

# Semaine 4

---

Booléens, conditions et débogage

[Intro. à la programmation](#)



- ❖ Révision
- ❖ Booléens et opérateurs de comparaison
- ❖ Conditions
  - ◆ if, else
  - ◆ else if
- ❖ Opérateurs logiques
  - ◆ && et ||
- ❖ Débogage



## ❖ Semaine 3 :

### ◆ Variables globales et locales

```
function texte1(){
  let phrase = "Natacha n'attache pas son chat";
  document.querySelector(".texte1").textContent = phrase;
}
```

```
function texte2(){
  document.querySelector(".texte2").textContent = phrase;
}
```



```
let gPhrase = "Ces seize chaises sont sèches";
```

```
function texte1(){
  document.querySelector(".texte1").textContent = gPhrase;
}
```

```
function texte2(){
  document.querySelector(".texte2").textContent = gPhrase;
}
```



## ❖ Semaine 3 : ◆ Événements

**Cliquez-moi délicatement**



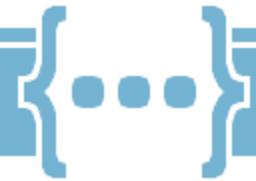
```
document.querySelector(".bouton1").addEventListener("click", changerTexte);
```

```
function changerTexte(){
  document.querySelector(".bouton1").textContent = "Tu as cliqué trop fort 😠";
}
```

**Cliquez-moi délicatement**



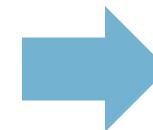
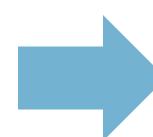
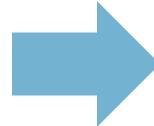
**Tu as cliqué trop fort 😠**



## ❖ Semaine 3 :

### ◆ Modifier les styles du DOM

```
document.querySelector(".classe").style.propriété = "valeur";
```





## ❖ Semaine 3 :

### ◆ currentTarget

```
function init(){
    document.querySelector(".bouton1").addEventListener("click", texteBleu);
    document.querySelector(".bouton2").addEventListener("click", texteBleu);
}
```

```
function texteBleu(event){
    event.currentTarget.style.color = "blue";
}
```





## ❖ Semaine 3 :

### ◆ Fonctions avec paramètres

```
function texteEtBordure(couleurChoisie){  
    document.querySelector(".texte").style.color = couleurChoisie;  
    document.querySelector(".texte").style.borderColor = couleurChoisie;  
}
```

texteEtBordure("pink");

Cet appel rendra le texte rose.

texteEtBordure("crimson");

Cet appel rendra le texte cramoisi.



## ❖ Booléens

- ◆ C'est un autre **type de données**. (Nous avons vu les **nombres entiers**, les **nombres décimaux** et les **chaînes de caractères** pour le moment)
- ◆ Les *booléens* ont seulement 2 valeurs possibles
  - **true**
  - **false**
- ◆ Ils permettent d'exprimer que quelque chose est **vrai** ou **faux**. Exemples :
  - J'ai les yeux bleus ?
  - Je suis majeur ?
  - Mon prénom contient la lettre T ?
  - J'ai déjà utilisé un extincteur ?

Par exemple, **Simone**, qui a les yeux **verts**, qui a **47 ans** et qui n'a jamais assisté à un incendie, on pourrait répondre, dans l'ordre : **false** , **true** , **false** , **false** .



## ❖ Booléens

- ◆ Ce sont des **valeurs** qu'on peut affecter à des **variables**
  - Attention ! Ce ne sont PAS des **chaînes de caractères** !

```
» let a = true;  
    let b = false;  
← undefined
```



```
» let c = "true";  
← undefined
```

- "true" est complètement différent de true.
- "true" est une chaîne de caractères banale.



## ❖ Opérateurs de comparaison

- ◆ Donnent un résultat qui est **true** ou **false**

◆ Plus grand que	<b>&gt;</b>	$5.5 > 6.5$ ( <b>false</b> )
◆ Plus grand ou égal	<b>&gt;=</b>	$5 + 2 >= 5$ ( <b>true</b> )
◆ Plus petit	<b>&lt;</b>	$5 < 7$ ( <b>true</b> )
◆ Plus petit ou égal	<b>&lt;=</b>	$5 <= 7 - 1$ ( <b>true</b> )
◆ Égal	<b>==</b>	$5 - 4 == 7$ ( <b>false</b> )
◆ Pas égal	<b>!=</b>	$5 != 7$ ( <b>true</b> )

```
» 1 < 2
```

```
← true
```

```
» 6.5 >= 6.5
```

```
← true
```

```
» let x = 5 < 3;
```

```
← undefined
```

```
» x
```

```
← false
```

Au lieu d'assigner directement **true** ou **false**, on peut le faire via une vérification comme ceci.



### ❖ Opérateurs de comparaison (Avec des chaînes de caractère)

- ◆ Donnent un résultat qui est **true** ou **false**

- ◆ Égal  $==$  "allo"  $==$  "allo\_" (**false**, car différents)

- ◆ Pas égal  $!=$  "allo"  $!=$  "allo\_" (**true**, car différents)

- ◆ En ce qui concerne  $<$ ,  $<=$ ,  $>$  et  $>=$ , ils évaluent l'ordre alphabétique de deux chaînes de caractères, mais nous n'utiliserons pas ce genre de comparaisons.



## ❖ Priorité des opérateurs

### ◆ Ordre de priorité (Du premier au dernier)

- Parenthèses
- \*, /
- +, -
- <, <=, >, >=, ==, !=
- =

```
>> let b = 4 + 6 == 2 + 8;
```

```
>> let b = 10 == 10;
```

```
>> let b = true;
```

```
>> let a = 4 + 6 > 2 * 3 + 5;
```

```
>> let a = 4 + 6 > 6 + 5;
```

```
>> let a = 10 > 11;
```

```
>> let a = false;
```



 **Booléens**

- ♦ Nous connaissons donc maintenant les « données de type **booléen** » (**true** et **false**) et les **opérateurs de comparaison** **>**, **<**,  **$\geq$** ,  **$\leq$** ,  **$\equiv$** ,  **$\neq$**
  - Exemples

5  $\geq$  5

$$5 < 5$$

"allo" != "allo "



## ❖ Bloc **if**

- ◆ Exécute un morceau de code seulement si une **condition** est respectée
- ◆ **Syntaxe :**

Entre les parenthèses ( ), on retrouve la condition, qui DOIT être un booléen (true ou false)

```
if(/*...Condition...*/)
{
    // Code à exécuter si la condition est « true »
}
```



Entre les accolades { }, on retrouve du code qui s'exécutera SEULEMENT si la **condition** est true.

**Exemple :** Au moment de passer à la caisse dans un magasin... **si** tu as **au moins 2\$** : tu obtiens un paquet de gommes et tu perds 2\$. Si tu n'as pas **au moins 2\$**, il ne se passe rien !



## ❖ Bloc **if**

### ◆ Exemples simplissimes

```
>> if(true){  
    console.log("Allo");  
}
```



Ici, la condition est **true**, alors nous allons exécuter le code dans le bloc.

```
>> if(false){  
    console.log("Allo");  
}
```



Ici, la condition est **false**, alors nous allons simplement sauter (skip) le bloc et passer à la suite sans exécuter son code.

- Bien entendu, on ne met jamais directement **true** ou **false** comme **condition** ! Sinon utiliser un bloc **if** ne sert à rien car on sait d'avance ce qui se produira.



## ❖ Bloc **if**

### ◆ Exemple plus pertinent

- Ici, on écrit "majeur" dans l'élément avec la classe **.statut** seulement si la variable **age** est supérieure ou égale à **18**.
- Sinon, on saute le bloc de code du **if**.

Ce bloc **if** sera exécuté car **age** (19) est supérieur ou égal à 18.

```
let age = 19;
if(age >= 18){
  document.querySelector(".statut").textContent = "Majeur(e)";
}
```

Ce bloc **if** ne sera **pas** exécuté car **animal** ("chien") n'est pas égal à "chat".

```
let animal = "chien";
if(animal == "chat"){
  document.querySelector(".message").textContent = "Miaou";
}
```



## ❖ Conditions incohérentes

- ◆ 😬 ATTENTION ! Ne jamais confondre les opérateurs `==` et `=`

```
>> if(x == 5){  
    console.log("Valide");  
}
```

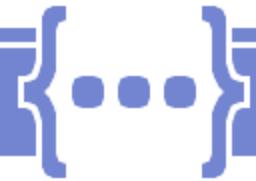


Ici, pas de problème ! On vérifie si `x` vaut 5. La condition est cohérente.

```
>> if(x = 5){  
    console.log("Invalide");  
}
```



Problème : au lieu de poser une condition, on a mis « Je veux affecter 5 à la variable `x` ». Résultat ? C'est toujours considéré comme `true`.



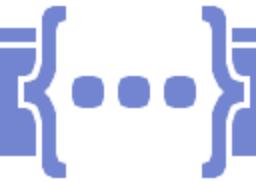
## ❖ Bloc **else**

- ◆ Les blocs **if** peuvent être accompagnés d'un bloc **else**.
- ◆ Syntaxe :

```
if(/*...Condition...*/)
{
    // Code à exécuter si la condition est « true »
}
else
{
    // Code à exécuter si la condition est « false »
}
```

- Le **else** n'est pas suivi d'une **condition**, car il est associé à la même condition que le **if**.
- Le bloc de code sous le **else** s'exécute seulement si la condition est **false**.
  - Ainsi, il y a forcément **un seul des deux** blocs de code qui s'exécutera.

**Exemple :** Si tu m'aides à déménager, je te paye une pizza. Sinon, nous ne serons plus amis. Dans ce cas, le fait d'aider ET de ne pas aider ont tous les deux une conséquence.



## ❖ Bloc **else**

### ◆ Exemples

- ◆ Ici, la condition du **if** est **true**. On va donc seulement exécuter le code du bloc **if**.
  - On écrit le message « Voici ton paquet de gommes » et on retire 2 dans la variable **argent**.

```
let argent = 2.15;
if(argent >= 2)
{
    document.querySelector(".message") .textContent = "Voici ton paquet de gommes";
    argent -= 2;
}
else
{
    document.querySelector(".message") .textContent = "Tu n'as pas assez d'argent.";
}
```



## ❖ Bloc **else**

### ◆ Exemples

- ◆ Ici, la condition du **if** est **false**. On va donc ignorer le code du bloc **if** et exécuter le bloc **else**.
  - On met le texte "Tu n'as pas assez d'argent." à l'élément **.message** et c'est tout !

```
let argent = 1.95;
if(argent >= 2)
{
    document.querySelector(".message") .textContent = "Voici ton paquet de gommes";
    argent -= 2;
}
else
{
    document.querySelector(".message") .textContent = "Tu n'as pas assez d'argent.";
}
```



## ❖ Plusieurs if

- ◆ Quand il y a plusieurs **if**, ils sont totalement indépendants les uns des autres.
  - Chaque **if** est vérifié (et **exécuté** si **true**), peu importe le résultat des **if** précédents.
  - Contrairement à un **else**, qui dépend totalement du **if** qui le précède.

Exécuté car **true** {

Exécuté car **true** {

Pas exécuté car **false** {

```
// temps en minutes
let temps = 21;

if(temps > 15){
  console.log("J'attends patiemment.");
}
if(temps > 20){
  console.log("aaaaaaaaa c'est long 💀");
}
if(temps > 25){
  console.log("dodo 😴");
}
```

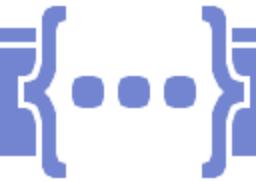


## ❖ Bloc **else if**

- ◆ Permet d'insérer une ou plusieurs conditions alternatives
- ◆ Syntaxe :

```
if(/*...Condition 1...*/)
{
    // Code à exécuter si la condition 1 est « true »
}
else if(/*...Condition 2...*/)
{
    // Code à exécuter si la condition 1 est « false » et la condition 2 est « true »
}
else
{
    // Code à exécuter si les conditions 1 et 2 sont « false »
}
```

Exemple : Si tu es riche, je veux être ton meilleur ami. Sinon, si tu as une belle personnalité, je veux être ton ami. Sinon, je ne veux pas être ton ami.



## ❖ Bloc **else if**

### ◆ Exemple 1

◆ La première condition est **false**. On saute le bloc **if**.



◆ La deuxième condition est **true**. On exécute le bloc **else if** !



◆ On saute le **else** puisqu'un des blocs au-dessus était **true**.



- On met le texte « Grand » à l'élément.

```
let a = 6;
if(a < 3)
{
  document.querySelector(".element") . textContent = "Petit";
}
else if(a > 5)
{
  document.querySelector(".element") . textContent = "Grand";
}
else
{
  document.querySelector(".element") . textContent = "Moyen";
}
```



## ❖ Bloc **else if**

### ◆ Exemple 2

◆ La première condition est **false**. On saute le bloc **if**.



◆ La deuxième condition est **false**. On saute le bloc **else if**.



◆ On exécute le **else** puisque les deux blocs au-dessus était **false**.



- On met le texte « Moyen » à l'élément.

```
let a = 4;
if(a < 3)
{
  document.querySelector(".element") . textContent = "Petit";
}
else if(a > 5)
{
  document.querySelector(".element") . textContent = "Grand";
}
else
{
  document.querySelector(".element") . textContent = "Moyen";
}
```



## ❖ Bloc **else if**

### ◆ Précisions supplémentaires

```
if(...)  
{  
    ...  
}  
else  
{  
    ...  
}
```

On peut mettre autant de **else if** qu'on le souhaite.

```
if(...)  
{  
    ...  
}  
else if(...)  
{  
    ...  
}
```

On n'est pas obligé d'avoir un **else** après le **else if**. Par contre, le **if** est **obligatoire**.

```
if(...)  
{  
    ...  
}  
-----  
if(...)  
{  
    ...  
}  
else if(...)  
{  
    ...  
}
```

Attention ! Quand on a 2 **if** d'affilé, ils sont **totalelement indépendants**.

Par contre, le **else if** est lié au **if** qui le précède.

Ce n'est pas grave si ces précisions ne sont pas claires pour vous pour le moment 😊



## ❖ Opérateurs logiques

- ◆ Permettent de combiner plusieurs expressions de comparaison !

- ◆ ET      **&&**

- Les 2 conditions doivent être **true**

**1 < 2 && 2 > 3**      (**false**, car **2 > 3** n'est pas **true**)

---

```
if(age >= 18 && age < 30)
{
    console.log("Tu es un jeune adulte 🎓");
}
```

Le **if** est exécuté seulement si on a **18 ans ou plus ET moins de 30 ans**.



## ❖ Opérateurs logiques

- ◆ Permettent de combiner plusieurs expressions de comparaison !

- ◆ OU      ||

- Au moins une condition doit être true

1 < 2 || 2 > 3      (true, car 1 < 2 est true)

```
if(age < 4 || age > 99)
{
    console.log("Tu ne peux pas jouer avec des LEGO 😞");
}
```

Si on a moins de 4 ans OU plus de 99 ans, on ne peut pas jouer avec des LEGO 😞





## ❖ Opérateurs logiques

- ◆ Permettent de combiner plusieurs expressions de comparaison !

- ◆ INVERSE              !

- Le booléen est **inversé** (true devient false, false devient true)

**! (1 < 2)**      (**false**, car **1 < 2** était **true**, mais on inverse)

---

```
if( !(age < 4 || age > 99) )
{
  console.log("Tu peux jouer avec des LEGO ! 😊");
}
```

Ce n'est pas grave si vous avez du mal à saisir, mais ici, ça donne : si on a **ni moins de 4 ans, ni plus de 99 ans**, on peut jouer avec des LEGO.



## ❖ Opérateurs logiques

### ◆ Autres exemples

```
» let ageAnne = 21;  
    let ageTom = 19;  
    let ageAli = 18;
```

- Anne est-elle plus la plus vieille ? Autrement dit, Anne est-elle plus vieille que Tom ET plus vieille qu'Ali ?

```
» ageAnne > ageTom && ageAnne > ageAli
```

```
» 21 > 19 && 21 > 18
```

Attention ! On ne doit pas écrire l'expression comme ceci : (Ça ne marche pas)



```
» ageAnne > ageTom && ageAli
```

```
» true && 18
```



## ❖ Opérateurs logiques

### ◆ Autres exemples

```
>> let prixPomme = 2.99;  
     let prixBanane = 1.99;  
     let prixBleuet = 5.99;
```

- Au moins un des trois prix est **plus élevé que 5** ?

```
>> prixPomme > 5 || prixBanane > 5 || prixBleuet > 5
```

```
>> 2.99 > 5 || 1.99 > 5 || 5.99 > 5
```

```
← true
```

Attention ! On ne doit pas écrire l'expression comme ceci : (Ça ne marche pas)



```
>> prixPomme || prixBanane || prixBleuet > 13  
>> 2.99 || 1.99 || true
```



## ❖ Opérateurs logiques

### ◆ Autres exemples

```
» let couleur1 = "rouge";
   let couleur2 = "rouge";
   let couleur3 = "bleu";
```

- Est-ce que les trois couleurs sont identiques ?

```
» couleur1 == couleur2 && couleur2 == couleur3
```

```
» "rouge" == "rouge" && "rouge" == "bleu"
← false
```

Attention ! On ne doit pas écrire l'expression comme ceci : (Ça ne marche pas)



```
» [couleur1 == couleur2] == couleur3
   » [true] == "bleu"
```



## ❖ Débogage

- ◆ Utiliser des stratégies qui permettent de trouver et corriger des « bogues ».
  - Bogue : Défaut de conception ou de réalisation dans un programme.
- ◆ **Visual Studio Code** dispose d'outils de débogage. Toutefois, pour le moment, nous allons apprendre à déboguer avec l'aide de la **console** du navigateur.





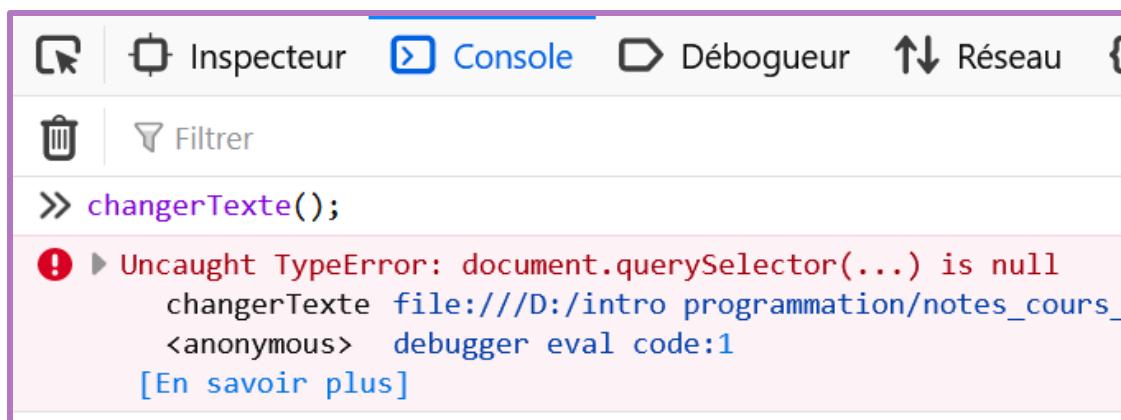
## ❖ Débogage avec la console

### ◆ Exemple 1 : Corriger une fonction simple

```
function changerTexte(){
  document.querySelector("description").textContent = "J'ai trouvé le bug";
}
```

```
<div>
  <h2 class="titre">Débogage</h2>
  <p class="description">Où sont les bogues ? Où ?!</p>
  
</div>
```

- Le but de ma fonction est de changer le texte de l'élément avec la classe `.description`. Je teste donc ma fonction dans la console :



En testant la fonction, on remarque 2 choses :

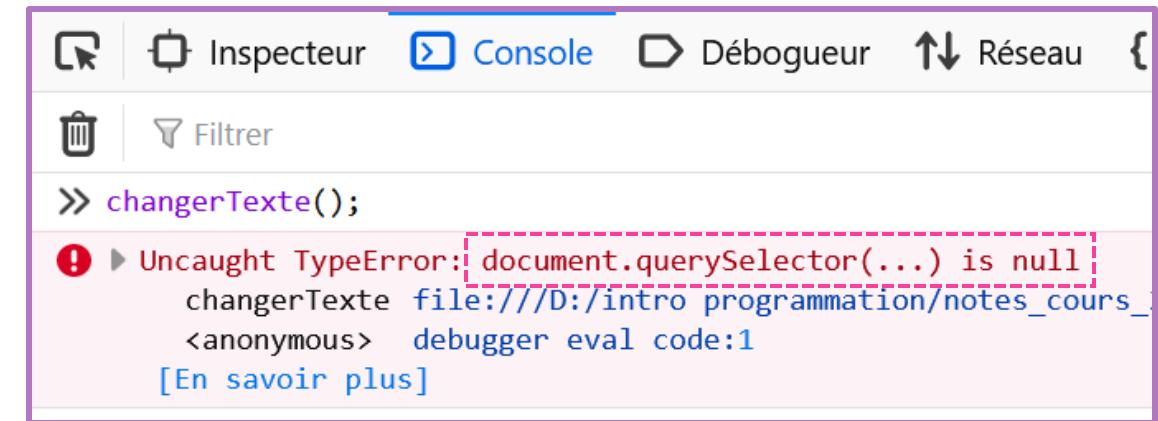
- La fonction n'a pas eu l'effet prévu : aucun texte n'a changé dans la page.
- Un **message rouge** apparaît dans la console.



## ❖ Débogage avec la console

### ◆ Exemple 1 : Corriger une fonction simple (suite)

Hélas, la console nous donne seulement des informations en anglais. C'est souvent le cas en programmation.



The screenshot shows a browser's developer tools interface with the 'Console' tab selected. The console output is as follows:

```
Inspecteur Console Débogueur Réseau {  
  Filtrer  
  >> changerTexte();  
  ! > Uncaught TypeError: document.querySelector(...) is null  
    changerTexte file:///D:/intro programmation/notes_cours_  
    <anonymous> debugger eval code:1  
    [En savoir plus]
```

The error message 'Uncaught TypeError: document.querySelector(...) is null' is highlighted with a red dashed box.

- Malgré tout, on peut deviner que le problème est avec `document.querySelector(...)`
- En particulier, dans ce cas-ci « `document.querySelector(...)` is null » signifie qu'aucun élément n'a été trouvé avec la classe demandée dans la page.



## ❖ Débogage avec la console

### ◆ Exemple 1 : Corriger une fonction simple (suite)

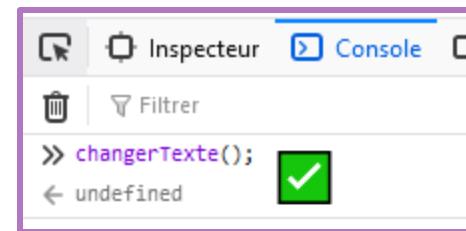
- Il reste donc à vérifier quelle était la classe demandée et la corriger si elle est erronée.

```
function changerTexte(){
  document.querySelector("description")textContent = "J'ai trouvé le bug";
}
```

- En vérifiant l'aide-mémoire, les notes de cours ou d'autres fonctions similaires (qui n'ont pas de bogues), on déduit qu'il manque le . au début de la classe :

```
function changerTexte(){
  document.querySelector(".description").textContent = "J'ai trouvé le bug";
}
```

Bien entendu, on teste à nouveau, car il restait peut-être d'autres bogues. Dans ce cas-ci, il n'y a plus de problème !





## ❖ Exemple 2

### ◆ Deux erreurs dans une fonction

The screenshot shows a browser developer tools console with the 'Console' tab selected. The code in the console is:

```
function changerTexte(){
  document.querySelector(".description").textcontent = "J'ai trouvé le bug";
}
```

The output shows the function being called and an error message:

```
» changerTexte();
! ▶ Uncaught TypeError: document.querySelector is not a function
  changerTexte file:///D:/intro programmation/notes_cours_202
  (anonymous) debuggen_oval_code:1
```

A callout box highlights the error message: "Uncaught TypeError: document.querySelector is not a function".

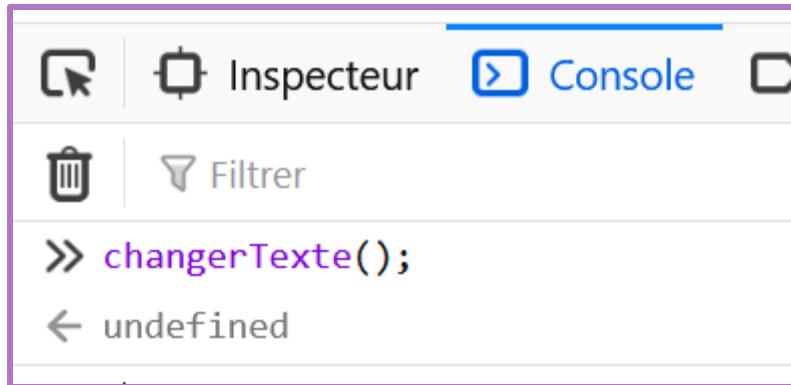
- ◆ Cette fois-ci, on peut comprendre que `document.querySelector` « n'est pas une fonction ». Donc, ça n'existe pas.
  - En vérifiant l'aide-mémoire, les notes de cours ou d'autres lignes de code similaires, on peut remarquer que la bonne orthographe est `document.querySelector`



## ❖ Exemple 2

### ◆ Deux erreurs dans une fonction

- Après avoir corrigé cette première erreur, on teste à nouveau ...



```
function changerTexte(){
  document.querySelector(".description").textcontent = "J'ai trouvé le bug";
}
```



- Il n'y a plus de message d'erreur, mais le texte dans la page n'est toujours pas changé...
  - La console ne peut malheureusement pas détecter tous les bogues !
  - Le problème était avec `.textcontent`, qui s'écrit plutôt `.textContent`. (Vérifiable dans l'aide-mémoire, les notes de cours, etc.)

```
function changerTexte(){
  document.querySelector(".description").textContent = "J'ai trouvé le bug";
}
```

