Opérateurs de comparaison / condition ternaire

Opérateurs de comparaison

Autre concept qui sera surtout utilisé pour les conditions, ce sont les opérateurs de comparaison cela renvoi un booléen (true ou false).

```
/**
 * ************************************
 * 7- Les opérateurs
 * ************************************
 */
//! Les booléens : 2 états possibles TRUE ou FALSE (vrai ou faux)
let a = 11;
let b = 99;
console.log("variable a:",a);
console.log("variable b:",b);
//! avec == on demande si a est égal à b
console.log("c'est égal ? :",a == b);
//!pour vérifier si a est différent de b on utilise !=
console.log("C'est inégal ? :",a != b);
//! Ensuite on retrouve les même opérateurs qu'en Mathématique
//! ici on demande si a est strictement suppérieur à b
console.log("Strictement suppérieur ? :",a > b);
//! ici on demande si a est strictement inférieur à b
console.log("Strictement inférieur ? :",a < b);
//! ici on demande si a est inférieur ou égal à b
console.log("Inférieur ou égal ? :",a <= b);
//! ici on demande si a est suppérieur ou égal à b
console.log("suppérieur ou égal ?:",a >= b);
//? Attention : de base JS ne prend pas en compte le typage des variables :
//? ci dessous le nombre 2 est égal au caractère "2" 🤔
console.log("le chiffre 2 = \"2\"?:",2 == "2");
//! Pour prendre en compte le type des donnée que l'on compare, on utilise l'opérateur
===
//! c'est l'égalité stricte
console.log("égalité stricte ?:",2 === "2");
//! il y a aussi l'inégalité stricte avec l'opérateur !==
console.log("inégalité stricte ?:",2 !== "2");
```

à noter la différence pour vérifier une égalité entre l'opérateur == et ===, le triple égale va permettre de vérifier aussi si les variables comparées sont du même type.

Condition / opérateur ternaire

Une syntaxe pour écrire des conditions simples qui renvoient quelque chose ou quelque chose d'autre si une condition est remplie ou non. Grâce à l'opérateur « ? »
Avant le ? on décrit notre condition, après le ? nous avons ce qui est return quand la condition est remplie puis « : » et ce qui est return quand la condition n'est pas remplie

```
//!-------CONDITIONS TERNAIRES-------
// ? on combine un opérateur de comparaison et l'opérateur ? pour établir une condition qui return une chose ou une autre chose
// ? cela permet de faire une condition if (simple) avec une syntaxe racourcie
let whatIsYourAge = 6;
console.log(whatIsYourAge >18 ? '🍹':'👮');
// Astuce pour check si une variable est définie (si ya QQchose dans son espace mémoire)
let userPremium;
// On check si une variable est définie la condition c'est juste uneVariable ?
console.log(userPremium?'OK 👍':'Not OK 👺');
// ↑ c'est l'équivalent de ↓
console.log(userPremium ==true?'OK 👍':'Not OK 👺');
// on doit lui assigner QQCHOSE
userPremium = 'YES';
console.log(userPremium?'OK 👍':'Not OK 👺');
```

On peut aussi combiner plusieurs conditions avec les opérateurs
|| (une condition OU une autre condition), && (une condition ET une autre condition)

```
// ? On peut utiliser des operateur aussi pour combiner des conditions && (pour ET) || (pour OU)
console.log(3==3&&3<4);
let typeUtilisateur = 'Extra';
console.log(typeUtilisateur == 'Extra' || typeUtilisateur == 'Premium');
```

Condition IF ELSE

Avec if, nous allons pouvoir exécuter du code seulement si une condition est remplie, on peut combiner **if** avec **else** qui correspond à SINON
Dans l'exemple ci-dessous nous avons une fonction qui prend un nombre en paramètre et à l'intérieur de cette fonction nous avons plusieurs conditions :
SI le nombre est supérieur ou = à 30 alors on return une phrase
SINON SI le nombre est inférieur à 10 alors on return une autre phrase
SINON on return une troisième phrase

```
//!-------CONDITION avec IF ELSE-------
// ? Avec if on va pouvoir créer un bloc de code qui s'exécute si une condition est
remplie
function calculTableResto(nombreDeReservation) {
    if (nombreDeReservation>=30){
        return 'il nous reste pas beaucoup de tables, ça serait pour combien de personnes
?';
    }
    else if(nombreDeReservation<10){
        return 'Il nous reste une table'
    }…
    else{
        return 'On est fermé !'
    }
};
console.log(calculTableResto(50));
```

Exercice : If Else

```
//! EXO 7 - IF ELSE
// TODO: Créer une fonction qui reçoit un tableau de 3 notes et qui calcule une moyenne
entre ces 3 notes
// TODO: Dans cette f°, SI la moyenne est supérieure ou égale à 15 on renvoi une string
(très Bien)
// TODO: Dans cette f°, SINON SI la moyenne est supérieure ou égale à 10 on renvoi une
string (assez Bien)
// TODO: Dans cette f°, SINON renvoi une string (refus)
```

Solution

```
function msgMentionBacOfficiel(tabNotes) {
    let moyenneCalc = (tabNotes[0]+tabNotes[1]+tabNotes[2])/tabNotes.length;
    console.log('la Moyenne au Bac : ',moyenneCalc);
    if (moyenneCalc>=16) {
        return "Tu as Gagné !"
    } else if (moyenneCalc >=10 && moyenneCalc<16) {
        return 'Assez Bien'
    } else {
        return 'YO T NUL GRO'
    }
};
console.log(msgMentionBacOfficiel([13,6,3]));
```

Objets

```
// ? syntaxe  { uneProprieté:uneValeur }
// ? dans un objet on assigne avec : plutot qu'avec =
let user = {
    id:3657826,
    'name':'Seagal',
    firstName:'Steven',
    badges:['📽️','👮','🎸','🥋','🎤']
};
console.log(user);
```

On peut accéder aux propriétés d'un objet avec la notation en point

```
console.log(user.name);
console.log(user.id);
```

Ou avec la notation en tableau associatif

```
console.log(user['id']);
```

Pour ajouter une propriété on fait une assignation de valeur (si la propriété existe sa valeur est écrasée par la nouvelle, sinon cela créer la propriété)

On peut également effacer une propriété d'un objet avec delete

```
// ? On peut supprimer une propriété
delete user.badges;
```

```
// ? On peut ajouter simplement des propriétés dans un objet avec une assignation de valeur
// ? Si on assigne à une propriété déjà existante cela écrase la valeur
// ? Mais Si on assigne à une propriété non existante cela créee la propriété
user.dps = 9999;
```

hasOwnProperty

JS propose plusieurs fonctions natives utilisables sur des objets notamment. hasOwnProperty(), qui renvoi true ou false pour vérifier si la propriété d'un objet existe.

Ci-dessous on utilise la fonction dans un console.log() directement.

```
// ? une f° native de JS poour connaitre les propriétés d'un objet, hasOwnProperty()
let menuDuJour={
    entree:"Bassine d'Aioli",
    plat:"Rat Adulte",
    dessert:'île Fidji'
};
console.log(menuDuJour);
console.log(menuDuJour.hasOwnProperty('entree'));
```

```
true                                                    app.js:31
```

Exercice : Objects

```
// ! EXO 8 OBJECTS
// TODO : Faire l'exo avec le User et les passions en mode objet
// (un objet user avec des propriétés pour le nom age et passions qui est lui aussi un objet avec 2 propriétés
```

Solution

```
// TODO : Faire l'exo avec les passions en mode objet
let nameUser = 'Dong Rodrigue';
let ageUser = 65;
let objetUser = {
    'nom' : nameUser,
    'age' : ageUser,
    'passions': {
        passion1:'Le Drift',
        passion2:'Jonquilles'
    }
};
console.log('objet de utilisateur : ',objetUser);
console.log(objetUser.nom);
console.log(objetUser['passions']);// c le taleau passions
console.log(objetUser.passions.passion2);

objetUser.name = 111;
objetUser.age = 'DonDiegoDelavega';
objetUser.passions.passion2 = 'Le Cinéma';
```

Comme pour les tableaux, dans un objet les propriétés peuvent être réassignées à une valeur

https://github.com/jefff404/cours-js/tree/9-objets

Boucle While / For / ForEach / For ...of / For ...in / map

Comme dans tous les langages de programmation Javascript a un système de boucle, de base cela va nous permettre de répéter des instructions de code selon une condition. Les boucles vont également s'avérer utile par la suite, pour parcourir des itérables comme des tableaux ou des objets.

While

Correspond à répéter une ou plusieurs instructions TANT QUE une condition est vraie.

```
let unIndex = 0;
while (unIndex < 10) {
  console.log("Le Nombre : " + unIndex);
  unIndex++;
};
```

Ci-dessus on a un index initialisé à 0 et TANT QUE cet index est strictement inférieur à 10 ALORS, on va faire un console.log(), puis ne pas oublier d'incrémenter l'index pour pouvoir passer à une itération de boucle suivante.

```
Le Nombre : 0    app.js:20
Le Nombre : 1    app.js:20
Le Nombre : 2    app.js:20
Le Nombre : 3    app.js:20
Le Nombre : 4    app.js:20
Le Nombre : 5    app.js:20
Le Nombre : 6    app.js:20
Le Nombre : 7    app.js:20
Le Nombre : 8    app.js:20
Le Nombre : 9    app.js:20
```

FOR

Autre manière de créer des boucles, avec for(), dans les paramètres on va pouvoir directement initialiser un index, définir une condition et incrémenter l'index dans l'exemple ci-dessous nous allons faire une boucle visant à parcourir chaque case d'un tableau pour l'afficher en console.

```
let listeFilm = ['Ultime Décision','Mission Alcatraz','Attack Force'];
//? Boucle for, on définit un index (ici c'est i),
//? puis on définit une condition qui va définir le nombre de fois que le code dans la boucle sera exécutée
//? puis on définit comment on passe à la prochaine itération de la boucle (ici i++, on augmente de 1 le numéro de la case que l'on console.log)
for(i=0;i<listeFilm.length;i++){
    console.log('boucle FOR : ',listeFilm[i]);
};
```

```
boucle FOR :  Ultime Décision     app.js:14
boucle FOR :  Mission Alcatraz    app.js:14
boucle FOR :  Attack Force        app.js:14
```

ForEach

Une autre alternative, la fonction forEach de JS automatise le parcours d'un tableau ou objet (sans que l'on ait à gérer un système d'indexation (i++))
forEach va prendre en paramètre une fonction, cette même fonction pourra avoir un paramètre qui correspondra à chaque case parcourue. (Généralement dans la parenthèse de forEach on passe une fonction fléchée).

```
let listeFilm = ['Ultime Décision','Mission Alcatraz','Attack Force'];
//? La méthode forEach() permet d'exécuter une fonction donnée sur chaque élément du tableau.
// ? On va choisir une variable temporaire pour parcourir les elements du tableau
listeFilm.forEach(unFilm => console.log('boucle forEach Tableau : ',unFilm));
```

Ici chaque case du tableau sera stockée temporairement dans unFilm.

```
boucle forEach Tableau :  Ultime Décision          app.js:27
boucle forEach Tableau :  Mission Alcatraz         app.js:27
boucle forEach Tableau :  Attack Force             app.js:27
```

For … of

Encore une alternative pour parcourir des variables tableaux (et autre) c'est la boucle for … of, qui de la même manière que dans l'exemple précèdent dans lequel on va définir une variable temporaire pour parcourir chaque case du tableau :

```
for(let unElementTablo of listeFilm){
    console.log('boucle FOR...OF : ',unElementTablo);
};
```

```
boucle FOR...OF :  Ultime Décision          app.js:35
boucle FOR...OF :  Mission Alcatraz         app.js:35
boucle FOR...OF :  Attack Force             app.js:35
```

For … in

Si l'on prend le cas des objets JS propose aussi un équivalent à for of (pour les variables de type Array), les boucles for in qui ont exactement la même utilisation que l'exemple précédent

```
const userData = {
    name: 'John Doe',
    email: 'john.doe@example.com',
    age: 25,
    dob: '08/02/1989',
    active: true
};
```

Il faut définir une variable temporaire qui stockera les clés (propriétés) de l'objet

```
// on définit une variable temporaire pour parcourir le objet :)
for(let cleObjet in userData){
    console.log(`boucle FOR...IN (objet) : clé:${cleObjet} - valeur :  $
{userData[cleObjet]} `);
};
```

Ici durant le parcours de l'objet chaque propriété ou clé seront stockées temporairement dans la variable cleObjet.
Rappel : pour accéder aux propriétés d'un objet on la notation en tableau associatif unObjet[quelque chose]

```
boucle FOR...IN (objet) : clé:name - valeur :  John Doe
boucle FOR...IN (objet) : clé:email - valeur :  john.doe@example.com
boucle FOR...IN (objet) : clé:age - valeur :  25
boucle FOR...IN (objet) : clé:dob - valeur :  08/02/1989
boucle FOR...IN (objet) : clé:active - valeur :  true
```

Convertir des objets en tableaux

Depuis sa version ES8 JS propose des fonctions utilisables sur les Objets qui vont pouvoir transformer leurs clés et ou leurs valeurs sous forme de tableau (pour utiliser les $f°$ forEach ou map par exemple).

```
//? Parcourir les Objet  (Depuis javaScript ES8)
//? La Method .keys() qui convertit les clés d'un objet en tableau
//? La Method .values() qui convertit les valeurs d'un objet en tableau
//? La Method .entries() qui renvoit un tableau dans un tableau pour combiner clé - valeur
const keyUser = Object.keys(userData);
console.log("les clé de l'objet converties en array : ",keyUser);

const valuesUser = Object.values(userData);
console.log("les valeur de l'objet converties en array : ",valuesUser);

const convertedDataUser = Object.entries(userData);
console.log("les entrées de l'objet converties en array : ",convertedDataUser);
```

```
// De fait, une fois les objets convertis en tableau  on peut ruser et utiliser forEach par exemple :
valuesUser.forEach((lesValeurs)=>{
    console.log(`FOREACH avec objet converti en tableau chaque valeurs : ${lesValeurs}`);
});
convertedDataUser.forEach(([key, value])=>{
    console.log(`FOREACH avec objet converti en tableau : ${key}: ${value}`);
});
```

Exercice : boucles

```
// TODO :JS map phase 1
// TODO : côté template html rajouter plein de <p></p>
// TODO :On va récupérer TOUS les <p> de notre page dans une
variable lesTxt via getElementsByTagName
// TODO :On va faire un console log de lesTxt
```

```
//TODO JS map Phase 2
//TODO Avec la methode Array.from(), dans une nouvelle variable
textesTab on va transformer notre htmlCollection en array
//TODO On console log la variables textesTab
//* On transforme le HTMLCollection(qui contient tous nos <p>) en
Array classique
```

```
//TODO JS Map Phase 3 (on peut travailler sur un Array)
//TODO Sur textesTab on va utiliser la f° map(),
//TODO dans map(), on va lui passer en param une fonction fléchée
qui elle a en parametre une variable temporaire
(nom de la variable au choix)
//TODO cette fonction fléchée elle va modifier le innerHTML ou
```

Solution

```
const lesTxt = document.body.getElementsByTagName("p");
console.log(lesTxt);
```

```
//* On transforme le HTMLCollection(qui contient tous nos <p>) en
Array classique
const textesTab = Array.from(lesTxt);
console.log(textesTab);
```

```
textesTab.map(uneCase => uneCase.innerHTML = "LOL JE SUIS
HACKERMAN" );
```

i http://127.0.0.1:5500

Les système de boucles

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Totam et rem accusantium dicta voluptatum quam suscipit, molestiae dolor, nihil asperiores corrupti, quisquam ducimus! Sint, ipsa. Voluptates adipisci facilis veniam impedit?

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Totam et rem accusantium dicta voluptatum quam suscipit, molestiae dolor, nihil asperiores corrupti, quisquam ducimus! Sint, ipsa. Voluptates adipisci facilis veniam impedit?

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Totam et rem accusantium dicta voluptatum quam suscipit, molestiae dolor, nihil asperiores corrupti, quisquam ducimus! Sint, ipsa. Voluptates adipisci facilis veniam impedit?

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Totam et rem accusantium dicta voluptatum quam suscipit, molestiae dolor, nihil asperiores corrupti, quisquam ducimus! Sint, ipsa. Voluptates adipisci facilis veniam impedit?

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Totam et rem accusantium dicta voluptatum quam suscipit, molestiae dolor, nihil asperiores corrupti, quisquam ducimus! Sint, ipsa. Voluptates adipisci facilis veniam impedit?

https://github.com/jefff404/cours-js/tree/10-boucle

Point sur var let ou const

En JavaScript de base pour déclarer une variable on utilise le mot « var », mais avec la version ES6 on a eu deux autres manières de déclarer des variables.

(Var cela permet de voir si un tutoriel commence à dater)

En fait var peut poser problèmes avec la notion de scope que l'on a vu précédemment

Rappel scope : Jusqu'où notre variable va être disponible

Avec let et const on gère le scope en fonction des blocs dans lesquels on se situe.
(Scope de bloc)

C'est plus de contraintes mais ça permet de garder une logique et un code plus propre

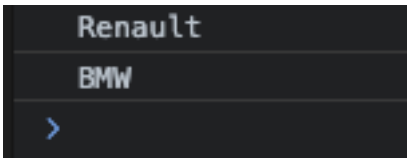
---

Exercice : Quizz Var

Je suis stagiaire dans votre entreprise (sous traitant Nasa) et je vous envoi ce code en « revue de code »
Que me répondez-vous ?

```
var voiture = "Renault";
console.log(voiture);
var voiture = "BMW";
console.log(voiture);
```

(Je vous jure cela fonctionne)



Cela fonctionne mais ce n'est pas correct dans la logique.
JS est peut être un langage assez permissif, mais cela peut engendrer des problèmes.

Exercice : Quizz let

```
//!Quizz : ca bug
console.log(bolide);
let bolide = 'Jaguar'
```

```
Uncaught ReferenceError: Cannot access 'bolide' before initialization                app.js:12
    at app.js:12:13
    (anonymous) @ app.js:12
```

Exercice : Quizz function-var

```
function choixVoiture(){
    var uneVoiture = "Harley Davidson"
}

choixVoiture();
console.log(uneVoiture);
```

```
Uncaught ReferenceError: uneVoiture is not defined                app.js:22
    at app.js:22:13
```

Solution function var

Pour le quizz précédent
Erreur : La var voiture est déclaré au sein de ma fonction et ne peut pas être utilisée en dehors.

```
var uneVoiture = "Harley Davidson"
function choixVoiture(){
}

choixVoiture();
console.log(uneVoiture);
```

Ici ça fonctionne car var est déclaré au-dessus.

Exercice : Quizz if-var

```
var car = "Nissan";

if(car=="Nissan"){
    var vitesse = 800;
}
console.log(vitesse);
```

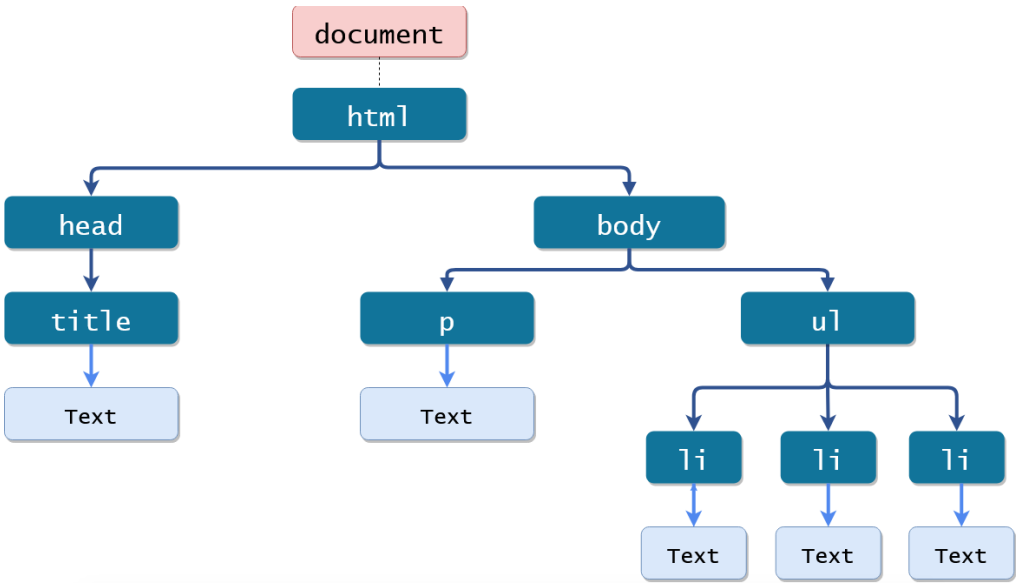
Ca fonctionne, pour vous est-ce normal ?

```
800                                                    app.js:38
```

D.O.M

Présentation

Le DOM est donc une arborescence que va fabriquer le navigateur quand il interprète un fichier HTML, comme cité précédemment en JS tout est objet, et dans le DOM nous allons pouvoir retrouver tous les éléments HTML de notre page sous forme d'objets (ayants des propriétés) manipulables par JS.



Sélectionner des éléments du DOM

getElement(s)By

Une première manière de sélectionner un élément Html (un lien, un titre, une image, … , n'importe quel élément Html).

getElementsByTagName( ) - **getElementsByClassName( )**
permet de sélectionner TOUS les éléments par le nom de leur balise (leur tag) html.

```
//? Une fonction type getElement pour récupérer tous les élément selon une certaine balise dans une HTMLCollection
let tousLesP = document.getElementsByTagName('p');
console.log('La HTMLCollection',tousLesP);
//? Quand on a une HTMLCollection on peut accéder à un certains éléments
console.log('le 3e <p> dans la HTMLCollection : ',tousLesP[2]);
//? Une fonction type getElement pour récupérer tous les élément selon une certaine class dans une HTMLCollection
let tousLesSuper = document.getElementsByClassName('super');
console.log(tousLesSuper);
```

La console du navigateur nous affiche un tableau de type HTMLCollection, un tableau qui va contenir tous les paragraphe <p> de notre page. On a accès à toutes les propriétés de ces éléments Html.
(Son contenu, son alignement, la couleur du texte, sa position dans la page, etc…)
On constate également que ce tableau est indexé (à l'indice [0], le premier paragraphe <p> de la page)

Défi : dans la console, trouver les propriétés dans lesquelles on retrouve ce que l'on a écrit dans les paragraphes.

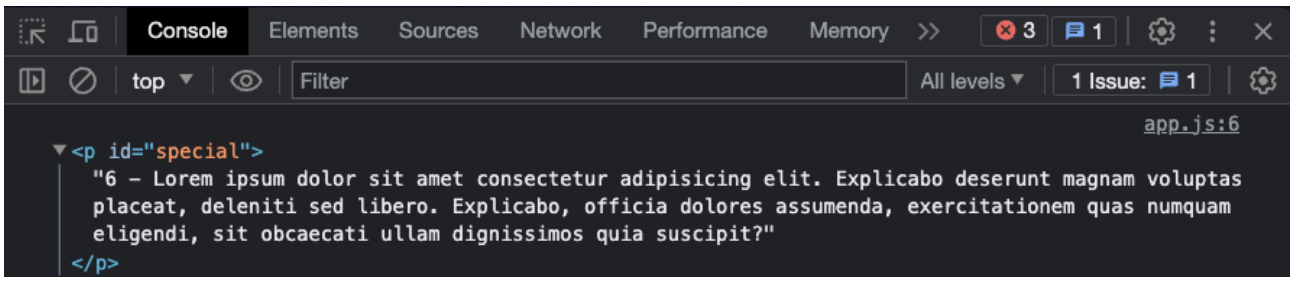
Auteur :
Jean-François Pech
Relu, validé & visé par :
Jérôme CHRETIENNE
Sophie POULAKOS
Mathieu PARIS
Date création :
03/03/2023
Date révision :
10/03/2023

getElementById( )

On peut aussi sélectionner un élément html par son id, on va utiliser getElementById ( ) (récupérer un élément selon le nom de son attribut id)

getElementById () renvoi directement les données elles-mêmes et non pas sous forme de HTMLCollection.
Il n'ya pas de S à Element dans le nom de la fonction, la console nous affichera le code html d'un seul élément en particulier (*en html on n'a qu'un seul id avec un certain nom par page)

```
//? Une fonction type getElement pour récupérer UN élément par son ID
let specialP = document.getElementById('special');
console.log(specialP);
```



QuerySelector( )

Une alternative, les fonctions de type querySelector, qui vont se basées sur la syntaxe de css (rappel : Les selector en CSS)

querySelector( )

```
//? Une fonction type querySelector pour récupérer UN élément (le 1er trouvé)
let lePremierP = document.querySelector('p');
console.log('lePremierP via querySelector : ',lePremierP);
```

⬇️ sélection par l'id (avec # comme en css)

```
//? Une fonction type querySelector pour récupérer UN élément par son ID
let pSpecial = document.querySelector('#special');
console.log('pSpecial querySelector + ID',pSpecial);
```

⬇️ Sélection par la classe (avec . comme en css)

```
//? Une fonction type querySelector pour récupérer UN élément (le 1er trouvé) par sa
classe
let pSuper = document.querySelector('.super');
console.log('pSuper querySelector + class',pSpecial);
```

querySelectorAll( )

Pour récupérer plusieurs éléments (par le nom de balise ou par leurs class css) dans une NodeList

```
//? Une fonction type querySelector pour récupérer TOUS les élément dans une NodeList
let allParagraphes = document.querySelectorAll('p');
console.log('allParagraphes querySelector + balise',allParagraphes);
let allSuper = document.querySelectorAll('.super');
console.log('allSuper querySelector + class',allSuper);
```

```
allSuper querySelector + class ▼NodeList(2) [h2.super, p.super] i
                                  ▶0: h2.super
                                  ▶1: p.super
                                    length: 2
                                  ▶[[Prototype]]: NodeList
```

https://github.com/jefff404/cours-js/tree/20-dom