0.1 Les variables

Révisions :

```
const CONTAINER = document.getElementById('
    container');

console.log(CONTAINER.id) /* => container */
```

Dans les bonnes pratiques vous écrivez les noms des constantes en lettres majuscules.

Ci-dessus CONTAINER est une variable assignée comme constante qui contient un élément HTML en partant du pricnipe que l'élément ayant l'id container existe réellement dans le document (DOM).

Etant donné que tout élément HTML est un objet alors CONTAINER qui contient en valeur un élément HTML est un objet.

Donc CONTAINER possède des propriétés et des méthodes accessibles via la notation par point (dot notation en anglais)

0.2 TypeScript

La première idée c'est que TypeScript **est** du JavaScript mais que le JavaScript n'est pas du TypeScript en ce sens où le TS défit les possibilités de JavaScript et écrase complètement son adversaire, son grand-mère.

Cybersécurité TS est en fait un langage indiquée à une époque où typer ses variables est une nécessité pour éviter ou essayer d'éviter à tout le moins des attaques XSS dévastatrices par obscurcissement ou des requêtes chanceuses ou d'ingénierie sociale vers des routes de serveur qui auraient pris à la légère le type des valeurs que les contrôleurs (middlewares / intergiciels du serveur) interceptent lorsqu'une requête HTTPS arrive vers eux.

Pour pouvoir transpiler c'est-à-dire convertir d'un langage de même niveau d'abstraction à un autre, il faut un "transpiler". En l'occurence celui que nous utilisons est tsc, un module que nous pouvons récupérer via le node package manager, le gestionnair de paquets de Node.js environnement d'exécution capable de comprendre grâce à son moteur V8 tout code écrit en JavaScript.

Dépendances de développement TSC notre transpiler peut être une bibliothèque, un module dont notre application dépend pour bien fonctionnée

ou tout simplement pour fonctionner. On parle de dépendance. Ainsi pour donner quelques exemples, un bibliothèque de cryptage de données est nécessaire lorsque vous voulez stocker dans une base de données des motes de passe. Une bibliothèque d'encodage de jetons d'authentication est nécessaire si vous voulez vous assurez de la sécurité d'une session...

Pour installer des dépendances vous pouvez initialiser un package.json avec la commande npm init -y

Le flag -y appelé également l'option y vous permet d'éviter de répondre aux questions les uns après les autres lors de la création du package.json. Avec ce flag vous répondez oui (yes) à toutes les questions d'un coup et cela vous crée le package.json à l'endroit où vous avez lancer la commande npm init.

Dépendance typescript dans notre projet En l'occurence ce que nous voulons c'est nous assurer que nos fichiers .ts sont transpilés en JavaScript donc nous téléchargeons la dépendance typescript qui elle-même dépendant du module tsc dont nous avons besoin.

Les deux seront donc téléchargés mais seule la dépendance typescript sera visible dans notre package.json qui référencie les dépendances de notre application tandis que son lock.json verrouille les versions de dépendances que nous utilisons pour que lors de travaux collaboratifs, nos collègues travaillant sur la même application utilisent les mêmes versions afin d'obtenir les mêmes résultat.

Node Modules Nous installons la dépendance typescript localement au sein du dossier racine qui représente notre application. Vous verez ainsi si c'est la premiere dépendance que vous installez apparaitre un dossier node_modules dans le dossier racine de votre application.

```
npm install typescript --save-dev
```

Il y a d'autres commandes qui permettent d'aboutir au même résultat

```
npm install typescript -D
npm i -D typescript
```

Pour transpiler tous les fichiers TypeScript en JavaScript vous écrivez dans le package json un nouveau script

```
"scripts": {
    "transpile": "tsc --project ./"
}
```

Ce script va dire au système d'utiliser la dépendance tsc dont dépend votre dépendance typescript pour transpiler, en suivant les directives de votre tsconfig.json, tous les fichiers .ts en JS"

Pour lancer la transpilation vous devez taper à la racine de votre application la commande suivante :

```
npm run transpile
```

0.3 Le DOM

L'un des concepts les plus importants lorsqu'on travaille avec JavaScript du côté du navigateur, du côté du client, est de comprendre que tout élément HTML est un objet autrement dit qu'il a des propriétés et des méthodes.

Chaque balise que vous mettez ou qui existe dans vos pages html représentent des éléments HTML. Elements qu'il vous appartient de contrôler, de protéger, de modifier en fonction d'événements ou fonction du comportement des utilisateurs si besoin.

```
const button = document.getElementById('submit-button')
```

Explications Je stocke dans la variable button un élément HTML en l'occurence celui qui a comme identifiant 'submit-button'

Le contrôle est le mot clé qui introduit à la notion de conditions.

0.4 Les conditions

Les conditions permettent de contrôler l'exécution du flux d'instructions qui composent le programme. Il s'agit de savoir si une ou plusieurs conditions sont remplies, traduisables en tru ou false, pour exécuter telle ou telle partie du code.

```
const count:boolean;
if(count == 0){
    //instruction accessible que si la condition
       est remplie.
    console.log("Starting")
}
```

switch La structure switch permet de gérer les changements de valeur d'une variable donnée comme différents cas aboutissant potentiellement à différentes conséquences.

```
    function ageHandler(age){

    const isAllowed = age > 18
    switch(isAllowed){
        case true:
            console.log('Can come in')
            break;
        case false;
            console.error('Cannot come in')
            break;
        default;
            break;
    }

    ageHandler(30);

}
```

0.5 Les boucles

Imaginons la situation où nous avons plusieurs individus qu'il faut que notre logiciel soit capable d'autoriser à accéder à une discothèque

L'idée est d'envoyer une requête http à un serveur web et ce dernier va récupéer la liste des individus via une réponse du serveur de base de données et vérifier si l'age des différentes personnes est supérieur à 18 ans.

Le formulaire de l'application front envoie dans la requête vers le serveur le tableau suivant :

['firstname' :'Lucie', 'lastname' : 'Moon', 'email' :'lucie.moon@gmail.com' , 'firstname' :'Marc', 'lastname' : 'Pierre', 'email' :'marc.pierre@gmail.com']

Le serveur reçoit le tableau en format json le parse en JavaScript et utilise une boucle pour vérifier l'âge de chaque objet du tableau reçu l'un après l'autre.

La requête vers la bdd La requête vers le serveur de bdd lui retourne la réponse suivante

```
const bddUsersList = [
    {
        'firstname':'Lucie',
        'lastname': 'Moon',
        'email':'lucie.moon@gmail.com',
        'age':'17',
        'gender':'female'
    },
    {
        'firstname':'Marc',
        'lastname': 'Pierre',
        'email':'marc.pierre@gmail.com',
        'age':'19',
        'gender': 'male'
    }
]
```

Explications Un tableau est un objet particulier, mais un objet n'est pas nécessairement un tableau. Un tableau est un objet ce qui est visible lorsque nous mettons un . derrière lui pour appeler sa méthode forEach() ou sa propriété length.

```
function verifyIfUserInBdd(user) {

    let isFound = false;
    for(const user of list){

        for(const bddUserIndex in bddUsersList){
            if(user.email == bddUsersList[
               bddUserIndex].email){
                isFound = true
                break;
            }
        }
```

```
        }

    }


}
```

0.6 Le formulaire

Les formulaires sont un moyen de communiquer avec l'utilisateur de l'application. Seulement il faut toujours se méfier de l'utilisateur donc il faut nécessairement contrôler ce qu'il entre dans les champs.

L'idée est d'écouter l'événement submit sur l'élément HTML form

```
const form:HTMLElement = document.getElementById('form
    ')
```

Explications Toujours en TS je déclare une variable form de type HTMLElement et je stocke dedans l'objet HTML qui a comme id "form".

```
    const form:HTMLElement = document.getElementById('
        form')

    const submitHandler = (e:any) => {
        e.preventDefault();
        const email = document.getElementById('email')
            as HTMLInputElement;
        const password = document.getElementById('
            password') as HTMLInputElement;


        const credentials = [email.value, password.
            value]

        credentials.forEach((credential, index) => {
            console.log('credential', credential, index
                )
        })
```

```
}

form.addEventListener('submit', (e) =>
    submitHandler(e))
```

Explications Comme vous pouvez le voir dans le corps de la fonction ci-dessus j'empêche (prevent en anglais) à l'événement d'avoir son comportement par défault autrement dit j'empêche à l'événement sous-mission d'avoir lieu.

Tableau Au-dessus j'ai déclaré et initialisé un tableau nommé credentials qui contient deux éléments email.value et password.value donc deux variables chacune étant la propriété value d'un objet de type HTMLInputElement.

0.7 Les fonctions de rappel

```
(e) => submitHandler(e)
```

La fonction fléchèe et anonyme ci-dessus est la fonction de rappel associée à l'événement submit. Quand le formulaire est soumis alors elle est exécutée. Comme vous pouvez le voir c'est une définition pas un appel quand vous la passer en deuxième argument de la méthode addEventListener, méthode utilisée pour écouter un événement particulier concernant un élément HTML également particulier.

0.8 La portée d'une variable

Le fait est qu'une variable a un propriétaire. Elle peut avoir une portée globale à un fichier ou locale à une fonction. La fonction est dite propriétaire de la variable si celle-ci a été déclarée dans son corps.

```
function createHtmlElementAndAppendToFather(
    newElementType: string,
    newElementId: string,
    fatherId: string
): void {
```

```
    const newElement = document.createElement(
        newElementType);
    newElement.id = newElementId
    document.getElementById(fatherId)!.appendChild(
        newElement)
}
```

Explications J'ai écrit la définition de la fonction createHtmlElementAndAppendToFather() en TS en précisant ce qui n'est pas obligatoire sauf si exigé par le tsconfig.json , les types des paramètres attendus, tous ont ici un type string (chaîne de caractères).

Je précise également le type de la valeur de retour mais vu qu'il n'y en a pas j'écris void