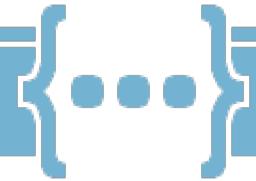


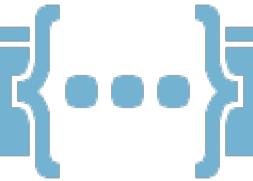
Semaine 4

Booléens, conditions et débogage

Intro. à la programmation - Aut. 2022



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



❖ Semaine 3 :

- ◆ Variables globales / locales et constantes

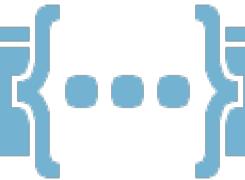
```
const PI = 3.1415;  
  
PI = 3.14; 😬  
PI = PI + 1;
```

```
function titre1(){  
    let texte = "Natacha n'attache pas son chat";  
    document.querySelector("#titre").textContent = texte;  
}  
  
function titre2(){  
    document.querySelector("#soustitre").textContent = texte;  
} 😬
```

```
let gTexte = "Ces seize chaises sont sèches";  
  
function titre1(){  
    document.querySelector("#titre").textContent = gTexte;  
}  
  
function titre2(){  
    document.querySelector("#soustitre").textContent = gTexte;  
}
```

✓

✓



❖ Semaine 3 : ◆ Événements

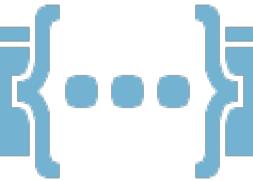
Cliquez-moi délicatement

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

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

Cliquez-moi délicatement

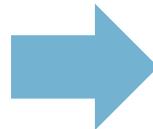
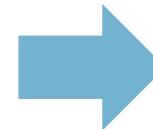
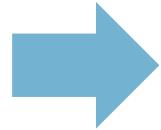
Tu as cliqué trop fort 😡

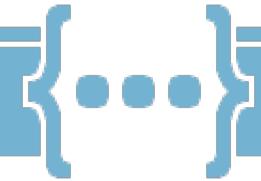


❖ Semaine 3 :

- ◆ Modifier les styles du DOM

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



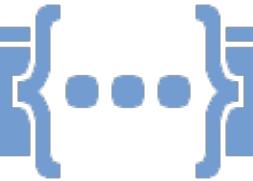


❖ Semaine 3 :

◆ Mot-clé this

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

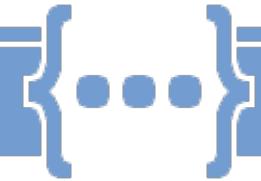
```
function texteBleu(){
    this.style.color = "blue";
}
```



❖ 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	>	5.5 > 6.5 (false)
◆ Plus grand ou égal	>=	5 + 2 >= 5 (true)
◆ Plus petit	<	5 < 7 (true)
◆ Plus petit ou égal	<=	5 <= 7 - 1 (true)
◆ Égal	==	5 - 4 == 7 (false)
◆ Pas égal	!=	5 != 7 (true)

```
» 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**

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



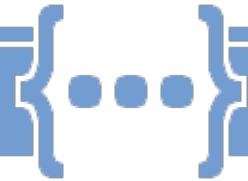
❖ Priorité des opérateurs

◆ Ordre de priorité (Du premier au dernier)

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

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

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



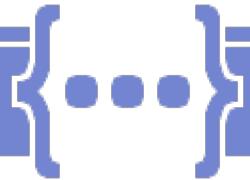
❖ Booléens

- ◆ Nous connaissons donc maintenant les « données de type **booléen** » (**true** et **false**) et les **opérateurs de comparaison** **>**, **<**, **>=**, **<=**, **==**, **!=**
 - Exemples

`5 >= 5` **true**

`5 < 5` **false**

`"allo" != "allo "` **true**



❖ 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**)

- ◆ OU **||**

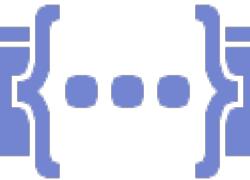
- Au moins une condition doit être true

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

- ◆ INVERSE **!**

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

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



❖ Opérateurs logiques

◆ Exemples concrets



```
let prixPerruche = 5;
```



```
let prixBaudruche = 10;
```



```
let prixRuche = 15;
```

- Le prix d'une **ruche** est à la fois plus élevé que le prix d'une **perruche** **ET** que le prix d'une **baudruche**

`prixRuche > prixPerruche && prixRuche > prixBaudruche`

» `prixRuche > prixPerruche && prixRuche > prixBaudruche`

Attention ! On ne doit pas écrire l'expression comme ceci :



» `prixRuche > prixPerruche && prixBaudruche`



❖ Opérateurs logiques

◆ Exemples concrets



```
let prixPerruche = 5;
```



```
let prixBaudruche = 10;
```



```
let prixRuche = 15;
```

- Au moins un des trois prix est plus élevé que 13\$

```
prixPerruche > 13 || prixBaudruche > 13 || prixRuche > 13
```

```
» prixPerruche > 13 || prixBaudruche > 13 || prixRuche > 13
```

Attention ! On ne doit pas écrire l'expression comme ceci :



```
prixPerruche || prixBaudruche || prixRuche > 13
```



❖ Opérateurs logiques

◆ Exemples concrets



```
let prixPerruche = 5;
```



```
let prixBaudruche = 10;
```

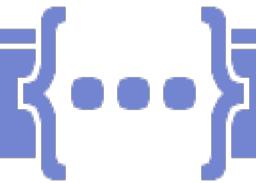


```
let prixRuche = 15;
```

- Le prix d'une perruche 🐦 N'EST PAS égal à 5

!(prixPerruche == 5) ou encore **prixPerruche != 5**

» !(prixPerruche == 5)



❖ Opérateurs logiques

◆ Exemples concrets



```
let prixPerruche = 5;
```



```
let prixBaudruche = 10;
```



```
let prixRuche = 15;
```

- Le prix d'une **ruche** N'EST PAS plus élevé que le prix de 2 **perruches** .

! (prixRuche > 2 * prixPerruche)

```
>> ! ( prixRuche > 2 * prixPerruche)
```



❖ 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 dans la vie : Si tu as au moins 5\$, tu peux faire un tour de montagne russe. Bien entendu, si tu détiens moins de 5\$, il n'y aura tout simplement pas de tour de montagne russe. 😔



❖ Bloc **if**

◆ Exemples simplissimes

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



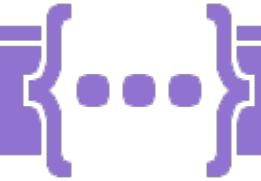
Ici, la condition est **true**, alors nous allons exécuter le code dans le bloc.

```
>> 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 l'id **#statut** seulement si la variable **age** est supérieure ou égale à **18**.
- Sinon, on saute le bloc de code du **if**.

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

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



❖ 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**.



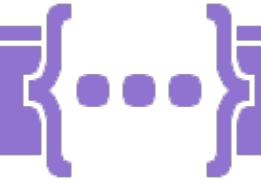
❖ Bloc **else**

- ◆ 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 dans la vie : 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.

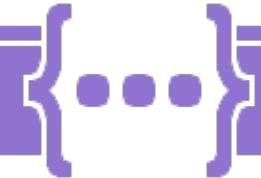


❖ Bloc **else**

◆ Exemples

- ◆ Ici, la condition du **if** est **true**. On va donc seulement exécuter le code du bloc **if**.
 - On met le texte "Enseignant(e)" à l'élément **#personne**.

```
let nom = "Maxime";
if(nom == "Maxime" || nom == "Maude" || nom == "Mathieu")
{
    document.querySelector("#personne").textContent = "Enseignant(e)";
}
else
{
    document.querySelector("#personne").textContent = "Étudiant(e)";
}
```



❖ 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 "Étudiant(e)" à l'élément **#personne**.

```
let nom = "Thérèse";
if(nom == "Maxime" || nom == "Maude" || nom == "Mathieu")
{
    document.querySelector("#personne").textContent = "Enseignant(e)";
}
else
{
    document.querySelector("#personne").textContent = "Étudiant(e)";
}
```



❖ 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 »
}
```

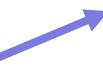
Exemple dans la vie : Si tu es riche, je veux être ton meilleur ami. Sinon, si tu as une belle personnalité, je veux être ton ami. Sinon, je ne veux pas être ton ami.



❖ 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.

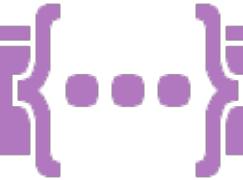
```
let a = 4;
if(a < 3)
{
    document.querySelector("#element") . textContent = "Petit";
}
else if(a > 5)
{
    document.querySelector("#element") . textContent = "Grand";
}
else
{
    document.querySelector("#element") . textContent = "Moyen";
}
```



❖ 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";
}
```

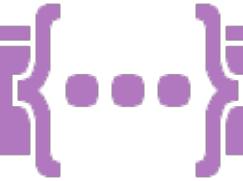
```
<div>
    <h2 id="titre">Débogage</h2>
    <p id="description">Où sont les bogues ? Où ?!</p>
    
</div>
```

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

```
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]
```

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** apparait 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:

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

A red dashed box highlights the error message "Uncaught TypeError: document.querySelector(...) is null".

- 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 l'`id` demandé dans la page.



❖ Débogage avec la console

◆ Exemple 1 : Corriger une fonction simple (suite)

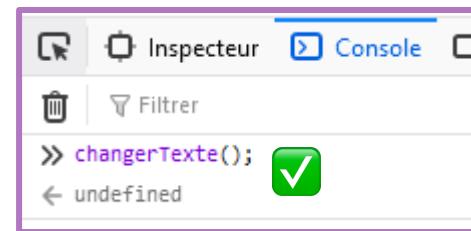
- Il reste donc à vérifier quel était l'id demandé et le corriger s'il est erroné.

```
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 l'id :

```
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's developer tools with the 'Console' tab selected. The code area contains:

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

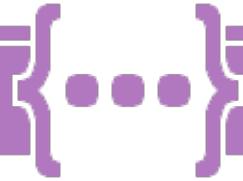
The command area shows:

```
» changerTexte();
```

The output area displays an error message:

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

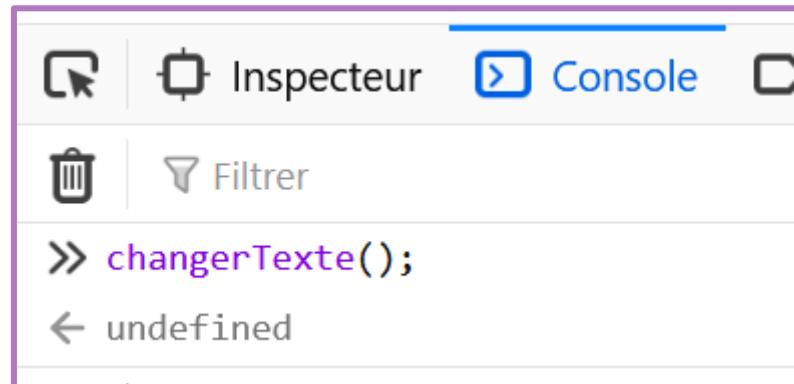
- ◆ Cette fois-ci, on peut comprendre que `document.querySelector` « n'est pas une fonction ». Donc, ça n'existe pas.
 - On vérifiant l'aide-mémoire, les notes de cours ou d'autres lignes de code similaires, on peut remarquer que la bonne orthographe est `document.querySelector`



❖ 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";
}
```

