.appendChild(row);
        for (var j = 0; j < N_SIZE; j++) {
            var cell = document.createElement('td');
            cell.style.height = 120;
            cell.style.width = 120;
            cell.style.textAlign 'center';
            cell.style.verticalAlign = 'center';
            cell.classList.add('col' + j, 'row' + i);
            if (i == j) cell.classList.add('diagonal0');
            if (j == N_SIZE - i - 1) cell.classList.add('diagonal1');
            cell.addEventListener("click", set);
            row.appendChild(cell);
            boxes.push(cell);
        }
    }
    document.getElementById("tictactoe").appendChild(board);
    startNewGame();
}
```

col0, row0, diagonal0	col1, row0	col2, row0, diagonal1
col0, row1	col1, row1, diagonal0, diagonal1	col2, row1
col0, row2, diagonal1	col1, row2	col2, row2, diagonal0

UN EXEMPLE : JEU DE TIC-TAC-TOE

```
function startNewGame() {
    moves = 0;
    turn = "X";
    boxes.forEach(function(square) { square.innerHTML = EMPTY; });
}
```

UN EXEMPLE : JEU DE TIC-TAC-TOE

```
function set() {
    if (this.innerHTML !== EMPTY)
        return;
    this.innerHTML = turn;
    moves += 1;
    if (win(this)) {
        alert('Winner: Player ' + turn);
        startNewGame();
    }
    else if (moves === N_SIZE * N_SIZE) {
        alert("Draw");
        startNewGame();
    }
    else {
        if (turn === "X") turn = "O";
        else turn = "X";
        document.getElementById('turn').textContent = 'Player ' + turn;
    }
}
```

UN EXEMPLE : JEU DE TIC-TAC-TOE

col0, row0, diagonal0	col1, row0	col2, row0, diagonal1
col0, row1	col1, row1, diagonal0, diagonal1	col2, row1
col0, row2, diagonal1	col1, row2	col2, row2, diagonal0

```
function win(clicked) {
    var classes = clicked.classList;
    for (var i = 0; i < classes.length; i++) {
        var testClass = '.' + classes.item(i);
        // Compte combien d'éléments de cette classe (row0, col1, ...) contiennent le même symbole
        var items = contains('#tictactoe ' + testClass, turn);
        // S'il y en a 3 : gagné !
        if (items.length == N_SIZE)
            return true;
    }
    return false;
}

function contains(selector, text) {
    // Récupère tous les éléments de la classe
    var elements = document.querySelectorAll(selector);
    // Ne garde que ceux qui ont le même symbole...
    return Array.prototype.filter.call(elements, function(element) {
        return RegExp(text).test(element.innerHTML);
    });
}
```

JAVASCRIPT : ON S'ARRÊTE LÀ !

- On n'a pas vu tout JavaScript, il s'agit d'un langage de programmation complet, très puissant avec de très nombreuses possibilités
- Vous avez quand même vu les bases, si vous voulez en faire davantage, ressources en ligne très nombreuses
- La prochaine fois, on passe à la programmation côté serveur en PHP !