Workshop I

Javascript

Javascript est un langage de programmation de script interprété orienté objet.

Principalement utilisé pour le web côté client et/ou serveur.

```
<html>
...
<body>
    ...
    <script>
        // instruction;
    </script>
    <script src="./monFichier.js"></script>
</body>
</html>
```

Les commentaires

Il y a deux méthodes pour commenter :

- Sur une ligne (//)
- sur plusieurs lignes (/* … */)

```
instruction ; // Commentaire
instruction ; /* Un
long
commentaire */
```

Les variables

Une variable est un espace de stockage permettant d'enregistrer différents types de données.

Syntaxe

- Le nom des variables s'écrit avec des caractères alphanumérique, il peut aussi contenir des underscores et des dollars mais ne peut pas commencer par un caractère numérique ou être un mot réservé.
- Exemple : variable1, variableTropLongue, variable_trop_longue, _variable, $variable

Il est conseillé de toujours déclarer une variable.

Il existe trois méthodes différentes pour déclarer une variable : var, let et const.

Les variables

Les différences entre var, let et const sont la portée (le scope) et le niveau de lecture.

Pour la portée on parlera de bloc, grossièrement un bloc est le contenu entre deux accolades.

```
{
//bloc parent
    {
    // bloc enfant
    }
}
```

Les variables : var

var à une portée sur l'ensemble de son bloc et de ses blocs enfants.

```
{
    var maVariable = 42;
    console.log(maVariable);
    // => 42
    {
        console.log(maVariable);
        // => 42
        maVariable = 'Hello';
    }
    console.log(maVariable);
    // => Hello
}
```

La valeur de var peut-être redéfini après sa déclaration.

Les variables : let

let à une portée sur l'ensemble de son bloc uniquement.

```
{
    let maVariable = 42;
    console.log(maVariable);
    // => 42
    {
        console.log(maVariable);
        // => ReferenceError maVariable is not defined
    }
    maVariable = 'Hello';
    console.log(maVariable);
    // => Hello
}
```

La valeur de let peut-être redéfini après sa déclaration.

Les variables : const

const à une portée globale à tous les blocs.

```
const maVariable = 42;
console.log(maVariable);
// => 42
    {
        console.log(maVariable);
        // => 42
    }
maVariable = 'Hello';
// => TypeError : Assigment to constant variable
console.log(maVariable);
// => 42
```

La valeur de const est constante, elle ne peut pas être redéfini après sa déclaration.

Les variables non-déclarées

Une variable non-déclarée à une portée globale à tous les blocs.

```
{
    maVariable = 42;
    console.log(maVariable);
    // => 42
}
console.log(maVariable);
// => 42
```

Les conditions

Une condition fonctionne avec des opérateurs de comparaison (==, !=, ===, !==, >, >=, < et <=) et des booléens (true et false).

```
var maVariable = 1 > 0;
console.log(maVariable);
// => true
```

Il existe quatre méthodes différentes pour faire une condition : if else, else if, switch et les ternaires.

Il est possible de combiner les conditions avec les opérateurs logique (&& et ||).

```
var maVariable = 1 > 0 && 42 > 0;
console.log(maVariable);
// => true
```

Les conditions : if else

if exécute le bloc si la condition retourne true, sinon exécute else.

```
var maVariable = 50;
if (maVariable != 42) {
    maVariable = 42;
} else {
    maVariable = 'Hello';
}
console.log(maVariable);
// => 42
```

Les conditions : else if

else if rajoute une étape à la structure conditionnelle.

Comme if, else if exécute le bloc si la condition retourne true, sinon passe à la condition suivante jusqu'à else.

```
var maVariable = 50;
if (maVariable < 42) {
    maVariable = 0;
} else if (maVariable === '50') {
    maVariable = 100;
} else if (maVariable != 42) {
    maVariable = 42;
} else{
    maVariable = 'Hello';
}
console.log(maVariable);
// => 42
```

Les conditions : switch

switch est une structure conditionnelle similaire à une multitude de else if.

switch exécute le bloc case si la condition comparative === retourne true, sinon passe à la condition suivante jusqu'à default.

```
var maVariable = 42;
switch (maVariable) {
    case 0 :
        console.log('maVariable retourne le nombre 0');
    break;

    case '42' :
        console.log('maVariable retourne la chaîne de caractères 42');
    break;

    case 42 :
        console.log('maVariable retourne le nombre 42');
    break;

    default :
        console.log('maVariable retourne quelque chose');
}
// => maVariable retourne le nombre 42
```

break est obligatoire après chaque case.

Les conditions : les ternaires

Les ternaires est une structure conditionnelle similaire à un if else.

```
condition ? return true : return false ;
```

Si la condition retourne true, le premier bloc est retourné, sinon c'est le second bloc qui est retourné.

```
var maVariable = 42;
console.log(maVariable == 42 ? maVariable : 0);
// => 42
console.log(maVariable > 100 ? maVariable : 0);
// => 0
```

Les boucles

Une boucle permet de répéter plusieurs fois la même instructions.

Il existe trois méthodes différentes pour faire une boucle : while, do while et for.

On appelle une répétition, une itération. Le mot réservé break peut-être utilisé pour arrêter une itération.

Les boucles : while

while s'exécute et se répète si la condition retourne true.

```
var maVariable = 50;
console.log(maVariable);
// => 50
while (maVariable > 42) {
    --maVariable;
}
console.log(maVariable);
// => 42
```

Les boucles : do while

do while s'exécute une fois et se répète si la condition retourne true.

```
var maVariable = 42;
console.log(maVariable);
// => 42
do {
    --maVariable;
} while (maVariable > 42);
console.log(maVariable);
// => 41
```

Les boucles : for

for s'exécute et se répète si la condition retourne true.

```
for (initialisation ; condition ; incrémentation) {
    instruction;
}
```

```
var maVariable;
for (maVariable = 0 ; maVariable < 42 ; maVariable++) {
}
console.log(maVariable);
// => 42
```

Le contexte d’exécution

À chaque fois qu’on exécute une instruction, on entre dans un contexte d’exécution.

Grossièrement à chaque fois que l’on change de bloc, le contexte change.

On utilisera le mot réservé this pour parler du contexte.

Les fonctions

Une fonction (function) est un ensemble d'instructions permettant d'exécuter ou de retourner quelque chose.

Syntaxe

- Le nom des fonctions s'écrit avec des caractères alphanumérique, il peut aussi contenir des underscores et des dollars mais ne peut pas commencer par un caractère numérique ou être un mot réservé.
- Exemple : function1, functionTropLongue, function_trop_longue, _function, $function

Le mot réservé return permet de retourner une valeur.

Les fonctions standards

Voici des exemples de function :

```
function maFonction(arg) {
    console.log('je suis une fonction avec un argument ' + arg);
}

maFonction(42);
// => je suis une fonction avec un argument 42
```

```
function maFonction(arg) {
    return 'je suis une fonction avec un argument ' + arg;
}

console.log(maFonction(42));
// => je suis une fonction avec un argument 42
```

Les fonctions anonymes

Une fonction anonyme permet d'utiliser des variables temporaires.

Voici une exemple d'une fonction anonyme :

```
(function (arg) {
    var maVariable = 2 + arg;
    console.log('je suis une fonction avec un argument ' + maVariable);
}) (40);
// => je suis une fonction avec un argument 42

console.log(maVariable);
// => ReferenceError maVariable is not defined
```

Les fonctions fléchées

Une fonction fléchées a une syntaxe plus courte et est anonyme.

```
(param1, param2, …, paramN) => expression
```

Voici des exemples de fonctions fléchées :

```
var addition = (a,b) => a+b;
console.log(addition(40, 2));
// => 42

var soustraction = (a,b) => a-b;
console.log(soustraction(50, 8));
// => 42

var maTable = ['Hello', 'world !'];
console.log(maTable.map(item => item.length));
// => Array [5, 7]
```

Une fonction fléchées ne possède pas de contexte d'exécution et utilise le contexte du scope parent.