Javascript module 2

Tableaux, objets et boucles

Les tableaux, ou array

Déclarer un array

Permet de stocker plusieurs variables à différentes positions.

On peut instancier un array vide de 2 façons différentes.

```
let players = [];

// or

let players = new Array();
```

Pour le déclarer avec des valeurs, on le délimite avec des crochets. Chaque valeur doit être séparée par une virgule.

La propriété ***.length*** permet d'obtenir le nombre d'éléments dans le tableau.

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];

typeof players; // object

players.length; // 5


players[0]; // Tim

players[2]; // Abdel
```

Tableaux à plusieurs dimensions

Pour de nombreuses applications, il est très pratique d'utiliser des tableaux à plusieurs dimensions. Il s'agit d'un tableau qui contient lui-même des tableaux à chaque index. Cette imbrication peut elle-même se répéter selon le nombre de dimensions nécessaire.

Voici un exemple de tableau à 2 dimensions :

```
const players  = [
    ["Tim", "Sonia", "Abdel"],
    [24, 32, 34],
    [76, 27, 14]
```

```
];
```

Méthodes pour manipuler les array

Tester un array

Pour tester si une variable est un array, nous ne pouvons pas nous contenter de ***typeof*** qui retournera “object”. Il faut utiliser la méthode ***Array.isArray()***.

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];

typeof players; // "object"

Array.isArray(players); // true
```

Ajouter des éléments

Vous pouvez ajouter un ou plusieurs éléments au début d’un array avec la méthode ***.push()***.

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];

players.push("Loïc");

// ["Tim", "Audrey", "Abdel", "Kevin", "Samia", "Loïc"]

players.push("Loanne", "JM");

// [ "Tim", "Audrey", "Abdel", "Kevin", "Samia", "Loïc", "Loanne", "JM" ]
```

Vous pouvez de la même manière ajouter des éléments au début avec la méthode ***.unshift()***;

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];

players.unshift("Loïc");

// ["Tim", "Audrey", "Abdel", "Kevin", "Samia", "Loïc"]
```

Extraire des éléments

La méthode ***.pop()*** permet d’extraire la dernière valeur d’un tableau. La méthode retourne cette valeur. L’array ne contient plus cette valeur.

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];

let lastPlayer = players.pop(); // Samia
```

```
players; // ["Tim", "Audrey", "Abdel", "Kevin"]
```

Il est également possible d'extraire la première valeur d'un tableau de la même manière avec la méthode ***.shift()***.

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];

let firstPlayer = players.shift(); // Tim

players; // ["Audrey", "Abdel", "Kevin", "Samia"]
```

Aller plus loin

Il existe de nombreuses méthodes permettant de manipuler des array. Vous en trouverez une liste complète ici :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array#m%C3%A9thodes_des_instances

Les objets

Le type object permet de stocker des couples clé / valeur.

Il se déclare entre des accolades ***{ }***.

Chaque couple clé/valeur est séparé par une virgule.

La clé et la valeur sont séparés par ***:***.

```
const player = {
    name: "Charlie",
    xp: 4,
    weapon: 8,
    shield: 6
};
```

Pour accéder ou modifier la valeur d'une clé on utilise un point "." après l'object.

```
player.name; // "Charlie"
player.xp; // 4
```

```
player.shield = 8;
player.xp++;
```

Pour connaître le nombre d'éléments d'un tableau vous pouvez utiliser la déclaration suivante :

```
const a = {a: 1, b: 2};

Object.keys(a).length; // 2
```

Les objects peuvent être imbriqués dans un array ou un autre object, tout comme les array à plusieurs dimensions.

Voici un exemple de tableau d'objets.

```
const characters = [
    {
        name: "Godzilla",
        health: 8,
        force: 9
    },
    {
        name: "Kong",
        health: 10,
        force: 8
    },
    {
        name: "Hulk",
        health: 7,
        force: 5
    }
];

characters[1].health; // 10
```

Voici un autre exemple d'un object contenant des objects pour chaque clé.

```
const characters = {
    Godzilla: {
        life: 8,
```

```
        force: 9
    },
    Kong: {
        life: 10,
        force: 8
    },
    Hulk: {
        life: 7,
        force: 5
    }
};

console.log(characters.Godzilla.life); // 8
```

L'opérateur ***delete*** permet de retirer une propriété d'un object.

```
delete characters.Hulk;
```

Array et Object dans la console

Pour afficher de manière lisible les array et les object dans la console des navigateurs (même à plusieurs dimensions) vous pouvez utiliser la commande suivante :

```
console.table(myVar);
```

Les boucles

Les boucles sont des outils qui permettent de répéter l'exécution d'instructions selon des conditions. Nous allons notamment voir comment se servir des différentes boucles pour parcourir et manipuler les données des array et des object.

Boucle for classique

La boucle ***for*** permet d'exécuter les instructions qu'il contient tant qu'une condition est remplie. Les paramètres à séparer par des ***;*** sont les suivants :

- expression initiale : permet d'initialiser un ou plusieurs compteurs ;
- condition : exécute le code contenu dans la boucle tant que cette condition est vérifiée ;

- expression d'incrément : permet d'incrémenter les compteurs.

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
// 0
// 1
// 2
// 3
// 4
```

Lorsque l'on souhaite parcourir un tableau avec une boucle ***for***, nous l'utilisons comme ceci.

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];
for (let i = 0; i < players.length; i++) {
    console.log(i, players[i]);
}
```

Boucle for ... in

La boucle ***for ... in*** donne la possibilité de parcourir un object de par ses clés.

```
const object = {a: 1, b: 2, c: 3};
for (const prop in object) {
  console.log(`${prop}: ${object[prop]}`);
}
// "a: 1"
// "b: 2"
// "c: 3"
```

Cela est également valable pour parcourir un array de par ses index.

```
const array = ["a", "b", "c"];
for (const i in array) {
  console.log(`${i}: ${array[i]}`);
}
// "0: a"
// "1: b"
```

```
// "2: c"
```

Boucle for ... of

La boucle ***for ... of*** permet de parcourir un array en retournant directement la valeur pour chaque index. ***for ... of*** ne permet pas de manipuler directement un object.

```
const object = ["a", "b", "c"];
for (const value of object) {
    console.log(value);
}
// "a"
// "b"
// "c"
```

Boucle while

Une boucle ***while*** permet d'exécuter les instructions qu'il contient tant que la condition est vérifiée.

```
while (condition) {
    // instructions
}
```

Voici l'équivalent d'un ***for*** classique avec un ***while*** :

```
let players = ["Tim", "Audrey", "Abdel", "Kevin", "Samia"];
let i = 0;
while (i < players.length) {
    console.log(players[i]);
    i++;
}
```

L'intérêt du ***while*** est de pouvoir utiliser une boucle avec une condition d'exécution plus complexe qu'un simple compteur.

```
let i = 0;
while (i < 5) {
    console.log(i);
```

```
    if (i >= 3) break;
    i++;
}
```

L'instruction ***break*** permet de terminer et de sortir immédiatement de la boucle.

Le *array.forEach*

Le ***forEach*** s'exécute sur un array et permet de le parcourir, pas sur un object.

Il permet de parcourir tous les éléments du tableau sur lequel la méthode est appliquée et d'y exécuter une fonction de callback (souvent anonyme) à laquelle peuvent être passé les 3 arguments suivants :

- la valeur à la position courante ;
- l'index de la position courante (facultatif) ;
- le tableau sur lequel la méthode est appliquée (facultatif).

Voici un exemple d'utilisation avec un array contenant des objects.

```
const characters = [
    {name: "Godzilla", life: 8, force: 9},
    {name: "Kong", life: 10, force: 8},
    {name: "Hulk", life: 7, force: 5}
];

characters.forEach(function(c, i) {
    console.log(`#${i} ${c.name} life: ${c.life}`);
});
```

Vous rencontrerez souvent ***forEach*** utilisé avec une fonction fléchée (arrow function, qui seront abordées au module suivant). L'utilisation précédente du ***forEach*** se résumerait alors à cette ligne :

```
characters.forEach((c, i) => console.log(`#${i} ${c.name} life:
${c.life}`));
```