Javascript

Les variables

Déclaration

```
let a = 2;
var b = 4;
```

Opérations arithmithéthiques

```
+ - * / ++ -- += -+ *= /= % mod
```

Les constantes

```
const nombreConst = 20;
nombreConst = 30; // Retournera une erreur dans la console car on
ne peut plus changer sa valeur
```

Résumé

Nous avons vu les bases des variables en JavaScript. Vous savez désormais :

- **déclarer** les variables par les mots clé *let* ou var et un identifiant, et les **initialiser** avec l'opérateur = ;
- **modifier** le contenu d'une variable en la réaffectant, ou avec des opérateurs ;
- utiliser des **constantes** pour éviter le remplacement d'éléments de données essentiels.

Les types de variables

Important : *JavaScript est un langage dit à types dynamiques et à typage faible. Cela signifie que vous pouvez initialiser une variable en tant que nombre, puis la réaffecter comme chaîne, ou tout autre type de variable. Ceci offre une grande souplesse, mais peut aussi conduire à un comportement inattendu si vous opérez sans précaution.*

En JavaScript, il y a trois types primitifs principaux :

- number (nombre) ;
- string (chaîne de caractères) ;
- boolean (valeur logique).

Les variables de type number peuvent être positives ou négatives. Elles peuvent aussi être des nombres entiers (1, 2, 3, etc.) ou décimaux (1,4 ; 67,34 ; etc.).

```
let integerCalculation = 1 + 2;
```

Les variables booleans sont le plus simple des types primitifs : elles ne peuvent avoir que deux valeurs, true ou false (vrai ou faux).

```
let userIsSignedIn = true;
```

```
let userIsAdmin = false;
```

Les chaînes de caractères (chaînes, ou *strings*, en anglais) sont la façon d'enregistrer du **texte** dans des variables JavaScript.

Les variables de type string sont encadrées par des guillemets simples ou doubles –' ou " :

```
let firstName = "Youssouf";
let lastName = 'EL ALLIOUI';
```

Opérateur de concaténation : +

```
let monNom = firstName + " " + lastName;  // valeur: "Will Alexander"
```

la string interpolation

C'est une écriture qui simplifie la concaténation des variables et des chaînes de caractère. Pour créer une string interpolation on écrit du texte encadrée par le signe ` et si on veut injecter une variable dans ce code on utilise l'expression ${maVariable}.

```
const myName =  `Youssouf`;
const salutation = `Bienvenue sur mon site ${myName}!`;
alert(salutation);   //retournera "Bienvenue sur mon site Youssouf!"
```

Résumé

Il y a trois principaux types de données primitifs en JavaScript :

- **number** (nombre) ;
- **boolean** (valeur logique) ;
- **string** (chaîne de caractères).

Types complexes, classes et objets

Les objets

Les objets JavaScript sont écrits en JSON (JavaScript Object Notation). Ce sont des séries de paires clés-valeurs séparées par des virgules, entre des accolades. Les objets peuvent être enregistrés dans une variable :

```
let myBook = {
    title: 'Le pain nu',
    author: 'Mohammed CHOUKRI',
    pages: 160,
    isAvailable: true
};
```

Accédez aux données d'un objet

Pour cela, utilisez le nom de la variable qui contient l'objet, un point (.), puis le nom de la clé dont vous souhaitez récupérer la valeur.

```
let bookTitle = myBook.title;                                    // "Le pain nu"
```

```
let bookPages = myBook.pages                    // 160
```

Ou bien, en utilisant les brackets avec la valeur du sous élément :

```
let bookTitle = myBook["title"];                        // "Le pain nu"
let bookPages = myBook["pages"];                // 160
```

Les classes

une classe est un **modèle** pour un objet dans le code. Elle permet de construire plusieurs objets du même type (appelés *instances de la même classe*) plus facilement, rapidement et en toute fiabilité.

Comment Créer une classe ?

➔ Utilisez le mot clé class, suivi par un nom et encadrez le code de la classe entre accolades comme dans l'exemple suivant :

```
class Book {
        // Le code de la classe Book
}
```

Dans cet exemple, nous souhaitons que chaque Book ait un titre, un auteur et un nombre de pages. Pour cela, vous utilisez ce qu'on appelle un constructor :

```
class Book {
        constructor (title, author, pages) {

        }
}
```

Pour attribuer le titre, l'auteur et le nombre de pages reçus à cette instance, utilisez le mot clé this et la notation dot :

```
class Book {
      constructor(title, author, pages) {
          this.title = title;
          this.author = author;
          this.pages = pages;
      }
}
```

Maintenant que la classe est terminée, vous pouvez créer des instances par le mot clé new :

```
let myBook = new Book("Le pain nu", "Mohamed CHOUKRI", 160);

let bookTitle = myBook.title;                   // "Le pain nu"
let bookPages = myBook.pages                    // 160
```

Résumé

Nous avons découvert :

- les objets avec les paires clés-valeurs en notation JSON.
- la notation pointée (dot) qui donne accès aux valeurs d'un objet et à la possibilité de les modifier ;
- les classes, et comment l'utilisation de classes peut vous permettre de créer des objets plus facilement et de façon plus lisible.

Les tableaux (array) en javascript

Créer un tableau vide, utilisez une paire de crochets :

```
let amis = [];
```

Créer un tableau rempli on doit placer les éléments voulus à l'intérieur de ces crochets :

```
let amis = ["Ali ALI", "Khalid KHALID", "Leila LEILA"];
```

Accéder aux éléments de ce tableau par leur indice :

```
let ami_1 = amis[0];       // "Ali ALI"
let ami_2 = amis[2];       // "Khalid KHALID"
let ami_12 = amis[12]      // undefined
```

Passage par valeur VS passage par référence en javascript

➔ En JavaScript, les types primitifs tels que les nombres, les valeurs logiques et les chaînes sont passés par **valeur**. Les objets et les tableaux sont passés par **référence**.

Voici un exemple :

```
// Cration d'un objet etudiant
let etudiant = {
    name: "Ali ALI",
    age: 20,
    available: true
};

// nouveau tableau contenant l'objet ci-dessus
let toutLesEtudiants = [etudiant];

// affichegage de etudiant avant la modification
alert(etudiant.available) ;

// modification de l'objet
toutLesEtudiants[0].available = false;

// affichegage de etudiant après la modification
alert(etudiant.available) ;
```

Attributs et méthodes très utiles

length pour le comptage d'éléments :

```
let amis = ["Ali ALI", "Khalid KHALID", "Leila LEILA"];
let nombreDesAmis = amis.length; // 3
```

L'ajout et la suppression d'éléments

Pour **ajouter un élément à la fin d'un tableau,** utilisez sa méthode push :

```
// ajoute "Khadija KHADIJA" à la fin de notre tableau amis
amis.push("Khadija KHADIJA");
```

Pour **ajouter votre élément au début du tableau** plutôt qu'à la fin, utilisez la méthode **unshift** :

```
// Ajoute "Adil ADIL" au début du tableau amis
amis.unshift("Adil ADIL");
```

Pour **supprimer le dernier élément d'un tableau,** appelez sa méthode **pop** , sans passer aucun argument :

```
// supprimer le dernier élément du tableau amis
amis.pop();
```

Résumé

- Nous avons appris à connaître les **collections** ;
- Nous avons exploré la collection la plus courante en JavaScript : le **tableau** ;
- Nous avons appris à *créer* des tableaux, à les *remplir*, et vu certains outils de base pour les *manipuler*.

Les instructions de contrôle en javascript

Les instructions if/else

Les instructions switch

Résumé

Nous avons :

- appris le fonctionnement des instructions *if/else* ;
- vu les différents types de conditions pouvant être utilisés pour les instructions *if/else* ;
- appris à regrouper les différentes conditions avec des opérateurs logiques ;
- exploré la portée des variables, et les conséquences qu'elle a sur la structure du code ;
- découvert l'instruction *switch* pour comparaison à une liste de valeurs attendues.

Les instru ctons itératives en ȷa vascript

Les boucles for

Les boucle while

Résumé

Nous avons abordé deux façons de répéter des tâches (morceau de code) :

- la boucle *for* , pour un nombre d'itérations connu ;
- la boucle *while* , pour un nombre d'itérations inconnu.

Les fonctions en javascript

Une **fonction** est un bloc de code auquel vous attribuez un nom. Quand vous **appelez** cette fonction, vous exécutez le code qu'elle contient.

Définir une fonction

Quand vous créez ou **déclarez** une fonction :

- Vous définissez son **nom**
- Vous indiquez la liste des variables dont elle a besoin pour effectuer son travail : les **paramètres** de la fonction.
- Vous indiquez sa **valeur de retour** s'il y a besoin.

```
// Créer une foction somme :
function somme (a, b) {
    let r;
    r = a + b;
    return r;
}
```

Appeler une fonction

Quand vous **appelez** cette fonction, vous exécutez le code qu'elle contient.

```
let x = 3;
let y = 7;

// Appeler la fonction somme :
alert (somme(x, y));
```

Méthodes d'instance

les méthodes d'instance, sont les méthodes qui agissent sur les instances individuelles d'une classe.

```
class Book {
    constructor(title, author, pages) {
        this.title = title;
        this.author = author;
        this.pages = pages;
    }

    printBookInfo() {
        alert(this.title + ", " + this.author + ", " + this.pages);
    }
}

let myBook = new Book("Le pain nu", "Mohamed CHOUKRI", 160);

myBook.printBookInfo();
```

Méthodes de classe

Les méthodes de classe, appelées aussi méthodes statiques, sont les méthodes qui s'appuient sur tous les instances d'une classe et pas sur chaque instance.