# LES TIMERS

Vous l'avez bien compris maintenant JavaScript s'exécute sur le navigateur du client et permet d'interagir avec lui.

JavaScript va donc pouvoir détecter les évènements éventuels qui se produisent sur la page.

Pour le moment vous avez vu 2 types d'évènements :

- les évènement lancés par le navigateur lui même :
  - j'ai fini de charger et lire le HTML (DOMContentLoaded),
  - j'ai fini de charger toute la page et ses dépendances (load),
  - etc...
- les évènements lancés par le client (l'utilisateur) :
  - je clique sur un bouton ou une zone (click),
  - je survole une zone (mouseover),
  - je scrolle sur la page (scroll),
  - je soumet un formulaire (submit)
  - etc...

Nous pouvons donc interagir facilement avec ce qu'il se passe sur la page. Pourtant il nous manque une notion importante pour compléter ce panel d'évènements :

***Comment déterminer le temps qui passe et s'en servir pour lancer des actions***

Un exemple très simple de "timer", vous y êtes confronté très régulièrement sur Internet : au bout de 10 secondes une popup s'ouvre et vous propose de vous abonner à une Newsletter.

La gestion du temps en JavaScript va nous servir à bien d'autres choses que de la simple information ou communication agressive. Nous allons même y trouver ce qu'il y a de mieux, gérer des animations, temporiser des actions, et de façon plus complexe créer des jeux vidéos par exemple en utilisant toutes les fonctionnalités évènementielles de JavaScript !

## SETTIMEOUT ET CLEARTIMEOUT

### SETTIMEOUT

Cette méthode va nous permettre de définir un intervalle en millisecondes avant le déclenchement d'une action.

let timeoutId = window.setTimeout(callBackFunction [, delay, param1, param2, ...]);

**Paramètres**

- callBackFunction : c'est la fonction qui sera appelée une fois le délai dépassé. On peut ici fournir un nom de fonction défini ailleurs dans notre code ou directement une fonction anonyme (comme pour toutes les fonctions évènementielles JavaScript qui appellent une fonction de callBack),
- delay (optionnel) : le délai en millisecondes avant que la fonction de callBack ne soit appelée. Par défaut, ce paramètre vaut 0, la fonction est exécutée dès que possible. Astuce : 1000 ms = 1 s

- param1, param2... (optionnels) : les paramètres (valeurs) qui seront passés à la fonction de callBack. Attention ces paramètres ne sont pas compatibles avec IE9 (Internet Explorer 9).

**Valeur de retour**

- timeoutId : un identifiant unique fourni par Javascript si vous souhaitez retrouver et arrêter votre "timer" avant son exécution par exemple,

### CLEARTIMEOUT

Cette méthode va nous permettre d'arrêter le "timer" s'il n'a pas encore été déclenché.

window.clearTimeout(timeoutId);

**Paramètres**

- timeoutId : un identifiant unique fourni par Javascript que nous récupérons au lancement de **setTimeout**.

**Exemple : un message alert après 3 secondes**

## SETINTERVAL ET CLEARINTERVAL

### SETINTERVAL

Cette méthode va nous permettre de définir une action qui sera exécutée à intervalles réguliers.

let intervalID = window.setInterval(callBackFunction [, delay, param1, param2, ...]);

### PARAMETRES

- callBackFunction : c'est la fonction qui sera appelée à tous les intervalles déterminés. On peut ici fournir un nom de fonction défini ailleurs dans notre code ou directement une fonction anonyme (comme pour toutes les fonctions évènementielles JavaScript qui appellent une fonction de callBack),
- delay (optionnel) : le délai en millisecondes entre chaque intervalles d'exécution de la fonction de callBack. Par défaut, ce paramètre vaut 0, la fonction est exécutée dès que possible et continuellement. **Astuce :** 1000 ms = 1 s.
- param1, param2... (optionnels) : les paramètres (valeurs) qui seront passés à la fonction de callBack. Attention ces paramètres ne sont pas compatibles avec IE9 (Internet Explorer 9).

**Valeur de retour**

- timeoutID : un identifiant unique fourni par Javascript si vous souhaitez retrouver et arrêter votre "timer" à tout moment.

**clearInterval**

Cette méthode va nous permettre d'arrêter le "timer" à tout moment.

window.clearInterval(intervalID);

**Paramètres**

- intervalID : un identifiant unique fourni par Javascript que nous récupérons au lancement de **setinterval**.

**Exemple : afficher un compteur de secondes depuis le chargement complet de la page**

## REQUESTANIMATIONFRAME ET CANCELANIMATIONFRAME

### REQUESTANIMATIONFRAME

Cette méthode va nous permettre de définir une action qui sera exécutée au rafraichissement de l'écran.

let animationID = window.requestAnimationFrame(callBackFunction);

**Paramètres**

- callBackFunction : c'est la fonction qui sera appelée au prochain rafraichissement de l'écran. On peut ici fournir un nom de fonction défini ailleurs dans notre code ou directement une fonction anonyme (comme pour toutes les fonctions évènementielles JavaScript qui appellent une fonction de callBack),

  **Valeur de retour**

- animationID : un identifiant unique fourni par Javascript si vous souhaitez retrouver et arrêter votre demande de rafraichissement à tout moment.

### CANCELANIMATIONFRAME

Cette méthode va nous permettre d'arrêter l'exécution du rafraichissement à tout moment.

window.cancelAnimationFrame(animationID);

**Paramètres**

- animationID: un identifiant unique fourni par Javascript que nous récupérons au lancement de **requestAnimationFrame**.

**Mieux comprendre le fonctionnement : les FPS**

Alors que setInterval et setTimeout fonctionnent avec un délai en millisecondes, requestAnimationFrame lui va être exécute à chaque fois que le navigateur fait appel au rafraichissement de l'écran auprès de la machine du client (rafraichissement assuré en grande parti par le processeur graphique).

**Cela signifie que :**

- la méthode requestAnimationFrame est plus adapté à la création d'animations fluides,
- le rafraichissement va dépendre de la machine client, en général un écran est rafraichi 60 fois par seconde, on dit que l'on affiche 60 FPS (frames par seconde),
- le rafraichissement va dépendre aussi du travail demandé au navigateur mais aussi de sa capacité à ce moment là d'assurer le travail (capacité de l'ordinateur, navigateur plus ou moins utilisé et nombre d'onglets ouverts), les FPS sont donc variables comme dans les jeux vidéos, mais le rafraichissement sera effectué quoi qu'il arrive,

- le navigateur va optimiser et préparer le rafraichissement, contrairement à un setInterval, il sait ce qu'il a à faire au prochain rafraichissement et va le prioriser et l'optimiser,
- le navigateur ne fera l'action de rafraichissement que si l'onglet du navigateur comportant le script est actif, contrairement à setInterval et setTimeout qui continuerons à être exécutés même si un autre onglet est actif (donc on optimise la charge du navigateur).

Alors tout ça à l'air magique mais il y a ici une chose que nous ne maitrisons pas c'est le nombre de frames par seconde (FPS). Il nous faudra ralentir le taux de rafraichissement si nous en avons besoin. L'exemple si dessous nous montre comment obtenir "facilement" la valeur des FPS !