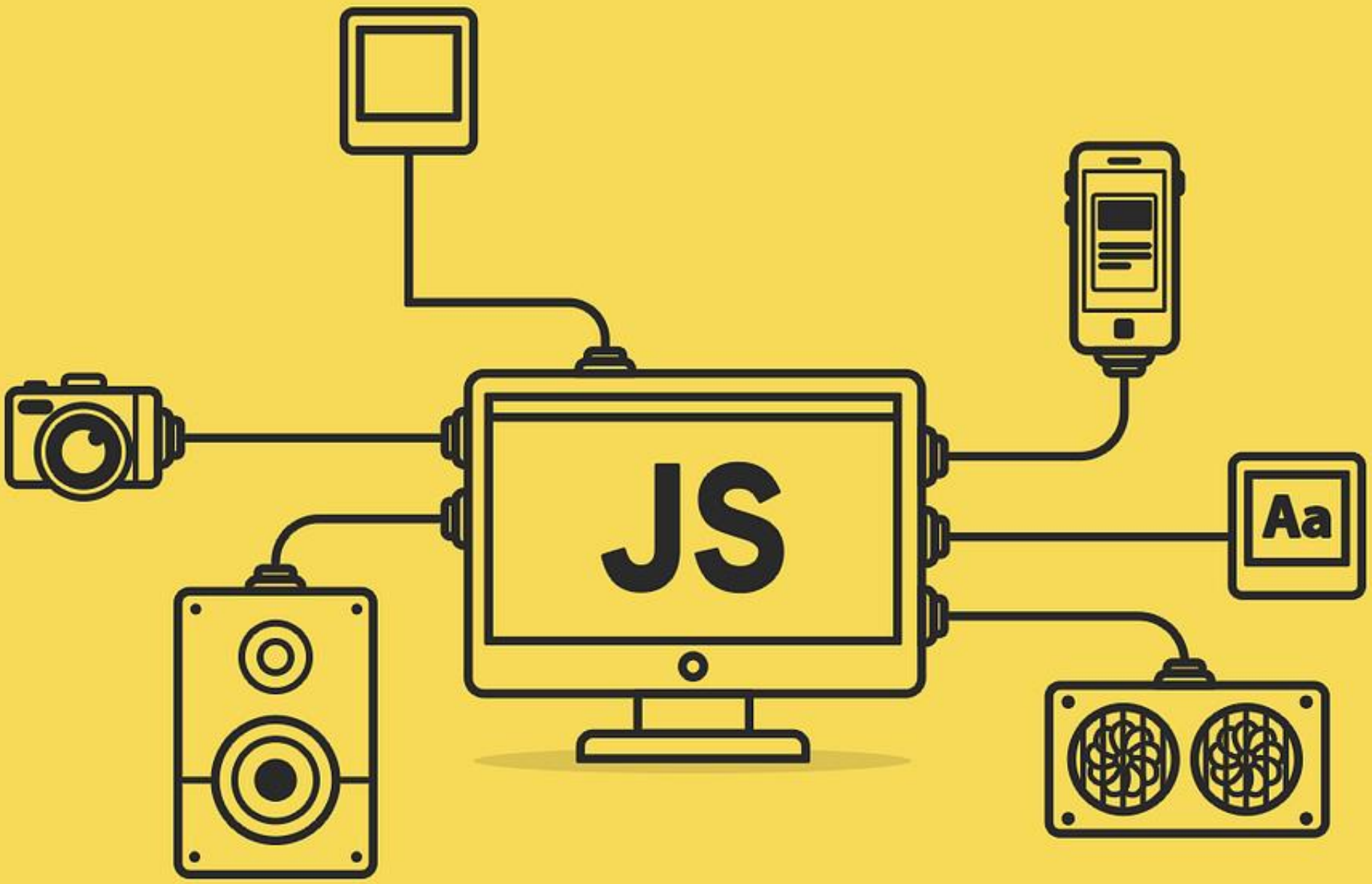

JS
Aa

Rappel des fondamentaux

- Le JavaScript est un langage de programmation de scripts
- Le langage a été créé en dix jours en mai 1995 pour le compte de la Netscape Communications Corporation par Brendan Eich
- JavaScript est aussi employé pour les serveurs Web[6] avec l'utilisation (par exemple) de Node.js , Deno ou récemment Bun.
- Les moteurs JavaScript des navigateur web : https://fr.wikipedia.org/wiki/Moteur_JavaScript
- JavaScript est standardisé sous le nom d'ECMAScript en juin 1997 par Ecma International dans le standard ECMA-262. La version en vigueur de ce standard depuis juin 2022 est la 13e édition.
- C'est un langage orienté objet à prototype : les bases du langage et ses principales interfaces sont fournies par des objets. Cependant, à la différence d'un langage orienté objets à classes, les objets de base ne sont pas des instances de classes
- Les fonctions sont des objets de première classe. Le langage supporte le paradigme objet, impératif et fonctionnel.

Particularités du langage

- En JavaScript, *toutes* les expressions (identifiants, littéraux et opérateurs et leurs opérandes) sont de type référence (comme en Python et Ruby, mais à la différence du C++, Java, C#, Swift et OCaml qui possèdent aussi des expressions de type valeur).
- C'est-à-dire que leur évaluation ne produit pas une donnée directement mais une référence vers une donnée. La référence se nomme le *référent* de l'expression et la donnée le *référé* de l'expression.
- Le JavaScript est un langage dynamique = mise jour sans modification de code mais à l'aide de facteurs externes (Variables, Evènements, DOM et BOM)
- Le JavaScript est un langage (par défaut) côté client = interpréter par un navigateur contrairement a nodeJS, une plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles
- Le JavaScript est un langage interprété = directement par un navigateur qui possède un interpréteur JavaScript
- Le JavaScript est un langage orienté objet. = définition et l'interaction de briques logicielles appelées *objets* *qui* représente un concept, une idée ou toute entité du monde physique

JavaScript et les **Prototypes**

- Les *prototypes* sont des objets utilisés lors d'un échec de résolution de nom, ce mécanisme est un type d'héritage : l'héritage par prototype
- En JavaScript, tout objet possède un prototype, accessible via la méthode : Object.getPrototypeOf (ou __proto__)
- De plus, l'opérateur new permet de transformer l'invocation d'une fonction constructeur en un objet (instanciation) dont le prototype est égal à la propriété prototype de fonction constructeur :
- Toutes instance de creerPersonne(...args) (ici personne_1) = creerPersonne.prototype
- Lors de l'utilisation d'une propriété personne_1.nom : si l'instance ne possède pas la propriété ou la méthode recherchée, la recherche se poursuit dans le prototype de l'instance
- Si la recherche échoue aussi avec cet objet, la recherche se poursuit dans le prototype de cet objet, et ainsi de suite jusqu'à arriver à la première fonction constructeur

```
function creerPersonne(nom, prenom, age, email) {
    return document.write('Nom : ' + nom + ' Prenom : ' + prenom + ' age : ' + age + ' email : ' + email);
}
let personne_1 = new creerPersonne('MICHEL', 'MICHAEL', 35, 'ok@gmail.com');

console.log(personne_1);
console.log(typeof (personne_1));
```

Les types de données en JavaScript

- En JavaScript, contrairement à d'autres langages de programmation, nous n'avons pas besoin de préciser à priori le type de valeur qu'une variable va pouvoir stocker.
- Le JavaScript va en effet automatiquement détecter quel est le type de la valeur stockée dans telle ou telle variable
- En JavaScript, il existe 7 types de valeurs différents. Chaque valeur qu'on va pouvoir créer et manipuler en JavaScript va obligatoirement appartenir à l'un de ces types.
- String : ou « chaine de caractères » en français ;
- Number & BigInt : ou « nombre » en français (entier (integer) et decimaux (float))
- Boolean : ou « booléen » en français (true = 1 et false = 0)
- Null : ou « nul / vide » en français (absence volontaire de valeur) let absence = null;
- Undefined : ou « indéfini » en français (variable déclarée sans valeur assignée) = let vide;
- Symbol : ou « symbole » en français (Le constructeur Symbol() retourne un identifiant unique pour une propriété d'un objet)
- Object : ou « objet » en français (Le constructeur Object() stock des ensembles de clés/valeurs et des entités plus complexes)

Un typage faible et Type Script

- Attention au typage faible de JavaScript : https://foutucode.fr/les-incoherences-de-javascript
- Let test = 1 + ‘’2’’; retourne 12 avec la concaténation alors que let test = 1 + 2; retourne 3 à l’aide de l’opérateur arithmétique
- https://fr.wikipedia.org/wiki/ECMAScript#
- JS suit la norme **ECMAScript**, standard que suivent certains langages de script comme JavaScript. Cette norme évolue en permanence. Les principaux navigateurs Web mettent à jour leur moteur d'exécution pour suivre les évolutions de ce langage.
- Une version majeure d'ECMAScript est celle qui a été définie en 2015 : ES2015 que l'on appelle ES6. Le nom de la version étant déterminé par la dernière version du standard en cours donc ES6 pour 2015. Aujourd'hui la dernière version officielle est ES 12 : EMACScript 2021.
- Le type d'une variable en JS est déterminé lorsqu'on la définit et qu'on lui assigne une valeur particulière. Ce dernier peut changer si on réassigne à la variable une valeur d'un autre type.

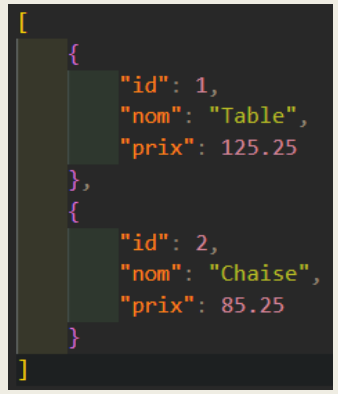
```
let n = 10;
console.log(typeof n); // number

// ré-assignation
n = "Hello";

console.log(typeof n); // string
```

Les types : Objects

```
class Model {

  get() {
    return "table";
  }
}

// Création d'un instance (objet)
const myModel = new Model();

function modelFunc(n) {
  let name = n;

  return name;
}
```

- Rappel : trois paradigmes de programmation
- La programmation procédurale
- La programmation fonctionnelle
- La programmation orientée objet
- Un objet en JavaScript est un conteneur qui va pouvoir stocker plusieurs variables qu'on va appeler ici des propriétés.
- Ils sont mutables, on peut modifier la valeur d'un objet. Un objet est une valeur conservée en mémoire à l'aide d'une référence unique.
- Un objet est donc un conteneur qui va posséder un ensemble de propriétés et de méthodes
- JavaScript est un langage qui intègre l'orienté objet dans sa définition même ce qui fait que tous les éléments du JavaScript vont être des objets

Divers éléments JavaScript



- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects
- Les collections :
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Indexed_collections
- Array.prototype.map()
- https://devdocs.io/javascript/global_objects/array/map
- Un objet Set permet de stocker un ensemble de valeurs uniques de n'importe quel type, qu'il s'agisse de valeurs primitives ou d'objets.
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Set
- https://devdocs.io/javascript/functions/set
- JSON (JavaScript Object Notation) : est un format d'échange de données léger et donc performant.
- C'est un format de texte indépendant de tout langage mais utilisant des conventions familières aux programmeurs de la famille de langages C (incluant JavaScript et Python notamment).
- Une collection de paires nom / valeur. Dans les différentes langages, ce type de structure peut s'appeler objet, enregistrement, dictionnaire, table de hachage, liste à clé ou tableau associatif.
- Une liste ordonnée de valeurs. Dans la plupart des langages, c'est ce qu'on va appeler tableau, liste, vecteur ou séquence.
- https://www.pierre-giraud.com/javascript-apprendre-coder-cours/json/

True – False et short-circuit-evaluations

- Ce qui est faux avec JavaScript : 0, NaN, undefined, false, "", '', \`\`, null
- https://developer.mozilla.org/fr/docs/Glossary/Falsy
- Les short-circuit evaluations = des conditions qui viennent court-circuiter une partie de votre code en fonction des opérandes que vous lui avez indiqué
- Ce type de conditions peut s'avérer utile dans quelques circonstances pour éviter un sempiternel (if - else) et permettre de garder une fluidité dans votre code.

```
//version classique
if (condition) {
  aRetournerSiVrai()
}
//version raccourci
if (condition) aRetournerSiVrai()

//version short-circuit
condition && aRetournerSiVrai()
```

```
const user = "Carlito"
//version classique
let name
if (user) {
  name = user
} else {
  name = "Personne"
}
console.log(name) //affiche Carlito

//version short-circuit
const name = user || "Personne"
console.log(name) //affiche Carlito
```

Les chaînes de caractères :interpolation

- Vous pouvez écrire des chaînes de caractères sur plusieurs lignes et insérer des expressions JS qui seront évaluées à l'aide de backquotes (accent grave ou backticks).

````
Exemple

```js
let a = 51;
let b = 90;
console.log("Somme " + (a + b) + " et\n multiplication " + a * b + ".");
```

Avec les backquotes on aura une expression plus facile à écrire :

```js
let a = 51;
let b = 90;
console.log(`Somme : ${a + b} et \n multiplication : ${a * b}.`);
```
````

- Les conditions if-else dite Ternaire :

```js
//Classique
let age = 18;
if (age > 18) {
    document.write('Vous êtes majeurs');
} else {
    document.write('Vous êtes mineur');
}
//Ternaire
let ternaire = age > 18 ? 'Vous êst majeur' : 'Vous êtes mineur';
console.log(ternaire);
```

```
logged = true ? ( true ? 'toujours yes' : 'no' )  : 'no'; // toujours yes
```

Portée (ou scope en Anglais) des variables en JS

- let ou var ?
- La variable définie avec let a une portée scopée au niveau du bloc dans lequel elle a été déclarée
- Remarque importante : lorsque vous définissez une variable à l'intérieur d'une fonction elle est scopée (portée) dans la fonction elle-même; elle n'a pas d'effet de bord avec le reste du script.
- Si vous définissez une variable de même nom à l'extérieur de la fonction, alors elle n'aura pas d'effet sur la variable définie à l'intérieur de la fonction foo
- JS cherche la définition de ses variables dans le scope courant et sinon il remonte les scopes. Si la variable n'est définie dans aucun des scopes, alors une erreur ReferenceError est levée.

```
let a = 11;

function foo() {
  let a = 10;
  console.log(a);
}

// affiche 10
foo();

// affiche 11
console.log(a);
```

```
// bloc courant pour b = Portée Globale
let b = 11;

function baz() {
  // bloc courant pour c = porté locale de la fonction baz()
  let c = 9;

  // JS ne trouve pas b dans le bloc courant => il remonte les scopes
  console.log(b, c);
}

// affiche 11 9
baz();
```