

Semaine 3

Variables globales, événements, DOM (styles), currentTarget

Intro. à la programmation



- ❖ Révision
- ❖ Variables globales et locales
- ❖ Événements
- ❖ DOM (styles)
- ❖ currentTarget
- ❖ Fonctions avec paramètres



❖ Semaine 1 :

- ◆ Opérateurs mathématiques `+, -, *, /, ()`
- ◆ Opérateurs d'affectation `=, +=, -=`
- ◆ Déclarer une variable : `let a = 3;`

```
>> let x = 3;  
     let x += 1;
```

Combien vaut `x` ? ☺

```
>> let x = 3;  
     x += 2;  
     x = 1;  
     x -= 7;
```

Combien vaut `x` ?

```
>> let x = -2;  
     x += -3  
     x -= -5;
```

Combien vaut `x` ? 🤔



❖ Semaine 2 :

◆ Chaînes de caractères

```
» let energie = "Je m'endors déjà";
```

◆ Concaténation avec +

```
» "sa" + "lut"  
← "salut"
```

◆ Concaténation avec +=

```
» let mot = "per";  
    mot += "ruche";  
← "perruche"
```

◆ Template strings

```
» let nom = "Roland";  
    let objet = "chaises";  
    let phrase = `Je suis ${nom} et j'aime les ${objet}`;  
← undefined  
» phrase  
← "Je suis Roland et j'aime les chaises"
```



- ❖ Semaine 2 (suite)
 - ◆ `document.querySelector()` et `.textContent`

```
<div class="pikachu"> Pick a shoe </div>
```



```
document.querySelector(".pikachu").textContent = "Pikachu";
```

- ◆ `.textContent` et concaténation



The screenshot shows the developer tools console with the following content:

- Toolbar: Inspector, Console, Debugger, Network, Style Editor.
- Console output:
 - Filter Output
 - » `document.querySelector(".titre").textContent += ", le chat consterné";`
 - ← "Smudge, le chat consterné"
 - »





❖ Semaine 2 (suite)

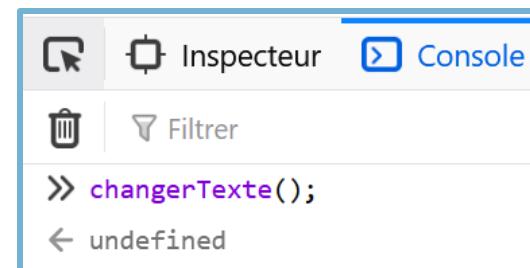
◆ Les fonctions

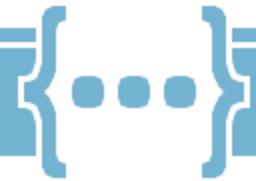
Ce mot-clé sert à déclarer une fonction

Ceci est le nom de la fonction. C'est ce qui l'identifie.

Le morceau de code réutilisable est situé entre des accolades { ... }

```
function changerTexte(){
  document.querySelector(".titre").textContent = "Meme cat";
}
```

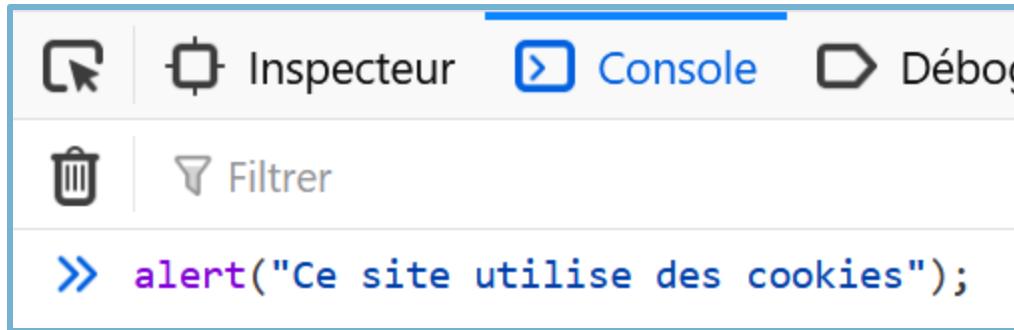




❖ Semaine 2 (suite)

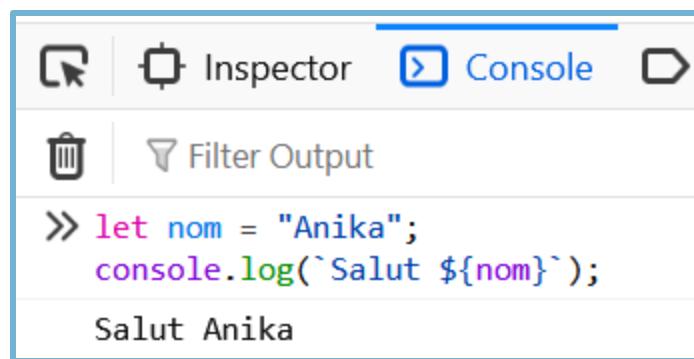
◆ Fonctions préexistantes

- `alert()`

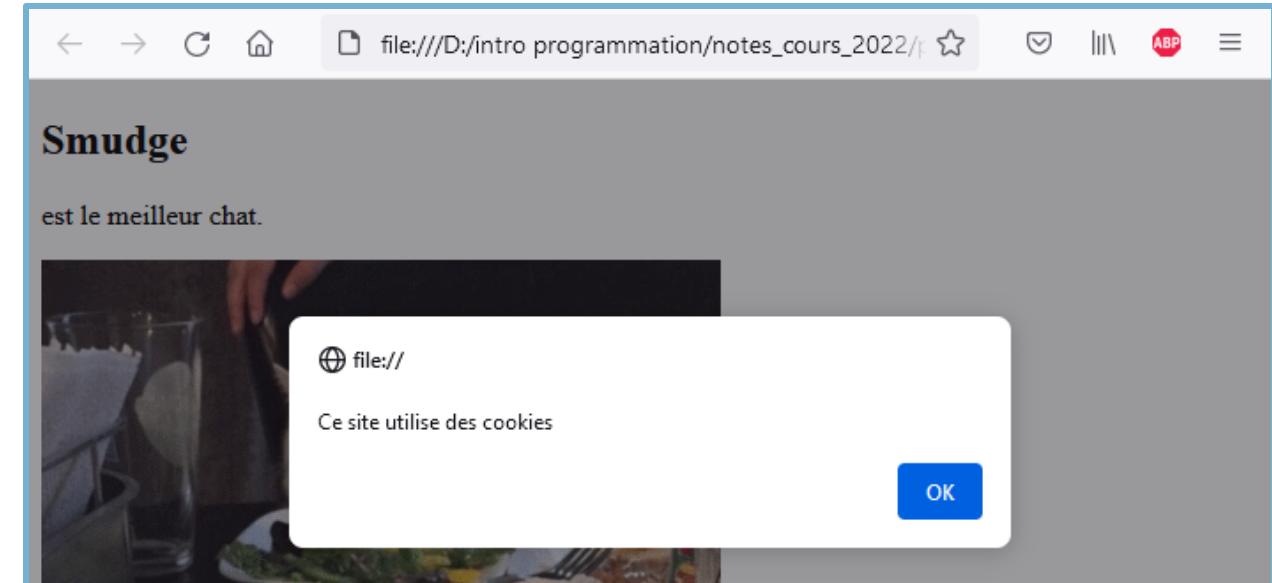


A screenshot of a browser developer tools console. The tabs at the top are 'Inspecteur' (selected), 'Console' (highlighted with a blue border), and 'Débog'. Below the tabs are 'Filtrer' (Filter) and a trash can icon. The console output shows the command: `>> alert("Ce site utilise des cookies");`

- `console.log()`



A screenshot of a browser developer tools console. The tabs at the top are 'Inspector' (selected), 'Console' (highlighted with a blue border), and 'Débog'. Below the tabs are 'Filter Output' and a trash can icon. The console output shows the command: `>> let nom = "Anika";
console.log(`Salut ${nom}`);` and the output: `Salut Anika`





- ❖ Nous avons déjà vu comment déclarer une variable
 - ◆ Ex : `let a = 4;`
- ❖ Toutefois, l'**emplacement** dans le code (dans le fichier `scripts.js`, par exemple) où cette variable est déclarée est important.
 - ◆ La variable n'existe qu'à l'intérieur de la fonction où elle est déclarée. On ne peut pas l'utiliser ailleurs.

La variable `phrase` est déclarée dans la fonction `texte1()`

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

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

On peut utiliser `phrase` ici sans problème.

On ne peut pas réutiliser la variable `phrase` ici, puisqu'on n'est pas dans la fonction `texte1()`. Cela provoque une erreur.



❖ Variables locales

- ◆ Dans cette situation, **phrase** est une variable **locale**. Elle ne peut être utilisée que « localement », c'est-à-dire seulement à l'intérieur de la **fonction** ou du « **bloc** » de code où elle est déclarée.

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

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





❖ Variables globales

- ◆ Une variable dite « **globale** » peut être utilisée n'importe où dans le code.
- ◆ Les variables **globales** doivent être déclarées en dehors de toute fonction, dans n'importe quel fichier JavaScript du projet Web.

- ◆ Ici, la variable **gPhrase** est déclarée en dehors de toute fonction, au début du code.

- ◆ Elle est donc utilisable n'importe où dans les **fonctions** qui suivent.

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

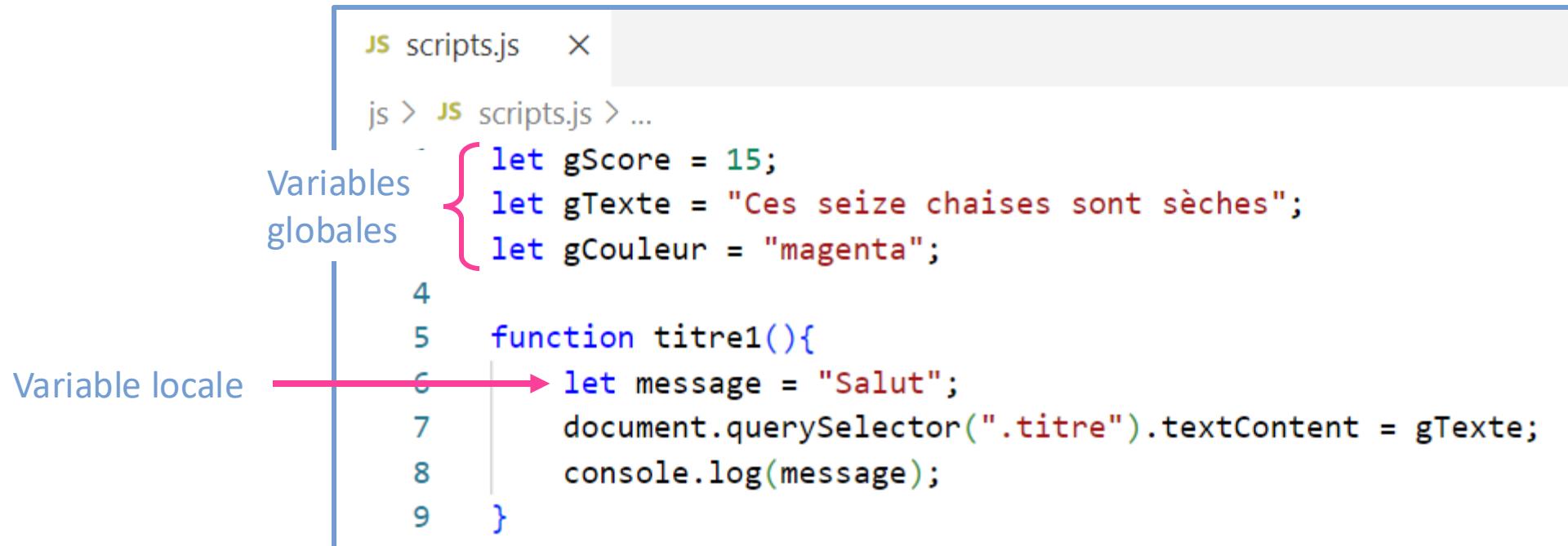
function texte1(){
  document.querySelector(".texte1").textContent = gPhrase;
}

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



❖ Convention de nommage

- ◆ Pour dissiper le doute , nous ajouterons toujours un **g** devant le nom d'une variable globale.
 - De plus, nous déclarerons toujours les variables globales tout en haut du fichier .js



```
JS scripts.js  X
js > JS scripts.js > ...
Variables globales { let gScore = 15;
                     let gTexte = "Ces seize chaises sont sèches";
                     let gCouleur = "magenta";
4
5   function titre1(){
6     let message = "Salut";
7     document.querySelector(".titre").textContent = gTexte;
8     console.log(message);
9   }
```



❖ Écouteurs d'événements

- ◆ Les écouteurs d'événements, permettent, entre autres, d'appeler des **fonctions** suite à la détection d'un **déclencheur**.

- Exemples simples :
 - En **cliquant** sur un élément... son **texte change** !



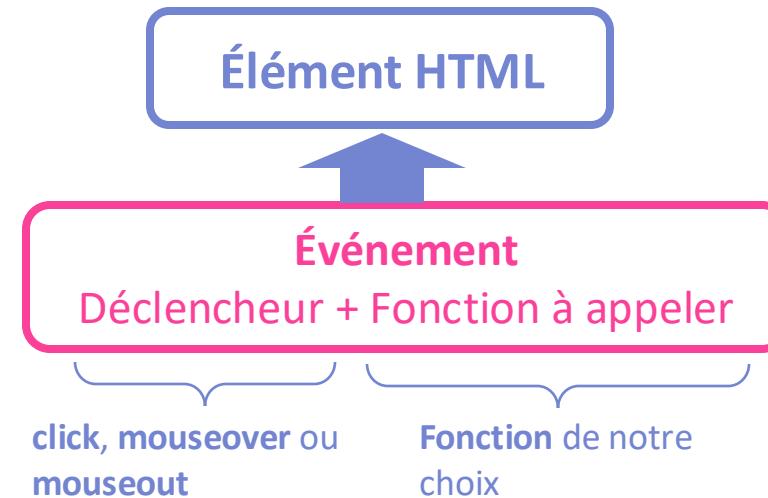
- En **cliquant** sur un élément... une **alerte** apparaît dans la page !





❖ Écouteurs d'événements

- ◆ Les écouteurs d'événements, permettent, entre autres, d'**appeler des fonctions** suite à la détection d'un **déclencheur**.
- ◆ Voici 3 **déclencheurs** :
 - **click** : Appelle une fonction lorsque l'élément HTML **est cliqué**.
 - **mouseover** : Appelle une fonction lorsque l'élément **est survolé**.
 - **mouseout** : Appelle une fonction lorsque l'élément **n'est plus survolé**. (La souris le quitte)





❖ Écouteurs d'événements

◆ Comment ajouter un écouteur d'événements

- Syntaxe :

```
document.querySelector(".classe").addEventListener("type", nom_fonction)
```

Élément associé à l'événement

Déclencheur et fonction

- Exemple

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

Classe de l'élément interactif

Type d'événement Fonction



❖ Écouteurs d'événements

◆ Exemple complet

- On a un **élément** avec la classe "**bouton1**". Il est associé à un **événement** de type « **click** » qui exécute la fonction « **changerTexte()** » lorsque déclenché.

```
<button class="bouton1">Cliquez-moi délicatement</button>
```

```
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 😠



❖ Écouteurs d'événements

◆ Où ajouter les événements

- Dans le cadre du cours, nous placerons toujours les **déclarations d'écouteurs d'événements** dans une fonction nommée **init()**, qui sera automatiquement appelée lorsqu'un projet Web est chargé dans le **navigateur**.

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

- Par exemple, ci-dessus, on peut voir que 2 **écouteurs d'événements** sont déclarés.



❖ Changer un **style** avec DOM

- ◆ Changer un style correspond à modifier le **CSS** d'un élément **HTML**. Pour cela, on utilise la syntaxe `document.querySelector(".classe").style`
- ◆ Nous allons voir comment changer...
 - La **couleur du texte**
 - La **couleur de fond**
 - La **couleur de la bordure**
 - La **largeur de la bordure**
 - La **largeur / la hauteur** de l'élément
 - L'**opacité** d'un élément
 - La **visibilité** d'un élément
 - L'**espacement** depuis la gauche / le haut d'un élément

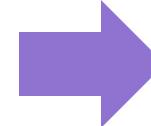


❖ Changer la couleur du texte

- ◆ Syntaxe : `document.querySelector(".classe").style.color = "nom_de_la_couleur";`

```
<h1 class="titre">Chat botté</h1>
```

```
document.querySelector(".titre").style.color = "red";
```

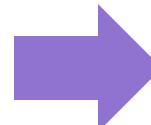


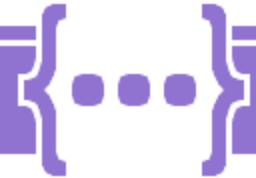
❖ Changer la couleur de fond

- ◆ Syntaxe : `document.querySelector(".classe").style.backgroundColor = "nom_de_la_couleur";`

```
<h1 class="titre">Chat botté</h1>
```

```
document.querySelector(".titre").style.backgroundColor = "lightblue";
```





❖ Changer la couleur de la bordure

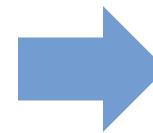
◆ `document.querySelector(".classe").style.borderColor = "nouvelle_couleur";`

❖ Changer la largeur de la bordure

◆ `document.querySelector(".classe").style.borderWidth = "taille_en_pixels";`

```
<div class="boite">
  <h1>Fée marraine</h1>
  
</div>
```

```
document.querySelector(".boite").style.borderColor = "gold";
document.querySelector(".boite").style.borderWidth = "20px";
```

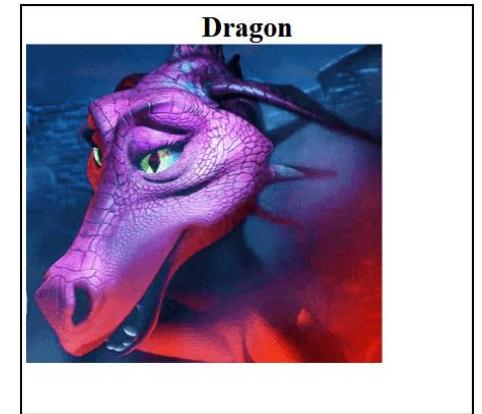
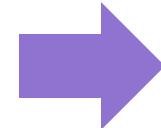




❖ Changer la largeur / hauteur d'un élément

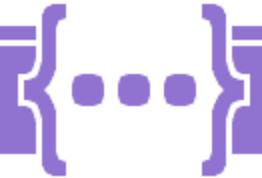
- ◆ `document.querySelector(".classe").style.width = "largeur_en_pixels";`
- ◆ `document.querySelector(".classe").style.height = "hauteur_en_pixels";`

```
document.querySelector(".boite").style.width = "500px";
document.querySelector(".boite").style.height = "450px";
```



❖ Changer la visibilité d'un élément

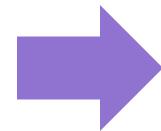
- ◆ `document.querySelector(".classe").style.display = "none";`
 - Permet de masquer l'élément : Il deviendra invisible.
 - Alternativement, les valeurs "block", "inline" et "inline-block" rendront l'élément visible.

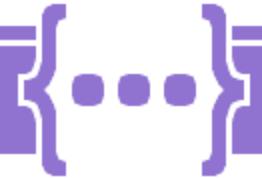


❖ Changer l'opacité d'un élément

- ◆ `document.querySelector(".classe").style.opacity = "0.5";`
- ◆ Valeur de **0** à **1**.
 - **0** -> totalement transparent
 - **1** -> totalement opaque.

```
document.querySelector(".suspect").style.opacity = "0.5";
```





❖ Changer l'espacement à gauche / en haut d'un élément

- ◆ `document.querySelector(".classe").style.left = "taille_en_pixels";`
- ◆ `document.querySelector(".classe").style.top = "taille_en_pixels";`

```

```

```
document.querySelector(".doris").style.left = "200px";  
document.querySelector(".doris").style.top = "50px";
```



On peut voir que l'image s'est éloignée de la **gauche** de **200 pixels** et du **haut** de **50 pixels**.



❖ Plus de **couleurs** s'il vous plaît

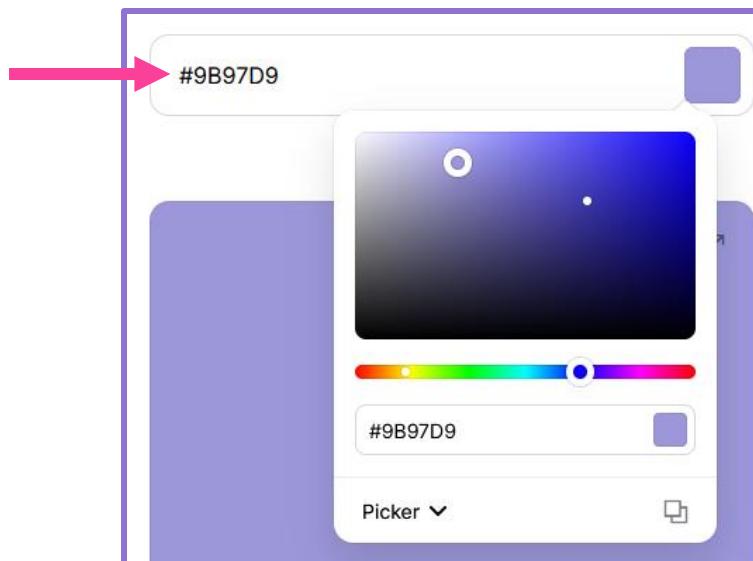
- ◆ Les navigateurs Web connaissent 140 couleurs en lettres comme ceci :

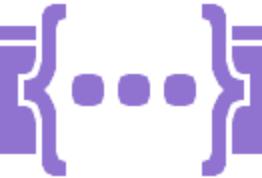
```
document.querySelector(".classe").style.color = "red";
```

- ◆ C'est plutôt limité. Afin de pouvoir utiliser des couleurs personnalisées, on doit utiliser les couleurs « hexadécimales » :

```
document.querySelector(".classe").style.color = "#DC143C";
```

- <https://colors.co/fe0313> Exemple de **roue chromatique** qui nous permet d'obtenir le code **hexadécimal** d'une couleur de notre choix.



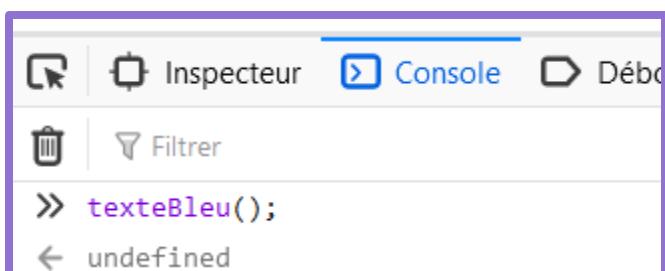


❖ On peut également glisser ces instructions (qui se servent du **DOM**) dans des **fonctions**. (Plutôt que d'écrire ces instructions en entier dans la **console**)

◆ Exemple

```
function texteBleu(){
  document.querySelector(".bouton1").style.color = "blue";
}
```

- Bien entendu, on pourra appeler cette **fonction** dans la **console** ou avec un **événement** par la suite pour l'utiliser !

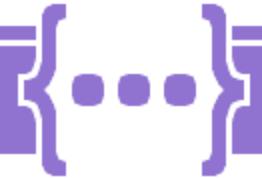


```
<button class="bouton1">Cliquez-moi délicatement</button>
```

Cliquez-moi délicatement



Cliquez-moi délicatement



- ❖ Jusqu'à maintenant, nous avons vu trois modifications que l'on peut faire à un élément HTML :
 - ◆ Lui ajouter un écouteur d'événements
 - ◆ Manipuler son contenu textuel
 - ◆ Modifier un style

```
document.querySelector(".bouton").addEventListener("type", maFonction);
```

```
document.querySelector(".bouton").textContent = "Texte";
```

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



❖ currentTarget

- ◆ Un aspect de la **fonction texteBleu()** est embêtant : cette fonction ne marche que pour l'élément avec la classe « **.bouton1** » !

```
function texteBleu(){
  document.querySelector(".bouton1").style.color = "blue";
}
```

- On ne peut pas utiliser la fonction pour un autre élément.
 - Une solution pourrait être de créer une fonction pour chaque élément dont le fond peut changer de couleur... mais cela implique de la **répétition de code** ! 😢

```
function texteBleu(){
  document.querySelector(".bouton1").style.color = "blue";
}

function texteBleu(){
  document.querySelector(".bouton2").style.color = "blue";
}
```



❖ currentTarget

- ◆ C'est ici qu'une *propriété* nommée « **currentTarget** » est pratique.
 - Si on remplace **document.querySelector(".class")** dans la fonction **texteBleu()** par **event.currentTarget**, c'est automatiquement **l'élément HTML qui appelle la fonction** (suite au déclenchement d'un événement) qui sera affecté par la **fonction**.
 - Pour que **event.currentTarget** fonctionne, il y a un **prérequis** : ajouter « **event** » dans les **parenthèses** de la fonction.

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



- Donc présentement, cette fonction marchera pour tous les éléments pour lesquels nous avons configuré un **événement** qui appelle la fonction « **texteBleu()** ».



❖ currentTarget

◆ Exemple

```
function init(){

    document.querySelector(".bouton1").addEventListener("click", texteBleu);
    document.querySelector(".bouton2").addEventListener("click", texteBleu);

}
```

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

- Si on clique sur l'élément avec la classe `.bouton1`, la fonction `texteBleu` rendra le texte de l'élément `.bouton1` bleu.
- Si on clique sur l'élément avec la classe `.bouton2`, la fonction `texteBleu` rendra le texte de l'élément `.bouton2` bleu.

currentTarget



❖ Attention !

- ◆ `currentTarget` peut seulement remplacer `document.querySelector` quand c'est l'élément avec lequel on interagit qu'on souhaite modifier.

Ex : Si je souhaite que **cliquer sur `.bouton1` change la couleur du texte de `.texte1`**, je ne peux pas utiliser `currentTarget`, car cela modifierait le `.bouton1` à la place de `.texte1` ...

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

function test(){
  document.querySelector(".texte1").style.color = "limegreen";
}
```

Élément interactif (`.bouton1`) est différent de l'élément modifié (`.texte1`) ... donc on ne peut pas utiliser `currentTarget` !

currentTarget



❖ currentTarget

- ◆ **currentTarget** peut être utilisé avec **.textContent** et **.style**.propriété

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

function test(event){
  event.currentTarget.style.color = "limegreen";
  event.currentTarget.style.backgroundColor = "black";
  event.currentTarget.textContent = "Tu m'as cliqué 😊";
}
```

- ◆ Notez bien que **event.currentTarget** ne peut jamais être utilisé dans la fonction **init()** !



❖ Fonctions sans paramètre et avec paramètres

Fonctions sans paramètre

```
Math.random();  
styleRouge();
```

Fonctions avec paramètre(s)

```
alert("Salut !");  
console.log(maVariable);
```

Lorsqu'une **fonction** contient une ou plusieurs **données** dans ses **parenthèses** lorsqu'elle est appelée, on dit que c'est une **fonction avec paramètre(s)**.

Quand on appelle **alert(...)** ou **console.log(...)**, il faut préciser du **texte** (ou une **variable** qui contient du texte) dans les **parenthèses**. Quand on fait ça, on dit qu'on « envoie un **paramètre** ». (On envoie une information à la fonction pour qu'elle l'utilise)

Fonctions avec paramètres



❖ Déclarer une fonction avec paramètre(s)

- ◆ Dans les **parenthèses**, il faut ajouter un ou plusieurs « **paramètres** »

Déclaration de la fonction

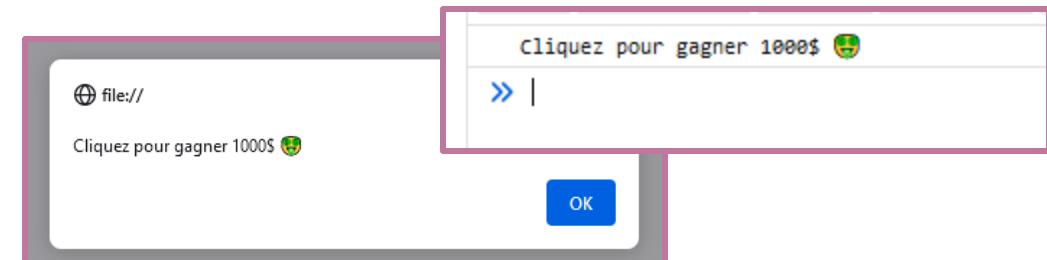
```
function alerteEtConsole(message){  
  alert(message);  
  console.log(message);  
}
```

message est le nom du **paramètre** de la fonction. La valeur qui est donnée à **message** lorsque la fonction est **appelée** va déterminer l'alerte et l'affichage dans la console.

Appel de la fonction

```
alerteEtConsole("Cliquez pour gagner 1000$ 🎉");
```

Ici, on appelle la fonction **alerteEtConsole(...)** et on donne la valeur "Cliquez pour gagner 1000\$ 🎉" au paramètre **message**. Cela signifie que c'est cette phrase qui apparaîtra dans l'**alerte** et dans la **console**.





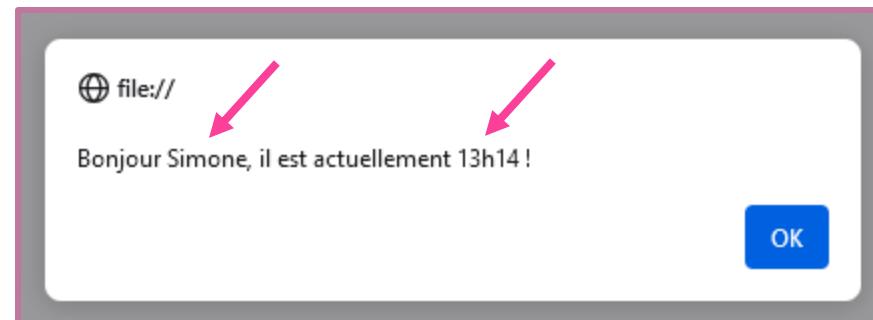
❖ Un paramètre, c'est un contrat

```
function salutations(nom, heure){  
  alert(`Bonjour ${nom}, il est actuellement ${heure} !`);  
}
```

- ◆ On peut voir que la fonction **salutations()** possède **deux paramètres** : **nom** et **heure**.
 - La fonction nous indique : « pour m'utiliser, tu dois me fournir **deux informations** : un **nom** et une **heure** ».
 - La fonction va ensuite utiliser ces deux informations pour compléter un message.

```
>> salutations("Simone", "13h14");
```

On donne la valeur "**Simone**" pour le paramètre **nom** et la valeur "**13h14**" pour le paramètre **heure**.



L'alerte affiche le message complété



❖ Pourquoi des paramètres ?

- ◆ Exemple, ces trois fonctions permettent de changer la **couleur de texte et de bordure** de l'élément `.texte` :

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

function texteEtBordureBleu(){
    document.querySelector(".texte").style.color = "blue";
    document.querySelector(".texte").style.borderColor = "blue";
}

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



❖ Pourquoi des paramètres ?

- ◆ Même exemple, mais on utilise un **paramètre** pour choisir la couleur :

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

Pas besoin de créer une fonction par couleur !

```
texteEtBordure("pink");
```

Cet appel rendra le texte rose.

```
texteEtBordure("crimson");
```

Cet appel rendra le texte cramoisi.



❖ Pourquoi des paramètres ?

- ◆ Même exemple, mais on utilise **deux paramètres** 🧑‍🚀 : un pour choisir la **couleur** et un pour choisir la **classe** de l'élément à modifier !

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

Pas besoin de créer une fonction par élément et par couleur !

```
texteEtBordure(".description", "crimson");
```

```
texteEtBordure(".texte", "pink");
```



❖ Paramètres : valeurs et variables

- ◆ Quand on passe des paramètres à une fonction, on peut le faire en utilisant directement des **valeurs** (nombres, chaînes de caractères, ...) ou des **variables**.

```
function colorierTexte(couleur){  
  document.querySelector(".paragraphe2").style.color = couleur;  
}
```

Ici, on écrit la **valeur** directement en paramètre en appelant la fonction **colorierTexte()**

```
colorierTexte("blue");
```

Ici, on utilise une **variable** en appelant la fonction **colorierTexte()**. C'est comme si on avait directement mis "red" en paramètre.

```
let teinte = "red";  
  
colorierTexte(teinte);
```