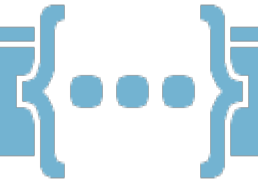
# Cours 3

## DOM avec styles CSS, variables globales et événements

**Intro. à la programmation - Aut. 2021**

# Menu du jour



- Révision
- DOM
  - Styles CSS
  - Fonctions pour changer les styles CSS
- Variables locales / globales
- Événements
- Mot-clé *this*

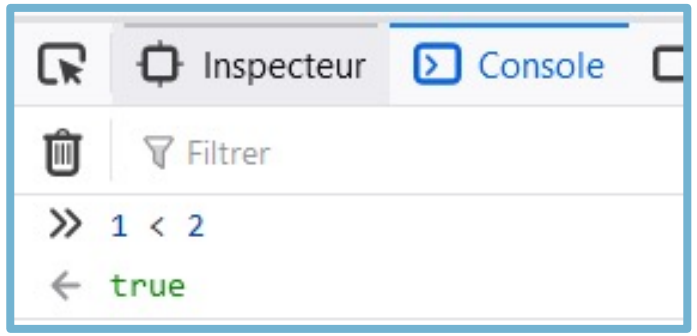
# Révision

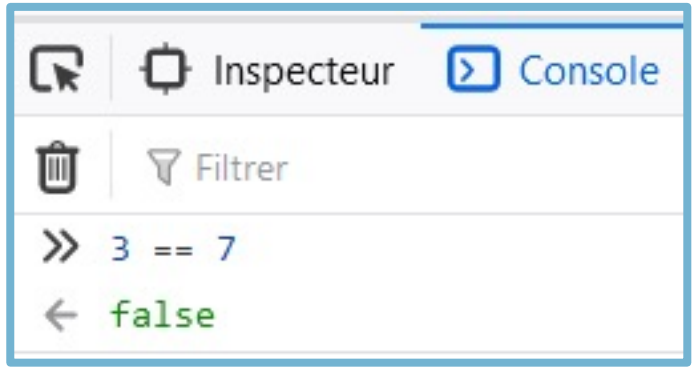
- Semaine 1 :
  - Déclaration de variable `let nomVariable = "valeur";`
  - Affectation (`=`, `+=`, `-=`, `++`, `--`)
  - Opérateurs arithmétiques (`+`, `-`, `*`, `/`, `%`)
  - Concaténation `"Salut " + "Janine." ;`
- Semaine 2 :
  - Booléens (`true`, `false`)
  - Opérateurs de comparaison (`<`, `>`, `<=`, `>=`, `==`, `!=`)
  - Opérateurs logiques (`&&`, `||`, `!`)
  - Instructions conditionnelles (`if`, `else`, `else if`)
  - Fonctions
  - DOM `document.getElementById("id").textContent`
  - WebStorm

## ❖ Opérateurs de comparaison

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

| | | |
|---|---|---|
| Plus grand que | > | 5.5 > 6.5 (false) |
| Plus grand ou égal | >= | 5 >= 5 (true) |
| Plus petit | < | 5 < 7 (true) |
| Plus petit ou égal | <= | 5 <= 7 (true) |
| Égal | == | 5 == 7 (false) |
| Pas égal | != | 5 != 7 (true) |



# ❖ Opérateurs logiques

- ♦ Permettent de combiner plusieurs expressions de comparaison !

- ♦ AND `&&`
  - ○ Les 2 conditions **doivent** être true

```
1 < 2 && 2 > 3
```

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

- ♦ OR* `||`
  - ○ **Au moins une** condition doit être true

```
1 < 2 || 2 > 3
```

(**true**, car **1 < 2** est true)

- ♦ NOT `!`
  - ○ Le booléen est inversé (true devient false, false devient true)

```
!(1 < 2)
```

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

*|| représente deux barres verticales, et non deux L minuscules.

## ❖ if, else, else if

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

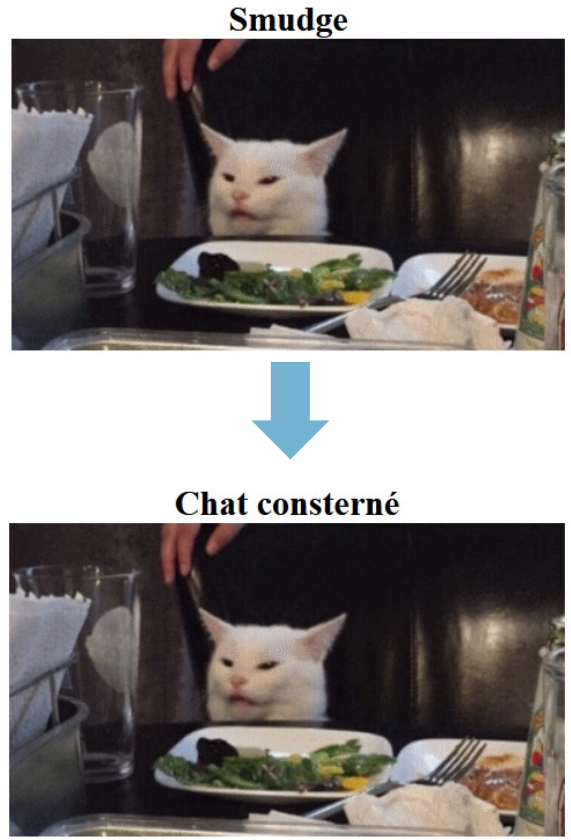
```
if(/*...Condition...*/)
{
    // Code à exécuter si la condition est true
}
else
{
    // Code à exécuter si la condition est false
}
```

```
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
}
```

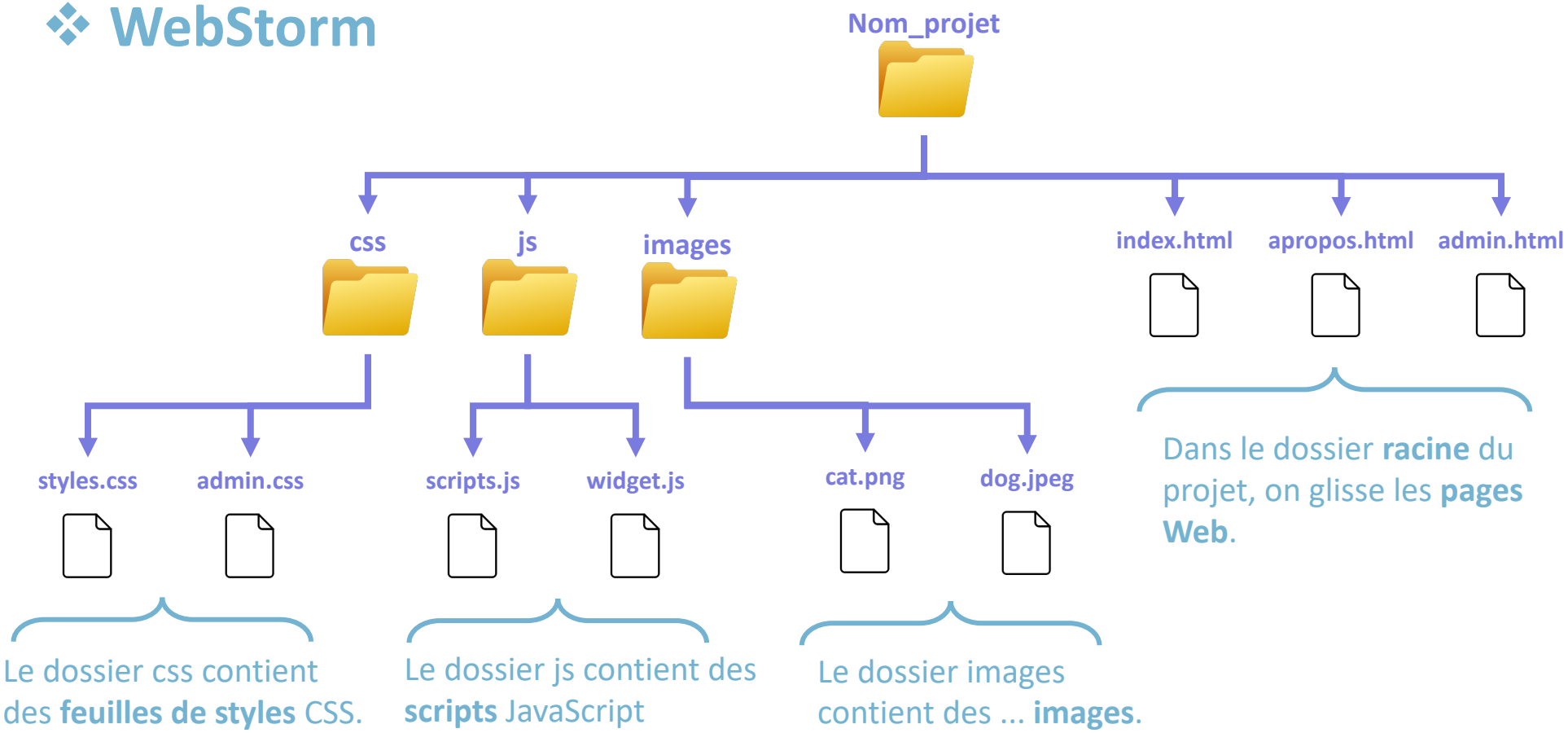
# ❖ DOM

## ♦ Modifier du texte

```
<h2 id="title">Smudge</h2>
<img src="images/smudge.png" alt="Smudge">
```

## ❖ WebStorm

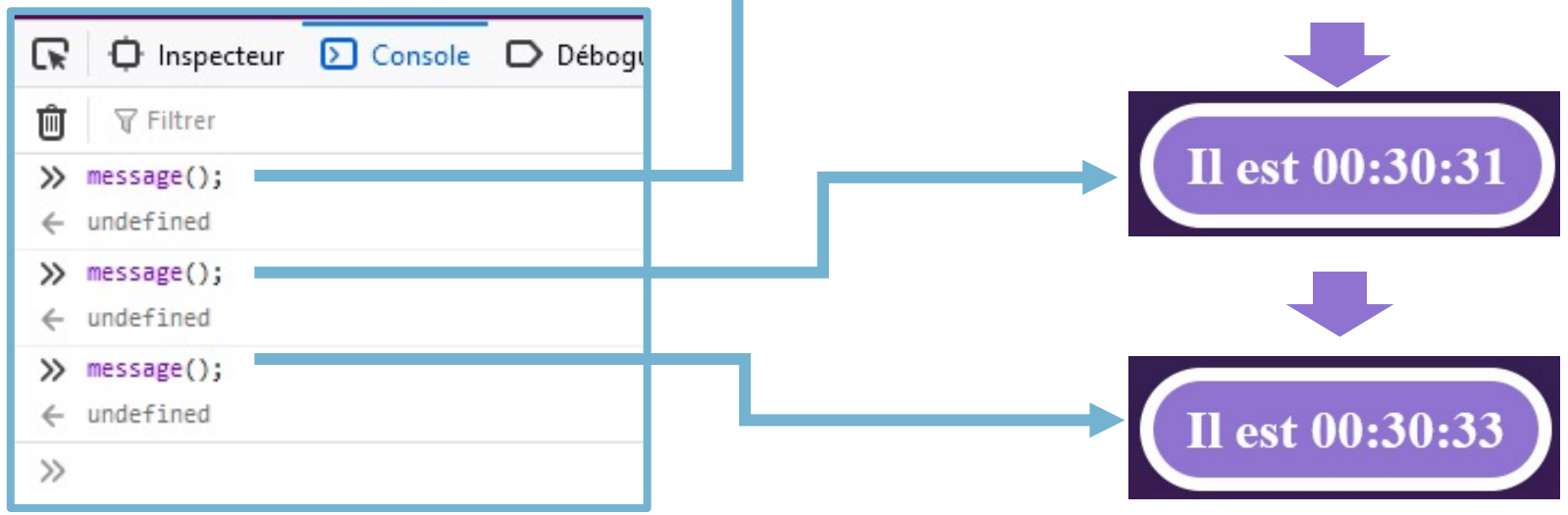

Nom_projet
css
js
images
index.html
apropos.html
admin.html
styles.css
admin.css
scripts.js
widget.js
cat.png
dog.jpeg
Dans le dossier **racine** du projet, on glisse les **pages Web**.
Le dossier css contient des **feuilles de styles** CSS.
Le dossier js contient des **scripts** JavaScript
Le dossier images contient des ... **images**.

## ❖ Fonctions

```
function message(){
    document.getElementById("log").textContent = "Il est " + new Date().toTimeString().slice(0,8);
}
```

```
<div id="log">Il est 19:55:47.</div>
```

Il est 19:55:47.



## ❖ Changer un **style** avec DOM

- ♦ Changer un style correspond à modifier le **CSS** d'un élément **HTML**. Pour cela, on utilise la syntaxe `document.getElementById("id").style`
- ♦ Nous allons voir comment changer...
  - ○ La **couleur du texte**
  - ○ La **couleur de fond**
  - ○ La **taille du texte**
  - ○ L'**alignement du texte**
  - ○ La **couleur de la bordure**
  - ○ La **largeur de la bordure**
  - ○ La **largeur** / la **hauteur** de l'**élément**
  - ○ La **visibilité** d'un **élément**

## ❖ Changer la couleur du texte

♦ Syntaxe : `document.getElementById("id").style.color = "nom_de_la_couleur";`

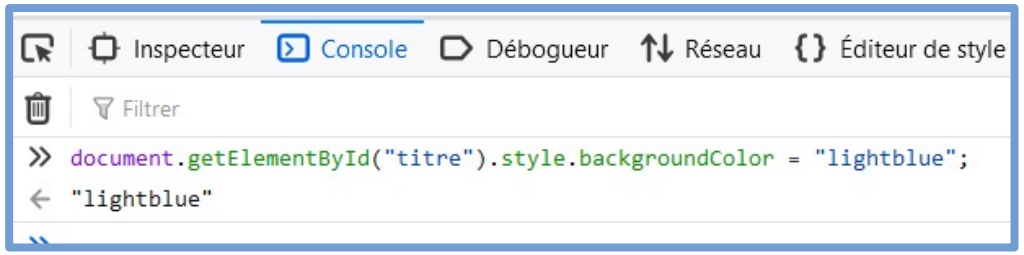
```
<h1 id="titre">Chat botté</h1>
```

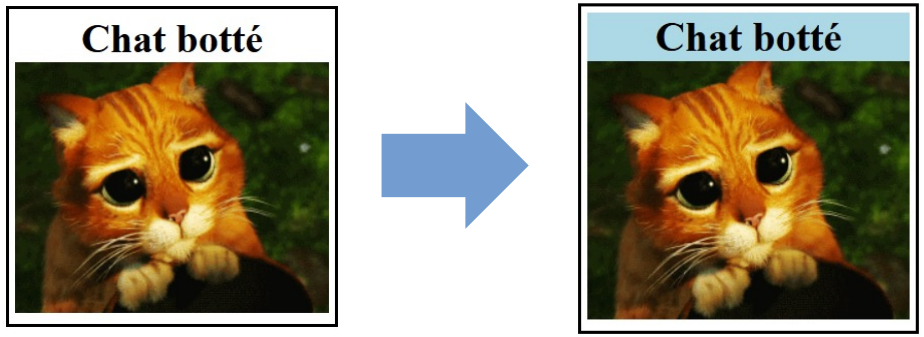


## ❖ Changer la couleur de fond

♦ Syntaxe : `document.getElementById("id").style.backgroundColor = "nom_de_la_couleur";`

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



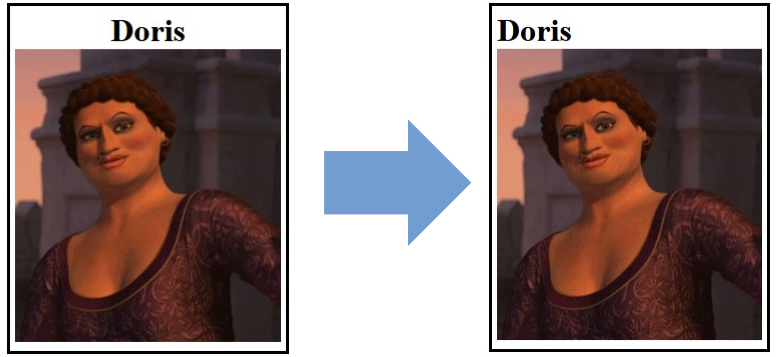
## Changer la taille du texte

- `document.getElementById("id").style.fontSize = "taille_en_pixels";`
  - Exemple : **20px**, **10px**, **40px**, etc.



## Changer l'alignement du texte

- `document.getElementById("id").style.textAlign = "nouvel_alignement";`
  - Les alignements possibles sont **left**, **right**, **center** et **justify**

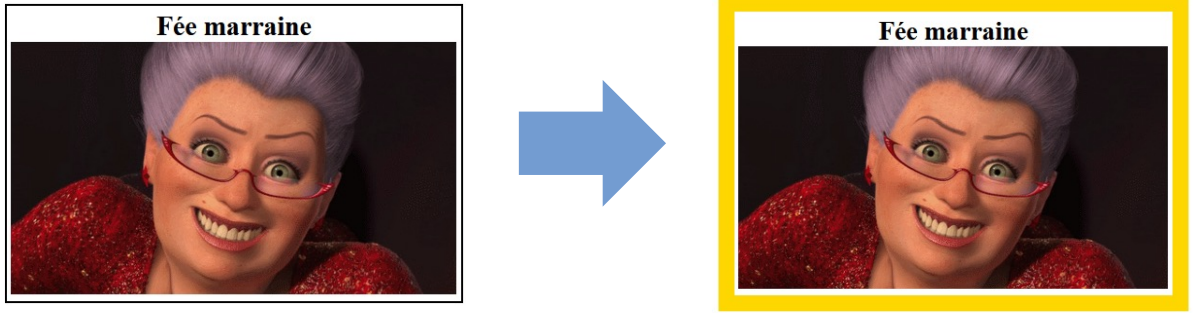
## Changer la couleur de la bordure

- `document.getElementById("id").style.borderColor = "nouvelle_couleur";`

## Changer la largeur de la bordure

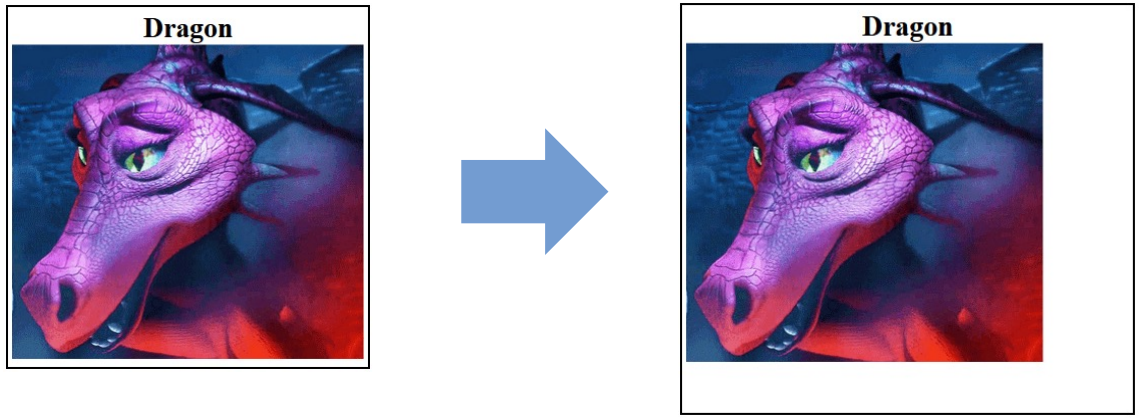
- `document.getElementById("id").style.borderWidth = "taille_en_pixels";`

```
<div id="boite">
    <h1 id="titre">Fée marraine</h1>
    <img id="image" src="images/fairy.png" alt="Fée marraine">
</div>
```

## ❖ Changer la largeur / hauteur d'un élément

- `document.getElementById("id").style.width = "largeur_en_pixels";`
- `document.getElementById("id").style.height = "hauteur_en_pixels";`

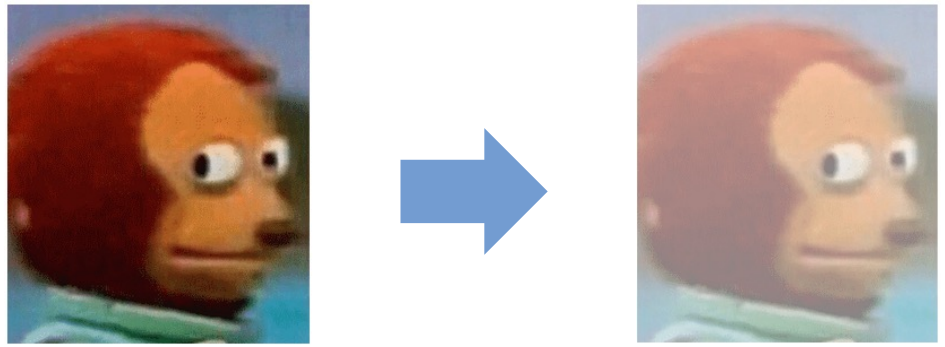




## ❖ Changer la visibilité d'un élément

- `document.getElementById("id").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.getElementById("id").style.opacity = "0.5";`
- ♦ Valeur de **0** à **1**. **0** étant totalement transparent et **1** totalement opaque.

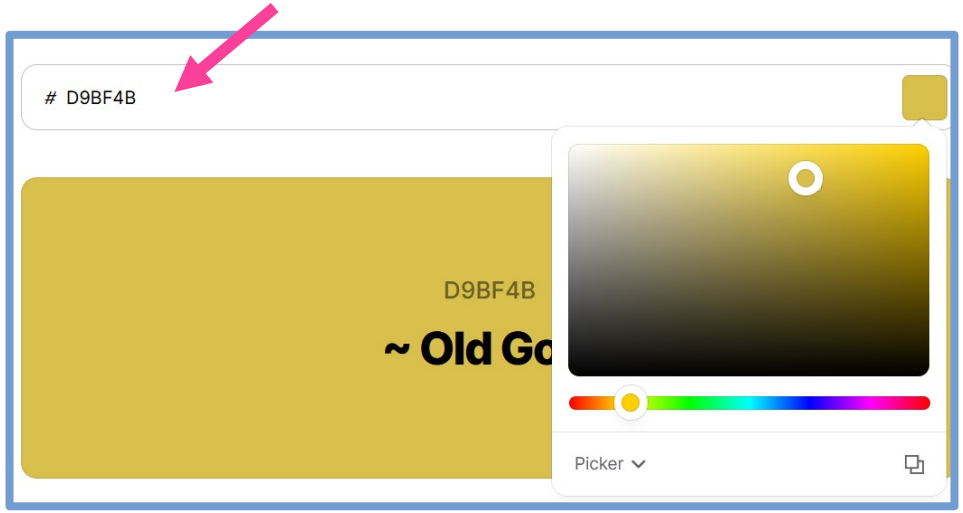
- Plus de **couleurs** s'il vous plait 🎨
  - Les navigateurs Web connaissent 140 couleurs en lettres comme ceci :

```
document.getElementById("id").style.color = "red";
```

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

```
document.getElementById("id").style.color = "#DC143C";
```

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

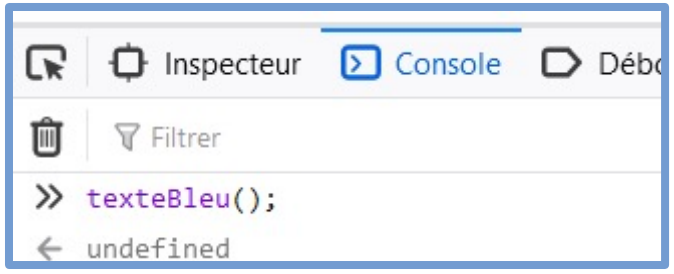


- On peut également glisser ces instructions (qui se servent du **DOM**) dans des **fonctions**. (Plutôt que d'écrire ces instructions en entier dans la **console**)
  - Exemple

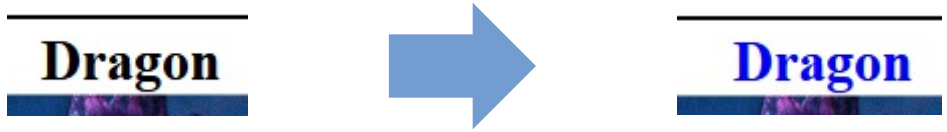
script.js | index.html

```
1  function texteBleu(){
2      document.getElementById( elementId: "titre").style.color = "blue";
3  }
```

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




```
<h1 id="titre">Dragon</h1>
```



# Variables globales

- Nous avons déjà vu comment déclarer une variable
  - Ex : `let a = 4;`
- Toutefois, l'**emplacement** dans le code où cette variable est **déclarée** est important.
  - Exemple : Déclarer une variable dans une **fonction**
    - La variable **n'existe qu'à l'intérieur de la fonction**. On ne peut pas l'utiliser ailleurs.

La variable « **couleur** » est déclarée dans la fonction **`texteBleu()`**.

On peut l'appeler juste ici sans problème.

```
function texteBleu(){
    let couleur = "blue";
    document.getElementById("titre").style.color = couleur;
}

function fondBleu(){
    document.getElementById("titre").style.backgroundColor = couleur;
}
```

On ne peut pas réutiliser la variable « **couleur** » ici, puisqu'on n'est plus dans la fonction **`texteBleu()`**. Cela provoque une erreur.

# Variables globales

## ❖ Variables **locales**

- ♦ Dans cette situation, « **couleur** » 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.

La variable « **couleur** » est déclaré dans la fonction **`texteBleu()`**.

On peut l'appeler juste ici sans problème.

```
function texteBleu(){
    let couleur = "blue";
    document.getElementById("titre").style.color = couleur;
}

function fondBleu(){
    document.getElementById("titre").style.backgroundColor = couleur;
}
```

On ne peut pas réutiliser la variable « **couleur** » ici, puisqu'on n'est plus dans la fonction **`texteBleu()`**. Cela provoque une erreur.

# Variables globales

## ❖ Variables **globales**

- ♦ Une variable dite « **globale** » (ou encore « avec une **portée globale** ») peut être utilisée **n'importe où** dans un « programme » JavaScript.

- ♦ Les variables **globales** doivent être déclarées en dehors de toute fonction, dans n'importe quel fichier JavaScript du projet Web.

- ♦ Ici, la variable « **couleur** » est déclarée en dehors de toute fonction, au début du code **js**.

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

index.html | script.js

js > script.js > ...

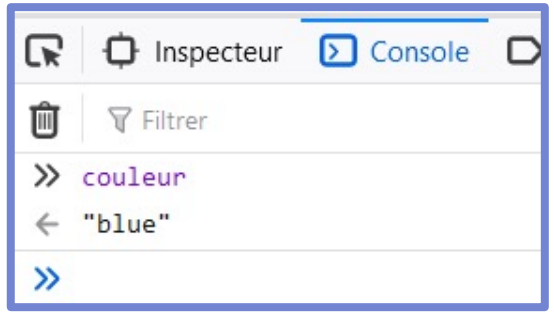
```
1  let couleur = "blue";
2
3  function texteBleu(){
4      document.getElementById("titre").style.color = couleur;
5  }
6
7  function fondBleu(){
8      document.getElementById("titre").style.backgroundColor = couleur;
9  }
```

# Variables globales

## ❖ Variables **globales**

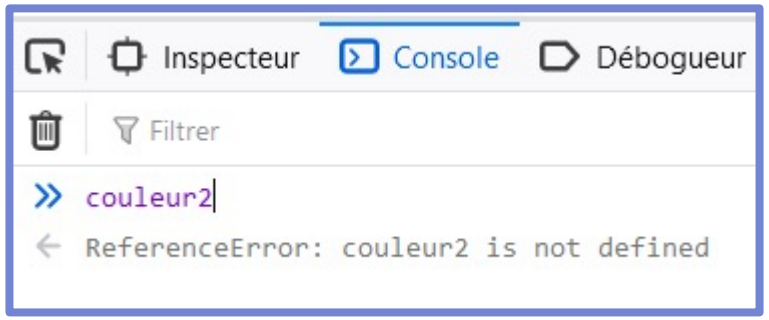
- ♦ Si on ouvre un projet Web dans notre navigateur, on peut ouvrir la **console** et constater que dès le démarrage de la page Web, la **console** connait les **variables globales**.

```
script.js
1   let couleur = "blue";
```



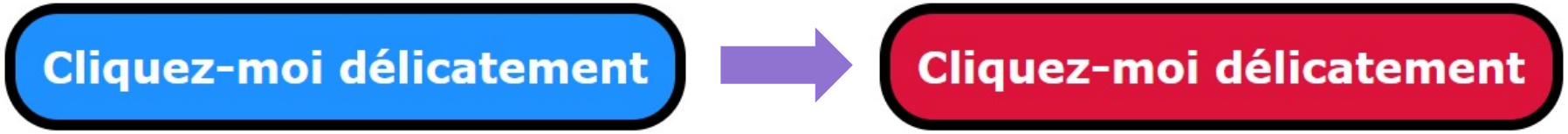

- ♦ Alors que « **couleur2** », qui est une variable **locale** (car déclarée dans une fonction), la **console** ne la connait pas !

```
function texteBleu(){
    let couleur2 = "red"
    document.getElementById( elemer
}
```

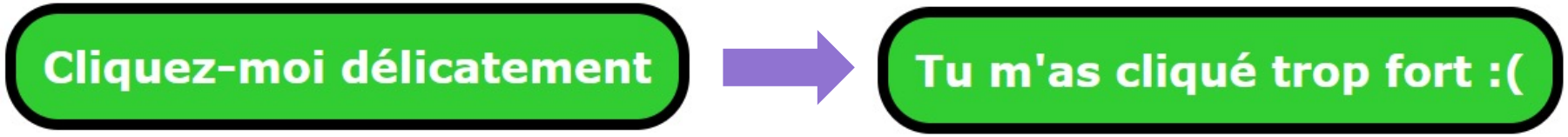


## ❖ Événements

- ♦ Les événements, permettent, entre autres, d'**appeler des fonctions** suite à la détection d'un **déclencheur**.
  - ○ Exemples simples :
    - En cliquant sur un **élément**... sa **couleur de fond** change !

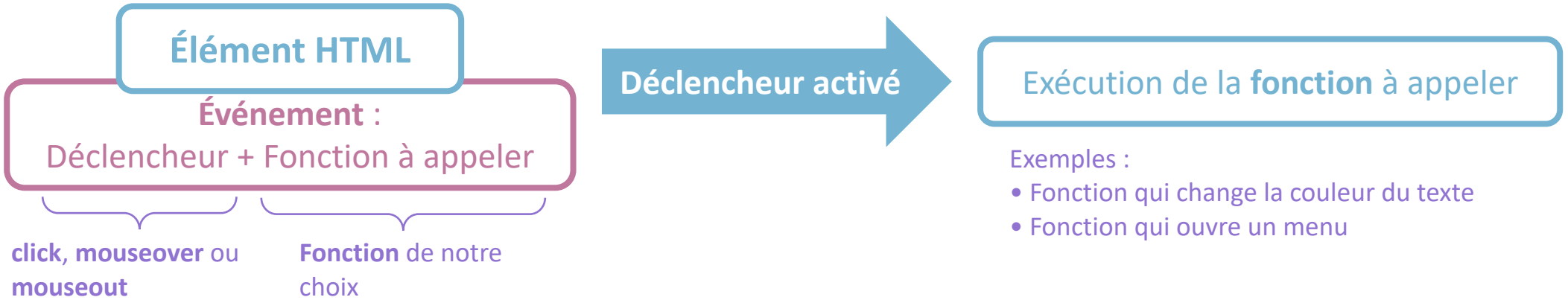



    - En cliquant sur un **élément**... son **texte** change !



## ❖ Événements

- ♦ Les 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)



## ❖ Événements

- ♦ Exemple d'activation d'un **événement** :
  - ○ On a un **élément** avec l'id "**bouton**". Il est associé à un **événement** de type « **click** » qui exécute la fonction « **fondRouge()** » lorsque déclenché.



```
function fondRouge(){
    document.getElementById( elementId: "bouton").style.backgroundColor = "red";
}
```

# ❖ Événements

## ♦ Comment ajouter un événement

### ○ Syntaxe :

```
document.getElementById("id").addEventListener('type', nom_fonction)
```

Élément associé à l'événement

Déclencheur et fonction

### ○ Exemple

```
document.getElementById( elementId: "bouton").addEventListener( type: 'click', vert);
```

# ❖ Événements

## ♦ Comment ajouter un événement

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

```
function init(){

    document.getElementById( elementId: "bouton").addEventListener( type: 'click', vert);
    document.getElementById( elementId: "titre").addEventListener( type: 'click', vert);
    document.getElementById( elementId: "piano").addEventListener( type: 'mouseover', fondJaune);

}
```

- Par exemple, ci-dessus, on peut voir que 3 **événements** sont déclarés.

# ❖ Alerte

- ♦ On peut envoyer une **alerte** à l'utilisateur grâce à la fonction **alert()**
  - Syntaxe : `alert("texte quelconque");`
- ♦ Cela ouvre une **boîte de dialogue** avec le texte spécifié.
  - L'apparence de cette boîte varie selon le **navigateur**.



- ♦ Glisser une **alerte** dans une **fonction** :
  - On pourrait appeler cette **fonction** suite au déclenchement d'un **événement** !
    - Exemple : **Cliquer** sur un bouton déclenche une **alerte**.

```
function alerteRouge(){
    this.style.color = "red";
    alert("Alerte rouge !");
}
```

## ❖ Mot-clé **this**

- ♦ Un aspect de la **fonction vert()** est embêtant : cette fonction ne marche que pour l'élément avec l'id « **bouton** » !

```
function vert(){
    document.getElementById( elementId: "bouton").style.color = "limegreen";
}
```

  - ○ 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 inutile de code** ! 😢

```
function vert(){
    document.getElementById( elementId: "bouton").style.color = "limegreen";
}

function vert2(){
    document.getElementById( elementId: "titre").style.color = "limegreen";
}
```

## ❖ Mot-clé **this**

- ♦ C'est ici que le mot-clé « **this** » est pratique.
  - ○ Si on remplace **document.getElementById("id")** dans la fonction **vert()** par **this**, c'est automatiquement **l'élément HTML qui appelle la fonction** (suite au déclenchement d'un événement) qui sera affecté par la **fonction**.

```
function vert(){
    this.style.color = "limegreen";
}
```

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

# Mot-clé this

## ❖ Mot-clé **this**

### ♦ Exemple

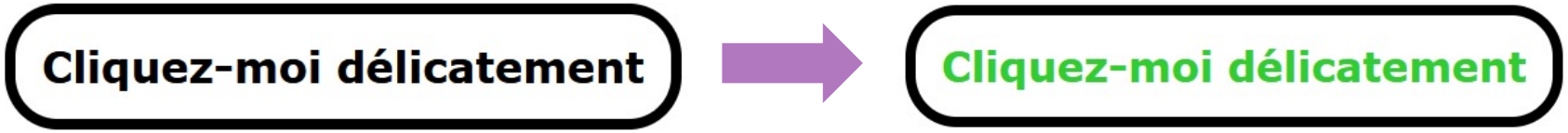
- Je **clique** sur l'élément avec l'id « **bouton** ».

```
<div id="bouton">Cliquez-moi délicatement</div>
```

- Étant donné qu'un événement « **click** » a été configuré pour cet élément, la fonction **vert()** est **appelée**.

```
function vert(){
    this.style.color = "limegreen";
}
```

- Grâce à la présence du mot-clé « **this** », c'est l'élément qui a été cliqué qui est affecté par l'instruction `.style.color = "limegreen";`
- Donc le texte de l'élément avec l'id « **bouton** » devient vert lime.



# Mot-clé this

## ❖ Mot-clé **this**

- ♦ Notons que le mot-clé « **this** » a d'autres utilités en JavaScript.
- ♦ Toutefois, nous n'abordons que son comportement dans le contexte des **événements** pour le moment !