

# Semaine 9

Fonctions avec retour, planificateurs, événements clavier

Intro. à la programmation



- ❖ Fonctions avec valeur de retour

- ❖ Planificateurs

  - ◆ setTimeout

  - ◆ setInterval

  - ◆ clearInterval

- ❖ Événements clavier

  - ◆ parseInt() et parseFloat()

  - ◆ keydown



- ❖ Créer une fonction avec une **valeur de retour**
  - ◆ À la fin de la fonction, on met le mot-clé « **return** » avec la valeur de notre choix.

```
function valeurPi(){  
  ...let pi = 3.14159265359;  
  ...return pi;  
}
```

Ici, on voit que la fonction **valeurPi()** va retourner la valeur 3.14159265...



## ❖ Appeler une fonction avec une valeur de retour

```
function valeurPi(){  
    ...let pi = 3.14159265359;  
    ...return pi;  
}
```

`valeurPi()` retourne la valeur 3.1415...

Voici ce qui se passe lorsqu'on appelle `valeurPi()` :

```
...let diametre = 3;  
...let perimetreCercle = diametre * valeurPi();
```

Ceci va se transformer en la valeur **retournée** par la fonction `valeurPi()`, c'est-à-dire 3.1415...

```
...let perimetreCercle = 3 * 3.14159265359;
```

Au final, le calcul se servira des valeurs ci-dessus



## ❖ Appeler une fonction avec une valeur de retour

- La fonction `Math.random()` existe par défaut. (Pas besoin de la créer, comme `alert()` et `console.log()`)
- Elle retourne un nombre aléatoire entre 0 et 0.999999...
- C'est très utile pour **simuler le hasard** !

```
let gNombreAleatoire = Math.random();
```

```
>> gNombreAleatoire  
← 0.6253974120258676  
>>
```

```
let nombreAleatoire = Math.random();  
if(nombreAleatoire < 0.25){  
  console.log("Bravo ! Tu gagnes 100$ !");  
}  
else{  
  console.log("Ilala... Tu as perdu 50$.");  
}
```

- Par exemple, dans la variable `nombreAleatoire`, on a un nombre entre 0 et 0.9999... (On ne peut pas connaître sa valeur d'avance, elle est **aléatoire** !)
- Avec le `if ... else`, on a **25%** de chances de gagner 100\$ et **75%** de chances de perdre 50\$.





## ❖ Point de non-retour !

- ◆ Notez que dès que l'instruction **return** est exécutée, on met fin à la fonction !

Ceci ne sera jamais exécuté, car l'instruction **return** est atteinte avant. **gNombre1** continue de valoir 3.

```
let gNombre1 = 3;

function test(){
    let x = 2;
    return x + 1; // On retourne 3
    gNombre1 = 4;
}

test() + gNombre1 // 3 + 3
```





## ❖ Point de non-retour !

- ◆ S'il y a plusieurs **return**, la fonction est interrompue dès qu'on atteint un de ceux-ci.
  - Dans ce cas précis, il y a un **"chat"** dans le tableau, alors c'est **"Il y a un chat 🐱"** qui sera **retourné**. (Et mis dans la console) Nous n'allons jamais atteindre l'autre **return**.

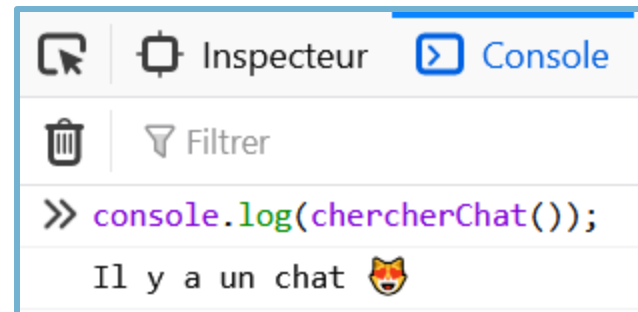
```
let gAnimaux = ["chat", "chien", "rat"];

function chercherChat(){

    let i = 0;

    while(i < gAnimaux.length){
        if(gAnimaux[i] == "chat"){
            return "Il y a un chat 🐱";
        }
        i += 1;
    }

    return "Pas de chat 🐱";
}
```





- ❖ On peut retourner n'importe quel type de données !
  - ◆ nombre, chaîne de caractères, booléen, tableau, élément HTML, etc.

## Retourner un tableau

```
function creerTableau(){  
    return [1, 2, 3];  
}  
  
let monTableau = creerTableau();  
// monTableau contient [1, 2, 3];
```

## Retourner un booléen

```
function tousEgaux(){  
    if(gNombre1 == gNombre2 && gNombre1 == gNombre3){  
        return true;  
    }  
    return false;  
}
```

## Retourner un élément HTML

```
function obtenirMario(){  
    return document.querySelector(".mario");  
}  
  
let elementMario = obtenirMario();
```





- ❖ En utilisant des **paramètres** ET une **valeur de retour**, on exploite le plein potentiel des fonctions !  

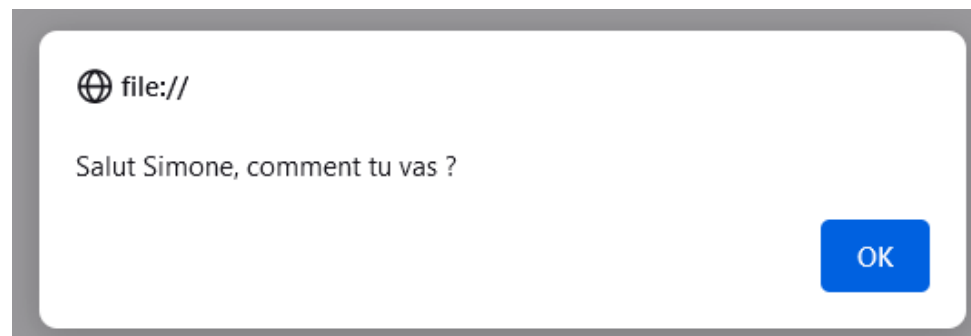
```
function maximum(x, y){  
  if(x > y){  
    return x;  
  }  
  else{  
    return y;  
  }  
}  
  
let nombre = maximum(2, 4); // nombre contient 4
```

Cette fonction prend 2 **paramètres** (deux nombres) et **retourne** la valeur la plus élevée.



On envoie un nom (chaîne de caractères) en **paramètre** et la **fonction** nous **retourne** un message (chaîne de caractères) qu'on peut utiliser dans une **alerte**, par exemple.

```
function saluer(nom){  
    return `Salut ${nom}, comment tu vas ?`;  
}  
  
alert(saluer("Simone"));
```





❖ Même fonction, en un peu plus complexe

```
function saluer(moment, nom){  
    if(moment == "jour"){  
        return `Bonjour ${nom}, comment tu vas ?`;  
    }  
    else{  
        return `Bonsoir ${nom}, comment tu vas ?`;  
    }  
}  
  
alert(saluer("soir", "Simone"));
```





❖ On peut prendre un tableau en paramètre 🤪

```
function totalTableau(tab){  
  
    let total = 0;  
    let i = 0;  
  
    while(i < tab.length){  
        total += tab[i];  
        i += 1;  
    }  
  
    return total;  
}  
  
let nombres = [1, 2, 3, 4];  
let somme = totalTableau(nombres); // Contient 10
```

Ici, la fonction `totalTableau()` prend un **tableau de nombres** en **paramètre** et **retourne** la somme de toutes les valeurs du tableau.



- ❖ 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.

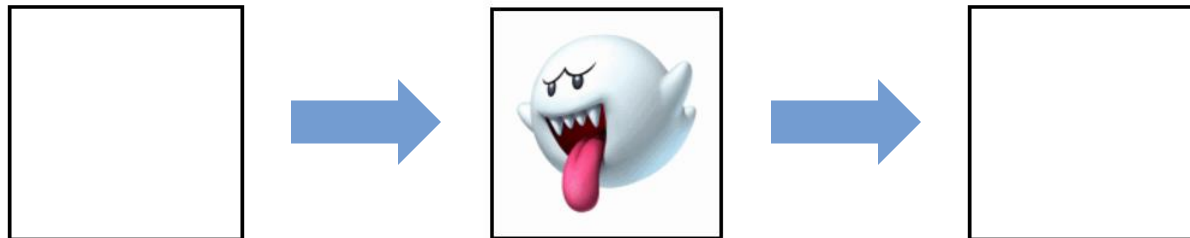


## ❖ setTimeout ⌚

◆ Exemple : Afficher puis cacher une image brièvement.

```
function boo(){  
  
    // Afficher boo dans 2 secondes  
    setTimeout(afficherBoo, 2000);  
  
    // Cacher boo dans 4 secondes  
    setTimeout(cacherBoo, 4000);  
  
}
```

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



Visible pendant 2 secondes



## ❖ setInterval

◆ Permet d'appeler une fonction ... toutes les x millisecondes.

◆ Syntaxe :

`setInterval (maFonction, tempsEnMillisecondes)`

○ Exemple : `setInterval (afficherAlerte, 3000)` appellera la fonction `afficherAlerte()` toutes les 3 secondes ! 




## ❖ setInterval

- ◆ Exemple : Afficher puis cacher une image continuellement
  - La fonction **basculerVisibilite()** sera appelée toutes les secondes.



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



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

```
.cacher{  
  display:none;  
}
```



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

- ◆ Il existe la fonction **clearInterval()** pour arrêter un planificateur !
  - Cela dit, il va falloir suivre quelques étapes pour pouvoir l'utiliser.
- ◆ **Étape 1** : Quand on utilise **setInterval()**, il faut « stocker le planificateur à intervalle » 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`



❖ Les **chaînes de caractères** contiennent parfois des **nombre**s.

◆ Exemples : "2", "7", "1", "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**



❖ 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"

```
let x = "5";  
let y = "2.5";
```

```
x + y // vaut "52.5" 😞
```

```
parseInt(x) + parseFloat(y) // vaut 7.5 😊
```



- ❖ Il existe un **écouteur d'événements** qui permet de savoir quand l'utilisateur appuie sur une **touche** de son **clavier**.
  - ◆ Pour pouvoir utiliser ce genre d'**événement**, il faut ajouter 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)



## ❖ 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()**.
- ◆ Pour « savoir » quelle **touche** a été **appuyée**, nous allons stocker ceci dans une **variable** :

Exceptionnellement, on devra glisser une variable spéciale (sans **let**) ici.

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

# Événements clavier



- ◆ Une fois qu'on a préparé le début de la fonction qui gère l'événement clavier comme ceci ...

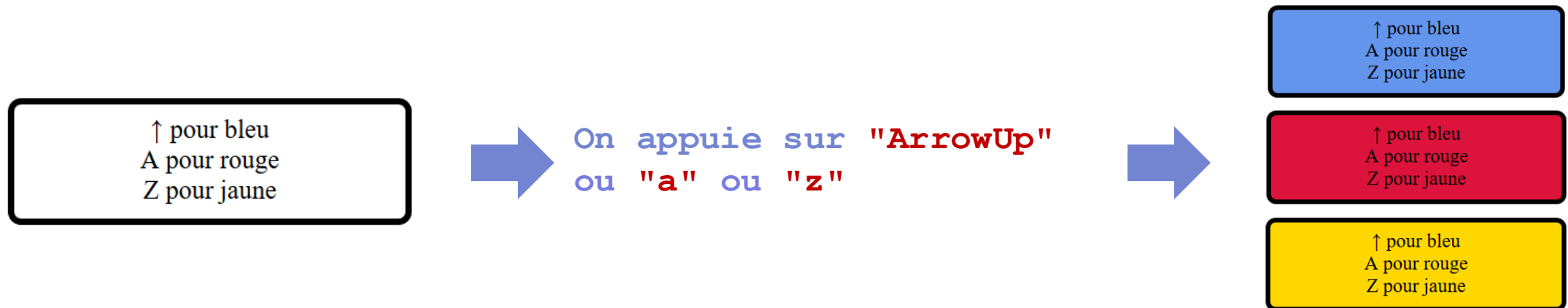
```
function toucheClavier(event){  
    // On obtient et stocke la touche appuyée dans la variable touche  
    let touche = event.key;
```

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





- ❖ On peut donc :
  - ◆ Savoir lorsqu'une **touche est appuyée**
  - ◆ Trouver **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** :





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

- ◆ Étape 1 : On ajoute notre **écouteur d'événement clavier** dans **init()**
  - Il appelle la fonction **toucheClavier()**

```
document.addEventListener("keydown", toucheClavier);
```

◆ Étape 2 :

- La fonction **toucheClavier(event)** va pouvoir obtenir la **touche** qui a été **appuyée** grâce à l'expression **event.key** (**event** aurait pu être nommé simplement **e** par exemple)

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





## ❖ 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 la classe **".clavierFond"**.

```
function toucheClavier(event){  
  
    let touche = event.key; // Touche appuyée  
    let element = document.querySelector(".clavierFond");  
  
    if(touche == "ArrowUp"){  
        element.style.backgroundColor = "cornflowerblue";  
    }  
    if(touche == "a"){  
        element.style.backgroundColor = "crimson";  
    }  
    if(touche == "z"){  
        element.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



## ❖ Déplacer un élément dans la page

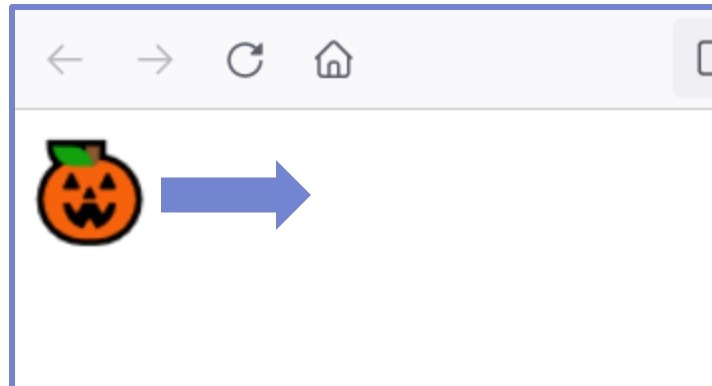
- ◆ Par exemple, on a cet élément dans la page :

```

```



- ◆ On aimerait, quand on appuie sur "**ArrowRight**" sur le clavier, déplacer l'image de citrouille vers la droite dans la page.





## ❖ Déplacer un élément dans la page

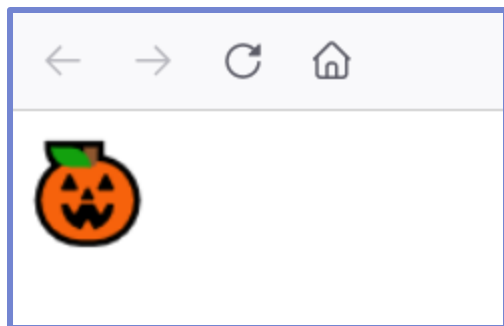
- ◆ Rappel sur les styles **left** et **top**

```

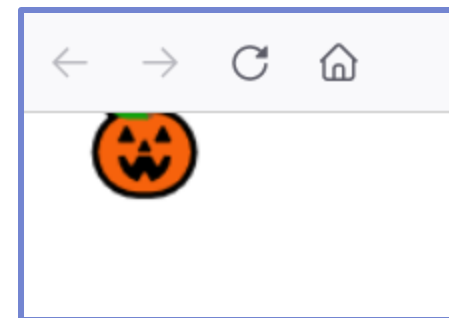
```



- ◆ **left** : Nombre de pixels d'espacement à gauche de l'élément.
- ◆ **top** : Nombre de pixels d'espacement en haut de l'élément.



`style="left:0px; top:0px;"`



`style="left:20px; top:-20px;"`



## ❖ Déplacer un élément dans la page

### ◆ Changer la valeur du style **left** (ou **top**)

- **Étape 1** : Ranger la valeur actuelle du style **left** dans une variable

```
let valeurLeft = document.querySelector(".pumpkin").style.left; // "0px"
```

---

- **Étape 2** : Se débarrasser de "px" et ne garder qu'une valeur numérique

```
valeurLeft = parseInt(valeurLeft); // "0px" -> 0
```

---

- **Étape 3** : Augmenter / réduire la valeur numérique

```
valeurLeft += 5; // 0 -> 5
```

---

- **Étape 4** : Changer le style **left** de l'élément avec la nouvelle valeur sans oublier de remettre le "px" après le nombre !

```
document.querySelector(".pumpkin").style.left = `${valeurLeft}px`; // "5px"
```



Utilisez la méthode que vous préférez !

## ❖ Déplacer un élément dans la page

### ◆ Changer la valeur du style **left** (ou **top**)

```
let valeurLeft = document.querySelector(".pumpkin").style.left; // "0px"
valeurLeft = parseInt(valeurLeft); // "0px" -> 0
valeurLeft += 5; // 0 -> 5
document.querySelector(".pumpkin").style.left = `${valeurLeft}px`; // "5px"
```

### ◆ On peut aussi le faire de manière plus compacte

```
let valeurLeft = document.querySelector(".pumpkin").style.left;
document.querySelector(".pumpkin").style.left = `${parseInt(valeurLeft) + 5}px`;
```

### ◆ Encore plus compacte

```
let pumpkin = document.querySelector(".pumpkin");
pumpkin.style.left = `${parseInt(pumpkin.style.left) + 5}px`;
```



## ❖ Déplacer un élément dans la page

### ◆ Avec un événement clavier

```
function toucheClavier(e){  
    let touche = e.key; // Quelle touche a été appuyée ?  
    let elementPumpkin = document.querySelector(".pumpkin");  
  
    if(touche == "ArrowLeft"){  
        elementPumpkin.style.left = `${parseInt(elementPumpkin.style.left) - 5}px`;  
    }  
    if(touche == "ArrowRight"){  
        elementPumpkin.style.left = `${parseInt(elementPumpkin.style.left) + 5}px`;  
    }  
}
```

- Appuyer sur "ArrowLeft" déplace l'élément de 5 pixels à gauche.
- Appuyer sur "ArrowRight" déplace l'élément de 5 pixels à droite.