Cours 9

Événements clavier, planificateurs, attribut data-, débogage

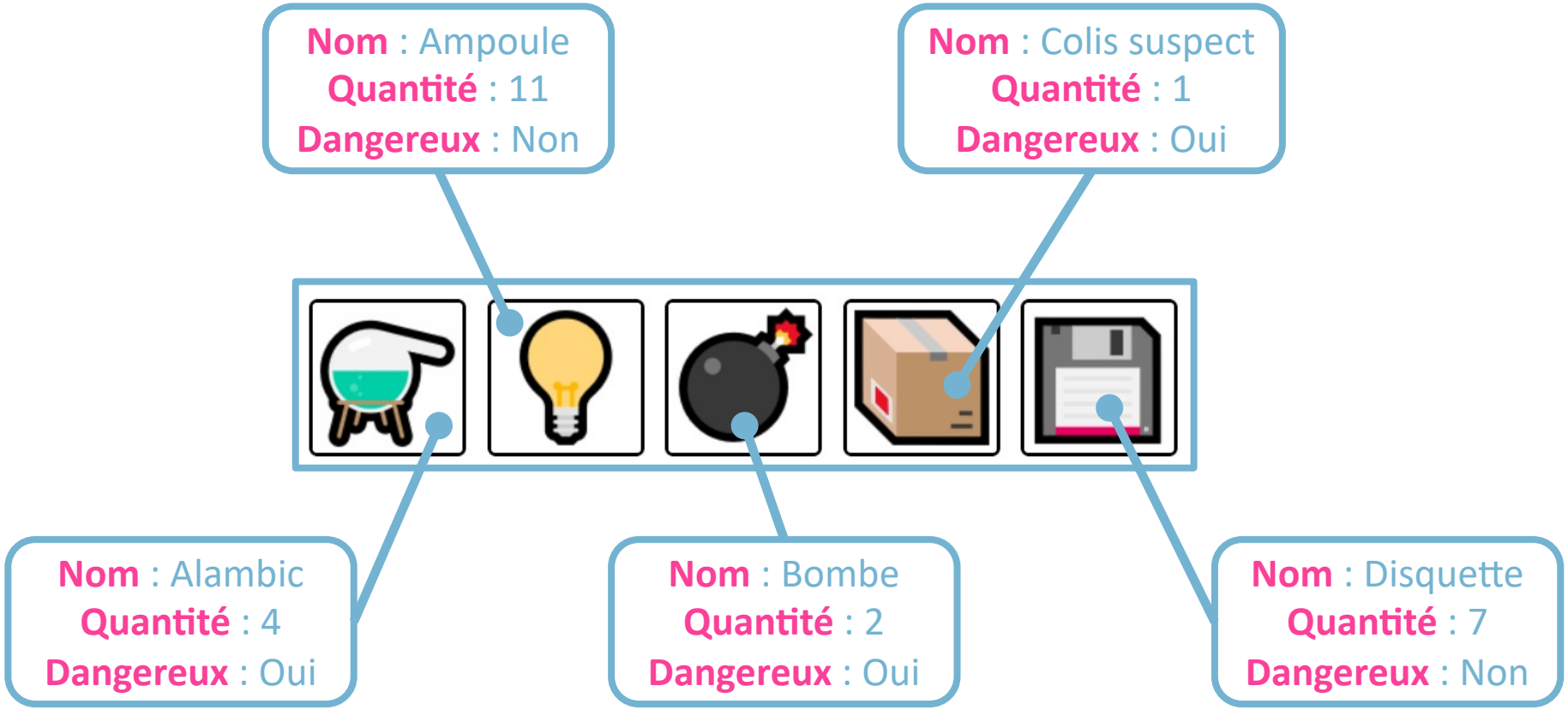
Intro. à la programmation - Aut. 2021

Menu du jour

- Attribut data-
  - parseInt() et parseFloat()
- Planificateurs
  - setTimeOut
  - setInterval
- Événements clavier
  - keydown
- Débogage avec Webstorm

Attribut data-

- Parfois, on veut stocker des données pour les éléments d'une page
  - Exemple : (Images dans une page Web)



Attribut data-

❖ Où stocker ces informations ?

♦ Dans des tableaux ?

```
let gNoms = ["Alambic", "Ampoule", "Bombe", "Colis suspect", "Disquette"];
let gQuantites = [4, 11, 2, 1, 7];
let gDangers = ["Oui", "Non", "Oui", "Oui", "Non"];
```

♦ Pas toujours pratique... si on change l'ordre des images dans la page où on retire un objet, ça devient vite embêtant. 😵

♦ Les informations sont classées par type d'information plutôt que par objet... Il faut garder à l'esprit que "Alambic", 4 et true vont ensemble... par exemple.

Attribut data-

❖ Attribut **data-**

- ♦ Permet de ranger n'importe quelle information **dans** un élément HTML
- ♦ Syntaxe de l'attribut :

```
data-propriété = "valeur"
```

- ♦ Exemple :

Nom : Alambic
Quantité : 4
Dangereux : Oui

```
<img id="alambic" src="images/alambic.png" alt="Alambic" data-nom="Alambic" data-quantite="4" data-danger="Oui">
```

- ♦ Comme les données sont stockées dans le HTML, pas besoin de les ranger dans des **tableaux** ou des **variables** dans ce cas-ci. 🥳

Attribut data-

- Comment **récupérer** ces données ?
  - On sait déjà comment 😎

```
<img id="alambic" src="images/alambic.png" alt="Alambic" data-nom="Alambic">
```

```
let nom = document.getElementById("alambic").getAttribute("data-nom");
// nom contient "Alambic"
```

`getAttribute("nomAttribut")` permet de **récupérer** la valeur d'un attribut de notre choix. On peut, par exemple, **stocker** cette valeur **dans une variable**.

Attribut data-

❖ Comment **stocker** / **modifier** des données ?

- ♦ On sait déjà comment aussi 💪

```
document.getElementById("alambic").setAttribute("data-quantite", "4");
```

```
<img id="alambic" src="images/alambic.png" alt="Alambic">
```

```
<img id="alambic" src="images/alambic.png" alt="Alambic" data-quantite="4">
```

`setAttribute("nomAttribut", "valeur")` permet d'**ajouter** ou de **modifier** un attribut de notre choix dans un **élément HTML**.

Attribut data-

❖ Utiliser les attributs data-

- ♦ **Exemple 1** : Afficher le nom d'un objet en cliquant dessus.

```
<img id="ampoule" src="images/ampoule.png" alt="Ampoule" data-nom="Ampoule">
```

Cette fonction est appelée lorsqu'un clique sur une image

```
function afficherNom(){
    // Obtenir le data-nom
    let nom = this.getAttribute("data-nom");

    // Afficher le nom dans la page
    document.getElementById("nom").textContent = nom;
}
```

Attribut data-

❖ Utiliser les attributs data-

- ♦ **Exemple 2** : Afficher toutes les données d'un objet en cliquant dessus.

```
<img id="colis" src="images/colis.png" alt="Colis"
data-nom="Colis suspect" data-quantite="1" data-danger="Oui">
```

Cette fonction est appelée lorsqu'un clique sur une image

```
function afficherInfo(){
    // Obtenir les données data-
    let nom = this.getAttribute("data-nom");
    let quantite = this.getAttribute("data-quantite");
    let danger = this.getAttribute("data-danger");

    // Afficher les données dans la page
    document.getElementById("nom").textContent = nom;
    document.getElementById("quantite").textContent = quantite;
    document.getElementById("danger").textContent = danger;
}
```

Nom de l'objet : Colis suspect

Quantité : 1

Dangereux : Oui

Attribut data-

❖ Utiliser les attributs data-

♦ Exemple 3 : Mettre une bordure rouge si l'objet est dangereux

```
function bordureSiDanger(monId){
    // Obtenir la valeur de data-danger (Donc "Oui" ou "Non")
    let danger = document.getElementById(monId).getAttribute("data-danger");

    // Mettre une bordure rouge SI la valeur de data-danger est "Oui"
    if(danger == "Oui"){
        document.getElementById(monId).style.borderColor = "red";
    }
}
```

```
<img id="bombe" src="images/bombe.png" alt="Bombe"
data-nom="Bombe" data-quantite="2" data-danger="Oui">
```

Appel de la fonction avec l'id "**bombe**"

```
bordureSiDanger("bombe");
```

```
<img id="disquette" src="images/disquette.png" alt="Disquette"
data-nom="Disquette" data-quantite="7" data-danger="Non">
```

Appel de la fonction avec l'id "**disquette**"

```
bordureSiDanger("disquette");
```

Chaîne de caractères -> Nombre

- Les **chaînes de caractères** contiennent parfois des **nombres**.
  - Exemples : `"2"`, `"7"`, `"1.5"`, `"43"`
  - Si on tente de les **additionner** sous cette forme ... ils se **concatènent** ...

    `"2" + "7"` -> `"27"` 😞
  - On peut transformer une **chaîne de caractères** en **nombre** !
    - De **chaîne de caractères** à nombre **entier** :

```
parseInt("2") -> 2
```

    - De **chaîne de caractères** à nombre **à virgule** :

```
parseFloat("1.5") -> 1.5
```

Chaîne de caractères -> Nombre

❖ Pour **additionner des nombres** qui sont sous forme de **chaîne de caractères**, il faut donc commencer par les **convertir en nombre**.

♦ Exemple : Additionner `"5"` et `"2.5"`

```
"5" + "2.5" vaut "52.5"  // Pas bon !
```

```
parseInt("5") + parseFloat("2.5") vaut 7.5  // Mieux !
```

(Équivalent à `5 + 2.5`)

Attribut data-

❖ Utiliser les attributs data-

- ♦ **Exemple 4** : Calculer la **somme** de la quantité de bombes et d'ampoules.

```
<img id="ampoule" src="images/ampoule.png" alt="Ampoule" data-quantite="11">
<img id="bombe" src="images/bombe.png" alt="Bombe" data-quantite="2">
```

- ♦ Petit problème !
  - ○ Quand nous allons faire `getAttribute("data-quantite")` pour ces éléments... nous allons obtenir `"2"` plutôt que `2`, par exemple. (Une **chaîne de caractères** plutôt qu'un **nombre**)
  - ○ Nous aurons donc besoin de `parseInt(...)` pour convertir les valeurs obtenues en **nombre**.

❖ Utiliser les attributs data-

- ♦ **Exemple 4** : Calculer la **somme** de la quantité de bombes et d'ampoules.

"2" → `let quantiteBombe = document.getElementById("bombe").getAttribute("data-quantite");`

"11" → `let quantiteAmpoule = document.getElementById("ampoule").getAttribute("data-quantite");`

```
function calculerSomme(){
    // Obtenir la quantité de bombes et d'ampoules
    let quantiteBombe = document.getElementById("bombe").getAttribute("data-quantite");
    let quantiteAmpoule = document.getElementById("ampoule").getAttribute("data-quantite");

    // Convertir la quantité en nombre et les additionner
    let somme = parseInt(quantiteBombe) + parseInt(quantiteAmpoule);
                                2                          11
    // Afficher la somme dans la page
    document.getElementById("quantiteSomme").textContent = somme;
}

// Somme vaut 13
```

Qté de bombes + Qté d'ampoules : 13

Planificateurs

- Planifier... quoi ? 🙄
  - L'appel de fonctions !

- setTimeOut ⌛
  - Permet d'appeler une fonction ... dans x millisecondes.
  - Syntaxe :

```
setTimeout(maFonction, tempsEnMillisecondes)
```

    - Exemple : `setTimeout(afficherNom, 3000)` appellera la fonction `afficherNom()` dans 3 secondes.

❖ setTimeOut ⏳

- ♦ Exemple : Afficher puis cacher une image brièvement.

```
function afficherFantome(){

    // Afficher boo dans 2 secondes
    setTimeout(afficherBoo, 2000);

    // Cacher boo dans 4 secondes
    setTimeout(cacherBoo, 4000);

}
```

```
function afficherBoo(){
    document.getElementById("boo").style.display = "block";
}

function cacherBoo(){
    document.getElementById("boo").style.display = "none";
}
```

Planificateurs

setInterval ⏱️

- Permet d'appeler une fonction ... toutes les x millisecondes.
- Syntaxe :

```
setInterval(maFonction, tempsEnMillisecondes)
```

  - Exemple : `setInterval(afficherAlerte, 3000)` appelera la fonction `afficherAlerte()` toutes les 3 secondes ! 😰

❖ setInterval ⏱️

- ♦ Exemple : Afficher puis cacher une image continuellement
  - ○ La fonction **`toggleCacher()`** sera appelée **toutes les secondes**.

```
function alternerCrewmate(){
    setInterval(toggleCacher, 1000);
}

function toggleCacher(){
    document.getElementById("crewmate").classList.toggle("cacher");
}
```

❖ Et si on veut mettre fin à **setInterval** ?

- ♦ Il existe la fonction **`clearInterval()`** pour arrêter un planificateur !
  - ○ Par contre, il va falloir suivre quelques étapes pour pouvoir l'utiliser.

- ♦ Étape 1 : Quand on utilise **`setInterval()`**, il faut le « stocker » dans une **variable globale**.

```
function alternerCrewmate(){
    gPlanificateur = setInterval(toggleCacher, 1000);
}
```

- ♦ Étape 2 : Quand on souhaite **arrêter le planificateur**, on utilise **`clearInterval()`**.

```
function stopCrewmate(){
    clearInterval(gPlanificateur);
}
```

Dans ce cas-ci, on a un bouton qui permet d'appeler **`stopCrewmate()`**, ce qui arrête le planificateur qu'on a rangé dans la variable globale **gPlanificateur**

Événements clavier

- Il existe un événement qui permet de savoir quand l'utilisateur appuie sur une **touche** de son clavier.
  - Pour pouvoir utiliser cet événement, il faut le créer comme ceci dans notre fonction **`init()`** :

```
function init(){
    // Écouteur  d'événements clavier
    document.addEventListener("keydown", toucheClavier);
}
```

    - Remarquez que cet événement n'est pas attaché à un élément HTML en particulier. Seulement au « ***document*** » ! (C'est-à-dire la page Web en entier)

Événements clavier

❖ Comment peut-on savoir sur **quelle touche** l'utilisateur a **appuyé** ?

```
function init(){
    // Écouteur  d'événements clavier
    document.addEventListener("keydown", toucheClavier);
}
```

♦ D'abord, on s'assure de créer une **fonction** qui sera appelée par cet événement. Dans ce cas-ci, c'est **`toucheClavier()`**.

- Cette fonction doit avoir un **paramètre**. Cela nous permettra d'accéder à la touche appuyée.

♦ Pour « savoir » quelle touche a été **appuyée**, nous allons stocker ceci dans une **variable** :

```
function toucheClavier(evenement){
    // On obtient et stocke la touche appuyée dans la variable touche
    let touche = evenement.key;
```

Événements clavier

- Pour « savoir » quelle touche a été **appuyée**, nous allons stocker ceci dans une **variable** :

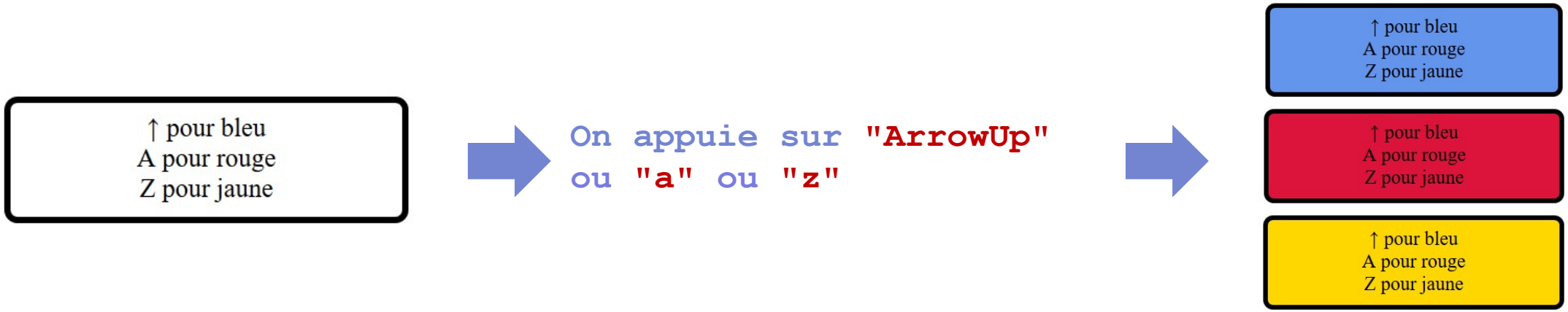
```
function toucheClavier(evenement){
    // On obtient et stocke la touche appuyée dans la variable touche
    let touche = evenement.key;
```

  - Que contient la variable **`touche`** ? Ça dépend de la **touche** qui a été **appuyée** :

Événements clavier

- On peut donc :
  - Savoir lorsqu'une touche est appuyée
  - Obtenir quelle touche a été appuyée

- Il nous reste à savoir ce qu'on veut faire avec ces informations !
  - Exemple : Quand on **appuie** sur une **touche**, cela change la **couleur de fond** d'un **élément HTML** :



Événements clavier

❖ Exemple : Quand on **appuie** sur une **touche**, cela change la **couleur de fond** d'un **élément HTML** :

- ♦ Étape 1 : On a notre écouteur d'événement clavier
  - Il appelle la fonction **`toucheClavier()`**

```
// Écouteurs d'événements clavier
document.addEventListener("keydown", toucheClavier);
```

- ♦ Étape 2 :
  - La fonction **`toucheClavier(parametre)`** va pouvoir obtenir la **touche** qui a été **appuyée** grâce à l'expression **`parametre.key`** (Ci-dessous, le paramètre a été nommé **`evenement`**)

```
function toucheClavier(evenement){
    // On obtient et stocke la touche appuyée dans la variable touche
    let touche = evenement.key;
```

Événements clavier

❖ Étape 3 : Que veut-on faire avec la touche appuyée ?

- ♦ Dans ce cas-ci, nous allons modifier la propriété backgroundColor du style de l'élément avec l'id "clavierFond".

```
function toucheClavier(evenement){
    // On obtient et stocke la touche appuyée dans la variable touche
    let touche = evenement.key;

    if(touche == "ArrowUp"){
        document.getElementById( elementId: "clavierFond").style.backgroundColor = "cornflowerblue";
    }
    else if(touche == "a"){
        document.getElementById( elementId: "clavierFond").style.backgroundColor = "crimson";
    }
    else if(touche == "z"){
        document.getElementById( elementId: "clavierFond").style.backgroundColor = "gold";
    }
}
```

↑ pour bleu
A pour rouge
Z pour jaune

↑ pour bleu
A pour rouge
Z pour jaune

↑ pour bleu
A pour rouge
Z pour jaune

❖ Débogage

- ♦ « Réparer un problème informatique dû à un bogue, c'est-à-dire à un mauvais fonctionnement d'un programme informatique, qui empêche le bon fonctionnement d'une application, d'un logiciel, etc. »

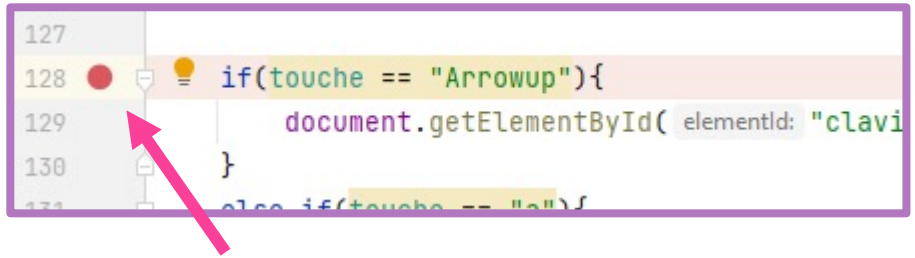
- ♦ Dans notre cas, cela signifie régler un bogue (« bug ») dans notre code **Javascript** qui fait obstacle au comportement qu'on souhaite obtenir.
  - ○ Il peut cela dit être difficile de trouver où est le problème lorsqu'on scrute le code.

Débogage avec Webstorm

❖ Stratégie : Utiliser des points d'arrêt

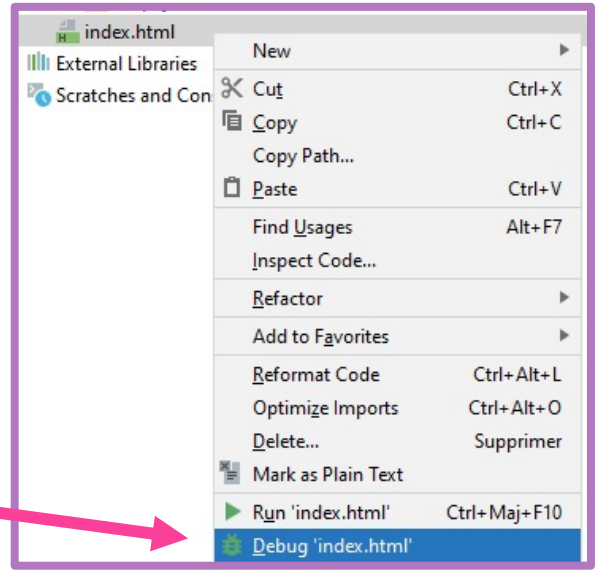
♦ Un point d'arrêt ?

- Permet d'interrompre brièvement l'exécution du code pour vérifier l'état du programme à un moment précis.



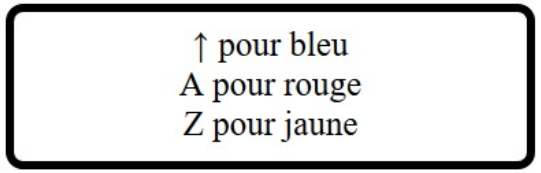
Pour ajouter un point d'arrêt, il suffit de faire un clic-gauche dans la marge vis-à-vis la ligne de code à laquelle nous souhaitons interrompre l'exécution.

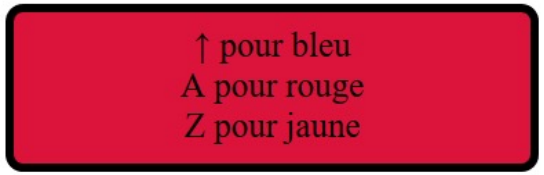
Pour tester l'exécution du programme AVEC le ou les points d'arrêt, on doit utiliser l'option qui permet de « Debug » notre page Web.

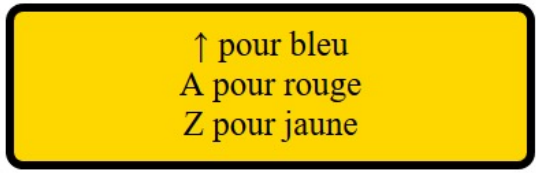


❖ Stratégie : Utiliser des points d'arrêt

- ♦ Exemple : En testant mon programme, je remarque un petit soucis.
  - ○ Appuyer sur A ou Z fonctionne, mais pas appuyer sur ↑ pour changer la couleur de fond.







Je déduis donc que le problème se situe dans cette portion du code.

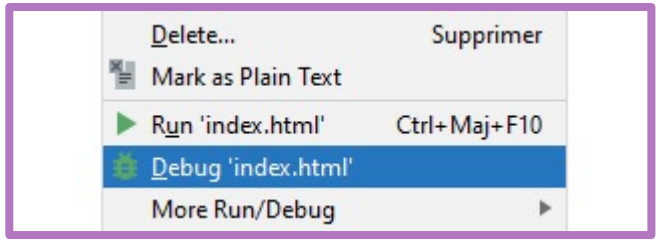
```
if(touche == "Arrowup"){
    document.getElementById( elementId: "clavierFond").style.backgroundColor = "cornflowerblue";
}
else if(touche == "a"){
    document.getElementById( elementId: "clavierFond").style.backgroundColor = "crimson";
}
else if(touche == "z"){
    document.getElementById( elementId: "clavierFond").style.backgroundColor = "gold";
}
```

Débogage avec Webstorm

❖ Stratégie : Utiliser des points d'arrêt

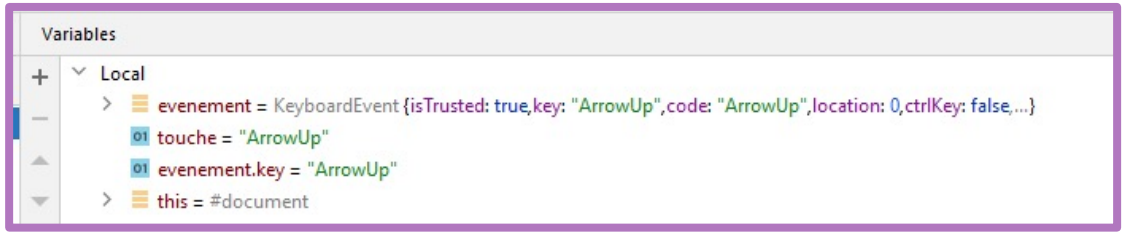
- ♦ J'ajoute donc des points d'arrêt pour ces lignes de code et je lance le « débogage » (debug) pour index.html.

```
128    if(touche == "Arrowup"){
129        document.getElementById( elementId: "clavierFond").style.backgroundColor = "cornflowerblue";
130    }
```



- ♦ Ma page Web se lance normalement, mais si je déclenche le code qui présente des points d'arrêt, Webstorm lancera une notification. (Dans ce cas-ci, quand j'appuyerai sur le clavier)
  - ○ Quand le point d'arrêt est atteint, il faut vérifier Web storm :

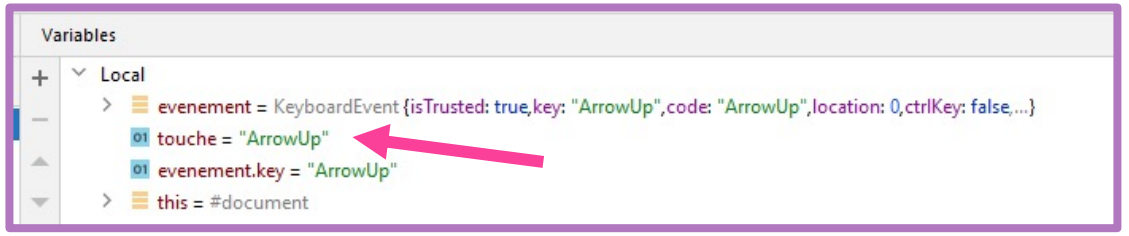
```
124  function toucheClavier(evenement){   evenement: KeyboardEvent {isTrusted: true,key: "ArrowUp",code: "
125      // On obtient et stocke la touche appuyée dans la variable touche
126      let touche = evenement.key;   evenement.key: "ArrowUp"
127
128      if(touche == "Arrowup"){   touche: "ArrowUp"
129          document.getElementById( elementId: "clavierFond").style.backgroundColor = "cornflowerblue";
130      }
```



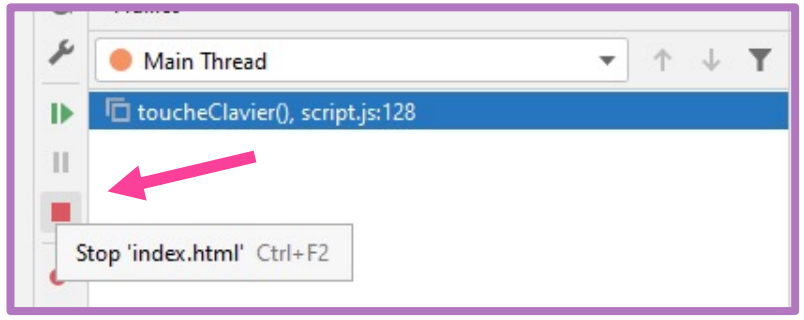
Débogage avec Webstorm

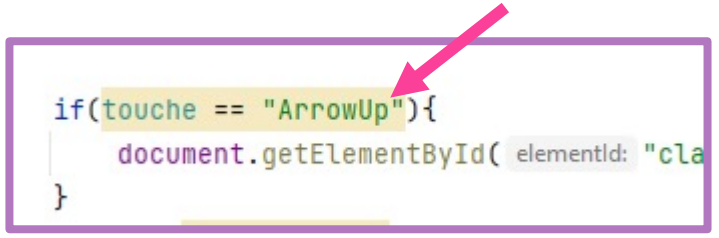
❖ Stratégie : Utiliser des points d'arrêt

- ♦ On a accès aux variables et leur valeur actuelle (au moment de l'exécution au point d'arrêt)



- ♦ On remarque que **touche** vaut "ArrowUp" et on la compare avec "Arrowup"…
  - ○ On a trouvé notre bug !
  - ○ On peut stopper le débogage et corriger le bogue.





Ne pas oublier de retirer les points d'arrêt !