JavaScript

Permet de créer du contenu mis à jour de façon dynamique

Dynamique = interaction avec une base de données ou avec des données extérieures

En général on évite de mettre le script dans le head pour ne pas casser le chargement de la page. On le place sur la dernière ligne avant la balise fermante de body.

`<script src=””></script>`

> Il peut y avoir une ambiguïté dans le code avec l’utilisation des apostrophes ‘ selon la langue : “ permet d’écrire “aujourd’hui” sans problème, aussi on peut utiliser les backtick ``

alert()

Créer une pop up avec un message

console.log()

Ecrire un message dans la console

prompt()

Demander quelque chose dans une pop up à l’utilisateur

Les variables

Un conteneur pour une valeur, une sortie de boite qui va stocker une donnée.

Permet d’être réutilisé plusieurs fois dans le code

Deux types de variables :

let - variable modifiable

const - variable ayant une valeur constante, qui n’es pas amenée à changer

Lorsqu’on ne connait pas la valeur de la constante, on mets le nom de la variable en minuscule.

Lorsqu’on connait la valeur, on note en majuscule.

```
const date = new Date():
const KEY_API = "12x678";
```

le dernier type n’est plus utilisé : **var**

Déclarer une variable

On commence par le type :

```
const name = “Clara”;
let age = 25;
```

Préférer le camelCase pour écrire le nom des variables

Différents types de data

number

string

boolean

null

undefined

object

symbol

Fonctions

une séquence d’instructions pour effectuer une tâche spécifique

Deux façons de déclarer une fonction

```
function sayHello() {
}
```

Dans la nouvelle version de JavaScript, on les appelle Fat Arrow Function

```
const sayHello = () => {
}
```

Fonctions anonymes : des fonctions jetables, on s’en sert qu’une fois

Paramètres & arguments

Les paramètres

Les paramètres sont des noms listés dans la **définition** de la fonction. On peut en mettre plusieurs.

```
const sayHello = (firstName, lastName) => {
  console.log(`Bonjour ${firstName} ${lastName}, comment vas-tu ?`);
}
sayHello(James, Bond);
// Affiche "Bonjour James Bond, comment vas-tu ?"
```

Les arguments

Quand on assigne une valeur à un paramètre, il devient un **argument**.

Les **arguments** d’une fonction sont les valeurs réelles passées à la fonction. Un argument sera utilisé dans le corps de la fonction et en modifiera donc le résultat.

```
sayHello("Michel","Dupont");
```

firstName et lastName sont des paramètres, Michel et Dupont sont des arguments

Dans une fonction les paramètres sont traités comme des variables.

Paramètres par défaut

Sinon sans paramètres par défaut, il va afficher `undefined`.

Il faut faire attention à l'ordre des paramètres qui dépends de l'ordre donné dans la définition de la fonction, mettre la valeur par défaut à la fin.

```
const sayHello = (Name = Anonyme) => {
  console.log(`Bonjour ${Name}, comment vas-tu ?`);
}
sayHello();

//va écrire Bonjour Anonyme, comment vas-tu ?
```

Bonnes pratiques :

- on déclare une fonction avant de l'appeler
- le nom d'une fonction doit être explicite
- une fonction ne doit pas être trop longue, sinon on découpe la fonction en plusieurs fonctions
- ne jamais nommer un paramètre "argument"

À lire : Guide de bonne pratique AirBnb style guide

Return

Si je veux enregistrer le résultat d'une fonction dans une variable.

La fonction pourra retourner une valeur.

Le scope

La portée d’une variable, l’endroit depuis laquelle elle peut être utilisée.

Les conditions

- if / if … else

```
let isHungry = true;
if (isHungry) {
console.log("Tu manges")
}
```

- if… else en **opérateur ternaire**

Pratique si c’est une petite condition à tester.

```
const isAWoman = true;
let firstName;

isAWoman ? firstName = "Tammy" : firstName = "Tommy";

console.log("new first name =", firstName);
```

- Opérateurs logiques

&& esperluette : “et”

|| : “ou”. `(Alt + Maj + L.)` Si une des deux conditions est true

! : “n’est pas”

^ : “ou exclusif” une seule des conditions doit être true mais pas les deux

Switch

C’est un **else if** avec une autre sémantique, s’il y a beaucoup de else if c’est une syntaxe plus simple.

switch ⇒ mot clef qui déclare le switch

entre parenthèses ⇒ condition à vérifier et que chaque case va comparer

case ⇒ regarde si l'expression qui la suit valide la condition

break ⇒ stipule de terminer le processus si la condition est vérifiée. OBLIGATOIRE

Default ⇒ valeur par défaut si aucune des conditions ne retourne true

```
const dayOfWeek = 'Samedi';

switch(dayOfWeek) {
  case 'Lundi':
    console.log(`Premier jour de la semaine`);
    break;

  case 'Mardi':
    console.log(`Deuxième jour de la semaine`);
    break;

  case 'Mercredi':
    console.log(`Troisième jour de la semaine`);
    break;

  case 'Jeudi':
    console.log(`Quatrième jour de la semaine`);
    break;

  case 'Vendredi':
    console.log(`Cinquième jour de la semaine`);
    break;

  case 'Samedi':
  case 'Dimanche':
    console.log(`C'est le week-eeeeeend !!!`);
    break;

  default:
    console.log(`Ce jour n'existe pas`);
    break;
}
```

En mettant true, ça permet de mettre une condition à l'intérieur, on cherche dans quel cas on dit "true"

```
const howLongIsYourName = (firstName) => {
    switch(true) {
        case firstName.length <= 4 :
            return `c’est un prénom court`;
            break;

        case firstName.length >= 5 && firstName.length <= 8 :
            return `c’est un prénom moyennement long`;
            break;

        case firstName.length >= 9 && firstName.length <= 15 :
            return `c’est un prénom trèèèèès long`;
            break;

        case firstName.length <= 16 :
            return `c’est un prénom trèèèèès long`;
            break;

        default:
            return `C'est un prénom ça ?`;
            break;
    }
}
console.log(howLongIsYourName(firstName));
```

Tableau - array

C’est une liste

Pour les crochets c’est `alt` + `maj` + `5`

```
const myEmpty = [];

const daysOfWeek = [`lundi`, `mardi`, `mercredi`,
 `jeudi`, `vendredi`, `samedi`, `dimanche`];

daysOfWeek[3] = `Troisème jour de la semaine`;
 //On peut modifier une valeur à l'intérieur même si le tableau est
  // une constante
 //On peut pas modifier un tableau (son nombre de valeur par exemple)

 console.log(daysOfWeek[3]);
```

list

Tableau fourre tout dans lequel on met ce qu'on veut, des tableaux dans des tableaux, des fonctions avec des variables, des entiers, des chaînes…

map

Des dictionnaires, liste d'éléments avec des clefs valeurs.

```
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

```
let map = new Map();

map.set("name", "franck");
map.set("email", "franck@mail.com");
console.log(map);
```

set

Liste d'éléments uniques, on ne peut pas mettre 2 fois le même élément

Méthodes

Appeler une méthode provoque une action sur l'élément sur lequel elle est appelée

Par exemple : `.toUpperCase();` permet de mettre une chaîne en majuscule

`.toFixed(2);` permet d'arrondir un nombre au dessus ou en dessous

`.concat()` pour concatener

etc… il en existe plein.

`.push()` Permet d’ajouter un élément à la dernière position du tableau

```
flowers.push(`Jonquille`)
flowers.push(`Lys`, `Lavande`, `Lila`)
```

`.pop()` Permet de supprimer le dernier élément du tableau

```
flowers.pop();

// Afficher le dernier élément supprimé
lastItem = flowers.pop();
console.log(lastItem);
```

`.join()` crée et renvoie une nouvelle chaîne de caractères en concaténant tous les éléments du tableau

```
const joinedFlowers = flowers.join(`, `);
//Entre parenthèses on mets ce qu'on aimerait bien
//mettre pour séparer chaque éléments

console.log(flowers); // Affiche le tableau
console.log(joinedFlowers); // Affiche Tulipe, Marguerite, Jasmin, Rose, Azalée
```

`.slice()` renvoie un nouveau tableau contenant la copie d’une portion du tableau d’origine. Mais ne change pas la valeur du tableau.

Un ou deux paramètres :

1 - index du 1er element que je veux afficher

2 - index du 1er element que je ne veux pas afficher, a être exclu

```
// slice(start);
// slice(start, end);

flowers.slice(0, 3);
// affiche les éléments index de 0 à 3

flowers.slice(-1);
// affiche le dernier élément du tableau
```

```
flowers.slice(-3);
// affiche les 3 derniers éléments du tableau
```

`.splice()` modifie le contenu d'un tableau en retirant des éléments et/ou en ajoutant de nouveaux éléments à même le tableau.

On peut ainsi vider ou remplacer une partie d'un tableau.

Change la valeur du tableau

```
// .splice(début, nombre d'éléments à supprimer, élément à remplacer)
//éléments à remplacer est optionnel

const months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb');
// inserts at index 1
console.log(months);
// output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// replaces 1 element at index 4
console.log(months);
// output: Array ["Jan", "Feb", "March", "April", "May"]
```

`.shift()` retire le premier élément du tableau

Transforme le tableau

`.unshift("Truc qu'on rajoute")`

Renvoie la longueur du tableau

`.reduce()`

```
const numbers = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = numbers.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue
);
return sumWithInitial;
```

```
// retourne: 10

// ou alors plus simple
function sum(numbers) {
  return numbers.reduce((a, b) => a + b, 0);
}
```

Les boucles

une boucle sans condition de sortie va faire planter le navigateur, retirer l'appel de la fonction et refresh le navigateur

Boucle "for"

```
for (let i = 0; i <= 10; i++) {
  console.log(i);
}

// output : 0, 1, 2, 3 ...

for (let i = 0; i <= 10; i+=3) {
  console.log(i);
// Pour incrémenter de 3 à chaque tour
}
```

for (initialisation ; condition d'arrêt ; action)

i++ ça veut dire i+1

i- - ça veut dire i-1

Sinon i+=3 ou i-=2

```
// Pour sauter 8, affiche tout sauf 8
for (let i = 4 ; i <=10 ; i+=2 ) {
    if(i !== 8) {
        console.log(i);
    }
}

// pour arrêter la boucle
for (let i = 4 ; i <=10 ; i+=2 ) {
```

```
    if(i === 8) {
        break;
    }
    console.log(i);
}
```

Boucle “while”

Signifie “tant que”

```
let counter = 0;

while (counter < 10) {
  counter++;
  console.log(counter);
}

// output : 1, 2, 3, 4 ...
```

let en dehors de la fonction

incrémentation prise en compte tout de suite

Quand utiliser while ?

Qd on ne sait pas combien d’itérations à l’avance

Pour une bouche qu’on veut répéter un nombre de fois indéterminé.

```
const cards = [`carreau`, `pique`, `coeur`, `trèfle`];
let currentCard;
while (currentCard !== `pique`) {
  currentCard = cards[Math.floor(Math.random() * 4)];
  console.log(currentCard);
}
```

Math.random() - Math.floor() - Math.round()

`Math.round(number*100)/100`

Le x100 permet d'avoir la somme en centimes, on divise par 100 pour passer des centimes en euros. C'est mieux d'utiliser Math.round pour arrondir une somme d'argent.

`Math.floor()` : arrondir à l'entier inférieur

`Math.random() * (max - min) + min;`

random : choisir aléatoirement

```
const randomNumber = (min, max) => {
  return Math.floor(Math.random() * (max - min) + min;
}
let entier = randomNumber(1, 10);
```

min et min c'est la même chose qu'il faut écrire

Boucle "do…while…"

Effectuer une tâche au moins une fois, puis continuer jusqu'à ce qu'une condition soit remplie.

```
let money2 = 0;

do {
    gain = Math.floor(Math.random() * (5000 - 0 + 1) + 0);
    lost = Math.floor(Math.random() * (100 - 0 + 1) + 0);
    money = money + gain - lost;
} while (money < 100000);
```

Les itérateurs

Des méthodes qui vont boucler sur un tableau, des méthodes d'itération

`.forEach()` effectue des actions sur chaque éléments du tableau

```
listDeCourse.forEach(mettre une fonction);
listDeCourse.forEach(
    (item, index) => {
        console.log(index, item)
    }
);

//  index et item sont des paramètres,
// on donne le nom qu'on veut et ça renvoie ) ce qu'il y a dans le tableau

listDeCourse.forEach((item, index) => {listDeCourse[index].toUpperCase();});
```

`.map()`

`.filter()` retire les éléments qui ne correspondent pas à la condition

```
const countSheeps = (arrayOfSheep) => {
  return arrayOfSheep.filter(sheep => sheep == true).length;
}

const filtered = listeDeCourse.filter((item) => {
  return item.length > 5;
});
console.log(filtered);
```

`.reduce()`

```
// Additionner tous les nombres d'un tableau
// Ici le tableau c'est numbers[]

function sum (numbers) {
  const initialValue = 0;
  const sumWithInitial = numbers.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue );
  return sumWithInitial;
```

```
};
```

Récupérer le contenu d’un formulaire

```
const username = document.getElementById("usernameInput").value;
```

Inverser des valeurs dans des variables

```
// Méthode rapide
let a = 10;
let b = 20;

b = b + a;
a = b - a;
b = b - a;

// Méthode avec une étape, une variable temporaire
let y = 10;
let z = 20;
let tampon;

tampon = y;
y = z;
z = tampon;
```

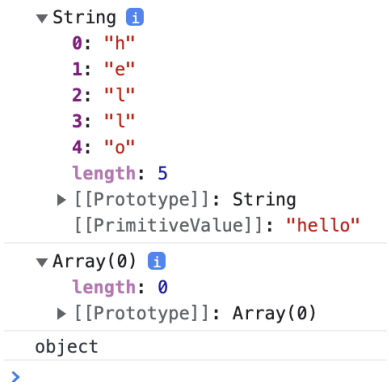
Afficher le type de constructeur

Comme ça on sait si un quelque chose est un objet, un string, un integer…

```
let tab1 = [];
let tab2 = new Array();

let chaine = "hello";
console.log(Object.prototype.constructor(chaine));

console.log(Object.prototype.constructor(tab1));
console.log(typeof(tab2));
```



Objets

```
class User {
  constructor(name) {
}
}
```