

# Semaine 7

DOM (classes et attributs) et boucles

Intro. à la programmation



## ❖ DOM

- ◆ Classes
- ◆ Attributs
- ◆ Astuce avec DOM

## ❖ Boucles (Répétition)



- ❖ Le DOM permet de changer le style d'un élément, mais également :
  - ◆ Ajouter une classe
  - ◆ Retirer une classe
  - ◆ Vérifier si un élément possède une classe
  
- ◆ Ajouter un attribut
  - ◆ Retirer un attribut
  - ◆ Modifier un attribut



## ❖ **Classes** des éléments HTML

- ◆ Les éléments **HTML** possèdent parfois une ou plusieurs **classes**

```
<div class="spongebob"> ... </div>
```

```
<div class="spongebob patrick"> ... </div>
```

- Notez que lorsqu'un élément HTML a plus d'une classe, elles sont séparées par des **espaces**.

- ◆ Les **classes** permettent d'appliquer un groupe de styles à plusieurs éléments.

```
<div class="spongebob">Bob</div>
<div class="spongebob">Patrick</div>
<div class="spongebob">Carlos</div>
```

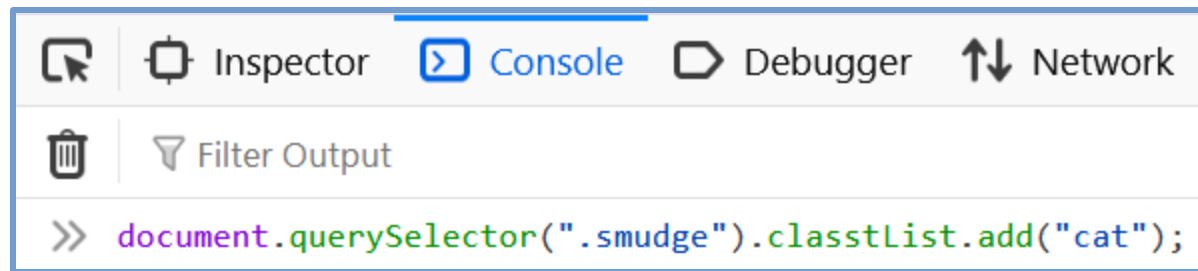
```
.spongebob{
  color: ■ yellow;
  background-color: ■ lightskyblue;
}
```

Bob  
Patrick  
Carlos



## ❖ Ajouter une classe :

```
document.querySelector(".classe").classList.add("nouvelle_classe")
```



```

```



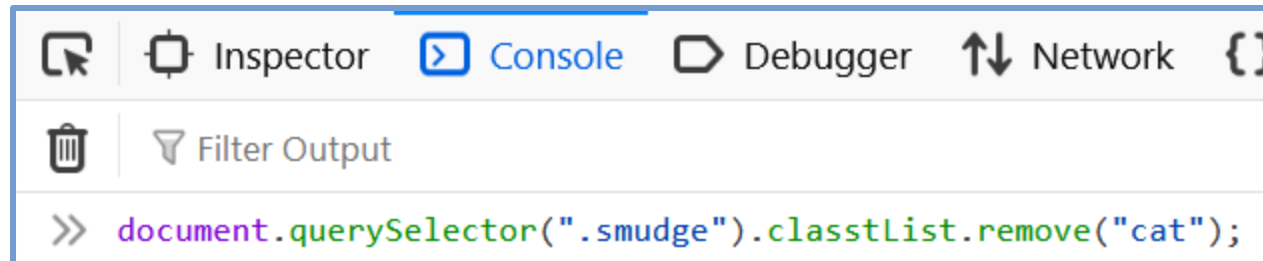
```

```



## ❖ Supprimer une classe :

```
document.querySelector(".classe").classList.remove("ancienne_classe")
```



```

```



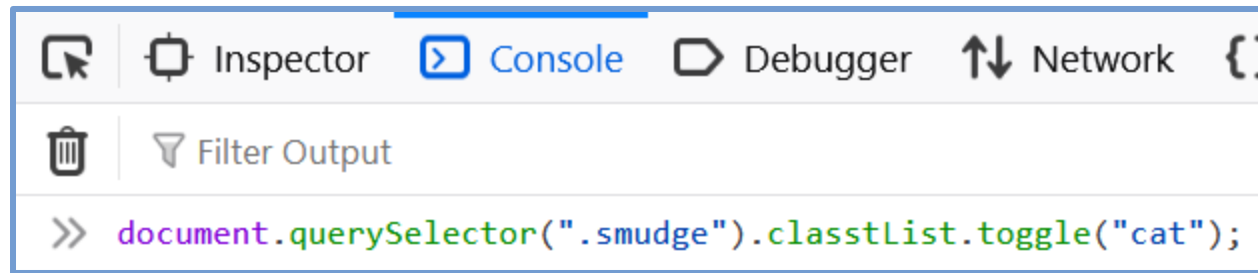
```

```



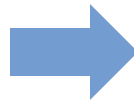
## ❖ « Basculer » la présence d'une classe dans un élément

- ◆ Donc si elle est présente, la retire. Si elle est absente, l'ajoute.
- ◆ Syntaxe : `document.querySelector(".classe").classList.toggle("classe")`



```

```



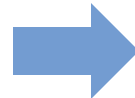
```

```

OU

```

```



```

```



## ❖ « Vérifier » si un élément possède une classe

- ◆ `document.querySelector(".classe").classList.contains("nom_classe")`
- ◆ Si l'élément possède la classe, le résultat est « **true** », sinon « **false** ».
  - Exemple

```
<p class="phrase poisson">Si on donne un poisson à un homme, il mange des micro-plastiques.</p>
```

```
let a = document.querySelector(".phrase").classList.contains("poisson");  
// a contient « true », car l'élément possède la classe poisson.
```

```
let b = document.querySelector(".phrase").classList.contains("chat");  
// b contient « false », car l'élément ne possède pas la classe chat.
```





## ❖ « Vérifier » si un élément possède une classe

- ◆ `document.querySelector(".classe").classList.contains("nom_classe")`
- ◆ Exemple de fonction qui exploite `classList.contains()` :

```
<p class="texte sobre">Ton thé t'a-t-il ôté ta toux ?</p>
```

Si l'élément `.texte` possède la classe `sobre`, son texte devient "Je possède la classe sobre 😊". Sinon, son texte devient "Je ne possède pas la classe sobre 😭".

```
if(document.querySelector(".texte").classList.contains("sobre") == true){  
    document.querySelector(".texte").textContent = "Je possède la classe sobre 😊";  
}  
else{  
    document.querySelector(".texte").textContent = "Je ne possède pas la classe sobre 😭";  
}
```



## ❖ Quelques précisions au sujet des classes

- ◆ Si plusieurs éléments possèdent la même classe, `document.querySelector()` va seulement réussir à accéder au premier dans le code HTML.

→ 

```
<p class="texte">Je suis accessible</p>
<p class="texte">Je suis inaccessible 😞</p>
<p class="texte">Je suis aussi inaccessible 😊</p>
```

`document.querySelector(".texte")`... nous permettra seulement d'accéder et de modifier le premier élément !

- ◆ Si on retire toutes les classes d'un élément ... on ne peut plus accéder ou manipuler facilement l'élément avec `document.querySelector()`...

```
<p>Bonne chance pour me modifier 🙈</p>
```



- ❖ Les **éléments HTML** possèdent parfois un ou plusieurs **attributs**
  - ◆ Ils sont situés dans la **balise ouvrante**.

```
<p>Pas d'attributs</p>
```

Attribut

Sa valeur

```
<p title="Titre">Un attribut</p>
```

```
<p id="banniere" title="Bannière">Deux attributs</p>
```

```
<p id="banniere" class="titre" title="Bannière">Trois attributs</p>
```



## ❖ Ajouter ou modifier un attribut à un élément HTML :

```
document.querySelector(".classe").nomAttribut = "valeur";
```

- Ex : `document.querySelector(".babyShark").title = "Baby shark doo doo doo doo";`

```
<div class="babyShark"> ... </div>
```



```
<div class="babyShark" title="Baby shark doo doo doo doo"> ... </div>
```

- Notez que si l'attribut avait déjà une valeur, elle serait remplacée :

```
<div class="babyShark" title="Doo doo ?"> ... </div>
```



```
<div class="babyShark" title="Baby shark doo doo doo doo"> ... </div>
```



## ❖ Retirer un attribut à un élément HTML :

```
document.querySelector(".classe").nomAttribut = "";
```

○ Exemple : `document.querySelector(".babyShark").style = "";`

```
<div class="babyShark" style="color: ■ aqua;"> ... </div>
```



```
<div class="babyShark" style=""> ... </div>
```

(Même si l'attribut `style` est toujours présent, il est vide, donc il ne fait rien)



## ❖ « Obtenir » la **valeur** d'un attribut

`document.querySelector(".classe").nomAttribut`

```

```

- Exemple : let **altBabyShark** = document.querySelector(".babyShark").**alt**
  - La variable **altBabyShark** contient donc la valeur "Bébé requin".

```
if(document.querySelector(".babyShark").alt == "Bébé requin"){  
    console.log("Le texte alternatif est valide ! 🐼💩");  
}  
else{  
    document.querySelector(".babyShark").alt = "Bébé requin";  
    console.log("Le texte alternatif a été corrigé.");  
}
```



## ❖ Résumé de toutes les modifications qu'on peut faire sur le DOM

```
document.querySelector(".classe") ...
```

- ◆ Obtenir / modifier / ajouter du contenu textuel
  - `.textContent = "..."`
- ◆ Modifier les styles
  - `.style.propriété = "valeur"`
- ◆ Ajouter un événement (Rendre interactif un élément)
  - `.addEventListener("type", fonction)`
- ◆ Modifier / obtenir des classes
  - `.classList.add("maClasse")`
  - `.classList.remove("maClasse")`
  - `.classList.toggle("maClasse")`
  - `.classList.contains("maClasse")`
- ◆ Modifier / obtenir des attributs
  - `.nomAttribut = "valeur"`



## ❖ Usage de **currentTarget**

- ◆ Notez qu'on peut très bien utiliser **currentTarget** pour modifier les classes et attributs !
  - Cela permet de modifier les classes ou attributs de l'élément avec lequel on vient d'interagir (clic, survol ou fin du survol)
- ◆ Modifier / obtenir des **classes**
  - `event.currentTarget.classList.add("maClasse")`
  - `event.currentTarget.classList.remove("maClasse")`
  - `event.currentTarget.classList.toggle("maClasse")`
  - `event.currentTarget.classList.contains("maClasse")`
- ◆ Modifier / obtenir des **attributs**
  - `event.currentTarget.nomAttribut = "valeur"`





- ❖ Vous devez manipuler fréquemment des éléments HTML via le DOM...



```
document.querySelector(".mario").textContent = "Mario brosse 🍄";  
document.querySelector(".mario").style.color = "red";  
document.querySelector(".mario").style.borderWidth = "5px";  
document.querySelector(".mario").title = "Mario";  
document.querySelector(".mario").classList.add("mamma_mia");  
document.querySelector(".mario").classList.remove("mushroom");
```

- ◆ Constamment réécrire `document.querySelector("...")` vous épuise ? 😞



- ❖ Vous pouvez « ranger » un **élément HTML** dans une variable 🧠



```
let elementMario = document.querySelector(".mario");

elementMario.textContent = "Mario brosse 🍷";
elementMario.style.color = "red";
elementMario.style.borderWidth = "5px";
elementMario.title = "Mario";
elementMario.classList.add("mamma_mia");
elementMario.classList.remove("mushroom");
```

- ◆ Pas besoin de réécrire `document.querySelector("...")` à chaque fois pour l'élément **.mario** !
  - Vous pouvez le faire avec n'importe quel **élément HTML** dès que vous comptez le modifier à plusieurs reprises.



## ❖ Répéter des bouts de code similaires...

```
document.querySelector(".message1").style.color = "red";  
document.querySelector(".message2").style.color = "red";  
document.querySelector(".message3").style.color = "red";  
document.querySelector(".message4").style.color = "red";  
document.querySelector(".message5").style.color = "red";  
document.querySelector(".message6").style.color = "red";
```

(La seule différence entre ces lignes de code est la **classe** des éléments)

- ◆ Il doit bien y avoir moyen de répéter ces lignes de code ultra similaires sans les réécrire 6 fois ?



## ❖ Boucles

◆ Permettent de **répéter** des bouts de code !

◆ Syntaxe :

```
while(condition d'exécution) {  
    // Code à répéter  
}
```

C'est un peu comme un **if**, sauf que le code s'exécute sans arrêt tant que la condition est **true**.

◆ Exemple :

- Cette boucle se répétera **3** fois

```
let i = 0;  
  
while(i < 3){  
    console.log("Allo");  
    i += 1;  
}
```

- La **condition d'exécution** est  **$i < 3$** . Ça signifie que tant que la variable **i** a une valeur **inférieure** à **3**, la boucle **while** va exécuter son code. (Les deux lignes entre les accolades **{ }**)
- Dans ce cas, « **Allo** » sera écrit **3** fois dans la console.
- À chaque fois que le code de la boucle est répété, la valeur de **i** augmente de **1**. Une fois que **i** atteint la valeur **3**, après la 3<sup>e</sup> répétition, **la boucle s'arrête** car la condition  **$i < 3$**  devient **false**.



## ❖ Boucles

### ◆ Exemple de déroulement pour la **boucle** illustrée à droite.

- La variable **i** commence avec la valeur **0**.
- Attention : remarquez que la valeur de **i** est affichée dans la console, PUIS elle augmente de **1**.

```
let i = 0;

while(i < 3){
  console.log(`Valeur de i : ${i}`);
  i += 1;
}
```

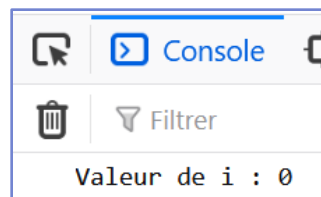
Cycle répétitif

La condition d'exécution est encore « true » ?

On exécute le code à l'intérieur de la boucle.

#### Itération #1

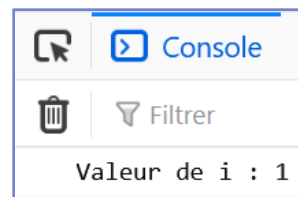
$i < 3$   
 $0 < 3$  est **true**



**i** passe à **1**.

#### Itération #2

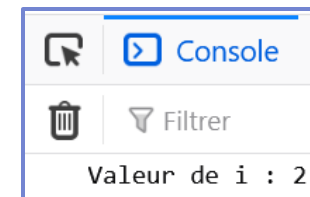
$i < 3$   
 $1 < 3$  est **true**



**i** passe à **2**.

#### Itération #3

$i < 3$   
 $2 < 3$  est **true**



**i** passe à **3**.

#### Itération #4

$i < 3$   
 $3 < 3$  est **false**  
Fin de la boucle





## ❖ Boucles


### ◆ Exemple 1


- Cette boucle fait **4 itérations** (4 répétitions)
- À chaque itération, on incrémente la variable « **valeur** » avec la valeur de **i**.


```
let valeur = 10;  
let i = 0;  
  
while(i < 4){  
  i += 1;  
  valeur += i;  
  console.log(`i vaut ${i} et valeur vaut ${valeur}.`);  
}
```



 Console

 Inspecteur



 Filtrer

i vaut 1 et valeur vaut 11.

i vaut 2 et valeur vaut 13.

i vaut 3 et valeur vaut 16.

i vaut 4 et valeur vaut 20.

- La **valeur** finale est :  $10 + 1 + 2 + 3 + 4$  (Donc 20)



## ❖ Boucles

### ◆ Exemple 2

- Cette boucle fait **9 itérations**.
- On se sert de la variable **i** à chaque itération pour ajouter du texte.

```
let elementNombres = document.querySelector(".nombres");
let i = 1;

while(i < 10){
  elementNombres.textContent += ` ${i}`;
  i += 1;
}
```

<div class="nombres"></div>



<div class="nombres"> 1 2 3 4 5 6 7 8 9</div>



## ❖ Boucles

### ◆ Exemple 3

- Cette boucle fait **3 itérations**. Elle ajoute donc la classe **image** à 3 éléments HTML.

```
let i = 1;

while(i < 4){
  document.querySelector(`.daenerys${i}` ).classList.add("image");
  i += 1;
}
```

```



```



```



```





## ❖ Boucles

### ◆ Construire une boucle

- Commencez par analyser un **code répétitif** et trouvez les **différences**.

```
document.querySelector(".daenerys1").classList.add("image");  
document.querySelector(".daenerys2").classList.add("image");  
document.querySelector(".daenerys3").classList.add("image");
```

- La seule chose qui varie dans ces **3 instructions**, c'est le **numéro** à la fin de la **classe**...
  - On a donc besoin d'une **boucle** où la variable **i** vaudra...
    - **1** pour la première itération
    - **2** pour la deuxième itération
    - **3** pour la troisième itération



## ❖ Boucles

### ◆ Construire une boucle

- On a besoin d'une **boucle** où **i** vaudra...

- **1** pour la première itération
- **2** pour la deuxième itération
- **3** pour la troisième itération

```
let i = 1;

while(i < 4){
    // ...
    i += 1;
}
```

- Il reste à intégrer le code et à se servir de la variable **i** pour remplacer la partie qui doit varier d'itération en itération

```
let i = 1;

while(i < 4){
    document.querySelector(`.daenerys${i}`).classList.add("image");
    i += 1;
}
```



## ❖ Boucles

### ◆ Construire une boucle

```
document.querySelector(".daenerys1").classList.add("image");  
  
document.querySelector(".daenerys2").classList.add("image");  
  
document.querySelector(".daenerys3").classList.add("image");
```



```
let i = 1;  
  
while(i < 4){  
  document.querySelector(`.daenerys${i}`).classList.add("image");  
  i += 1;  
}
```

- Si jamais on ajoute 2 images supplémentaires avec les classes « **daenerys4** » et « **daenerys5** », il suffira de changer la **condition d'exécution** pour **i < 6**



## ❖ Boucles

### ◆ Les dangers de l'infini

- Attention ! Les boucles peuvent **figer la page Web** si elles s'exécutent à l'infini.

```
let i = 1;

while(i < 4){
  console.log("Oups ! Boucle infinie.");
}
```

- ◆ Comme on a oublié de faire évoluer la valeur de **i** dans la boucle, **i** aura toujours la valeur **1** et **i < 4** sera toujours **true**.



## ❖ Boucles

### ◆ Exemple 4

- On peut très bien utiliser des **if** dans des boucles **while**, et des boucles **while** dans des **if**.

```
let i = 1;

while(i < 4){
  i += 1;
  console.log(`i vaut : ${i}.`);
  if(i == 4){
    console.log("La boucle est finie !");
  }
}
```

Console Insp

Filtrer

i vaut : 2.

i vaut : 3.

i vaut : 4.

La boucle est finie !



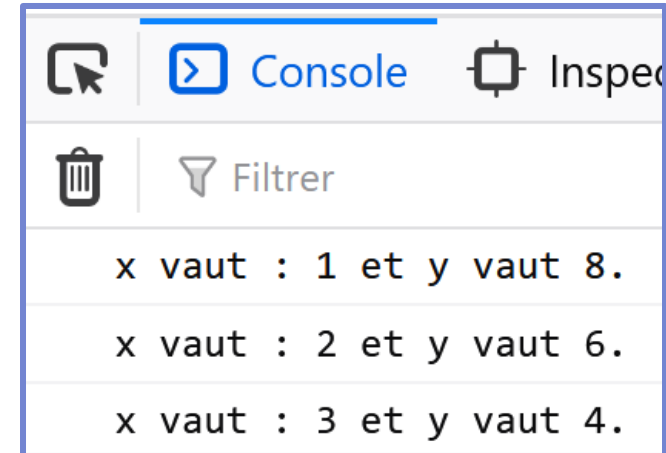
## ❖ Boucles

### ◆ Exemple 5

- Les conditions de boucle peuvent être aussi sophistiquées que nécessaire.

```
let x = 0;
let y = 10;

while(x < 5 && y > 5){
  x += 1;
  y -= 2;
  console.log(`x vaut ${x} et y vaut ${y}.`);
}
```



- À chaque itération, **x** augmente de **1** et **y** diminue de **2**.
- À cause de la condition, dès que **x** atteindra **5 ou plus** OU dès que **y** atteindra **5 ou moins**, la boucle s'arrêtera.
- Comme **y** diminue plus rapidement que **x** augmente, la boucle s'arrête alors que **x** respecte encore la condition, mais **y** vaut **4** et ne respecte plus la condition.



## ❖ Autre type de boucles

### ◆ do ... while

- Très similaire à une boucle **while**, mais la condition est vérifiée **APRÈS** chaque itération. (Plutôt qu'avant) Cela signifie qu'il y aura forcément **au moins une itération**.
- Avec une boucle **while**, si la **condition** était **false** initialement, elle n'effectuerait tout simplement aucune itération.

• Comme **i** vaut **1**, et que **1 < 1** est **false**, on ne rentre même pas dans la boucle !

• **i < 1** est encore **false**, mais on vérifie seulement la condition **APRÈS** que la première itération soit complétée.

```
let i = 1;

while(i < 1){
  console.log(`i vaut ${i}.`);
  i += 1;
}
```

```
let i = 1;

do{
  console.log(`i vaut ${i}.`);
  i += 1;
}while(i < 1); ← Condition placée à la fin
```

