

# JavaScript

## Table of Contents

Objectifs .....	1
Pourquoi organiser son code ? .....	1
D'ailleurs, depuis le début de la formation, on essaie de séparer les concepts : .....	1
Or, jusqu'à présent lors cette saison : .....	2
Comment s'organiser en JS ? .....	2
Mise en module de l'affichage d'un message d'erreur 1/4 .....	2
Mise en module de l'affichage d'un message d'erreur 2/4 .....	2
Mise en module de l'affichage d'un message d'erreur 3/4 .....	3
Mise en module de l'affichage d'un message d'erreur 4/4 .....	3
Module slider 1/2 .....	4
Module slider 2/2 .....	5
Conclusion .....	7
Objectif / Niveau .....	8
A toi de jouer .....	8

## Module 05 - Les modules JavaScript

## Objectifs

- Savoir bien organiser son code

## Pourquoi organiser son code ?

- Rend le code plus lisible
- Facilite la maintenance
- Facilite l'évolutivité
- Réduit la dette technique

## D'ailleurs, depuis le début de la formation, on essaie de séparer les concepts :

- Par exemple : HTML d'un côté, CSS de l'autre
- Idem pour PHP, on a essayé de séparer la préparation des données et génération du HTML dans les templates.

# Or, jusqu'à présent lors cette saison :

On a certes codé directement dans des fichiers dédiés JS, mais on a utilisé des fonctions "libres" (non regroupées) et notre code n'est pas vraiment rangé ☹

## Comment s'organiser en JS ?

- Découper en plusieurs fichiers, on l'a déjà mis en place en essayant de faire un fichier par fonctionnalité ☹☹
- Pour aller plus loin, on va regrouper nos fonctions dans des "modules" :
  - un **module** est un regroupement de fonctions dans lequel on peut également déclarer des variables appelée propriétés
  - précision, toute fonction appartenant à un "module" s'appelle une méthode
  - en réalité, un **module** est un objet (concept abordé en S04)

## Mise en module de l'affichage d'un message d'erreur 1/4

On reprend ce qui a été fait en atelier pour afficher un message d'erreur "Vous devez être connecté..." lorsque l'on clique sur l'icône ☹☹ de mise en favoris d'une annonce.

## Mise en module de l'affichage d'un message d'erreur 2/4

Création du module **messages** dans un fichier messages.js

```
// Module générique de gestion des messages d'informations
const messages = {

  // Méthode permettant d'ajouter un message à l'intérieur d'un élément
  addMessageToElement: function(messageContent, parentElement) {

    // suppression des anciens messages
    messages.removeOldMessages(parentElement);

    // ajout du nouveau message
    const messageElement = document.createElement('p');
    messageElement.className = 'message';
    messageElement.textContent = messageContent;

    // on ajoute le message en premier enfant
    parentElement.prepend(messageElement);
  },


  removeOldMessages: function(parentElement) {

    const oldMessages = parentElement.querySelectorAll('.message');

    for (const oldMessage of oldMessages) {
      // https://developer.mozilla.org/en-US/docs/Web/API/Element/remove
      oldMessage.remove();
    }
  }
};
```

## Mise en module de l’affichage d’un message d’erreur 3/4

Création d’un second module **destinations** dans un fichier destinations.js

- Objectif : gérer le bouton like d’ajout à sa liste de destinations favorites
-  Nouveauté : Utilisation du init() au DOMContentLoaded

## Mise en module de l’affichage d’un message d’erreur 4/4

```

// Module de gestion des actions possibles pour les destinations
const destinations = {

    // un module est un objet, il peut aussi contenir des propriétés
    notLoggedInUserMessage: 'Vous devez être connecté pour gérer vos favoris',

    // méthode permettant d'initialiser le module destination,
    // souvent la méthode init() sert à ajouter des écouteurs d'évènement
    init: function() {
        destinations.addLikeEvents();
    },

    // méthode permettant d'écouter les clics sur les boutons like
    addLikeEvents: function () {

        // on récupère l'ensemble des boutons like
        const heartElements = document.querySelectorAll('.btn__like');

        // sur chaque bouton like, on écoute le clic
        for (const heartElement of heartElements) {
            heartElement.addEventListener('click', destinations.handleLikeClick);
        }

    },

    // création d'une erreur dans la carte la plus proche
    handleLikeClick: function (event) {
        // event.target récupère l'élément sur lequel l'évènement a eu lieu
        // => le bouton like
        // .closest('.card') récupère le premier ancêtre possédant la classe 'card'
        const destElement = event.target.closest('.card');

        // on fait appelle au module messages pour ajouter notre message d'information
        messages.addMessageToElement(destinations.notLoggedInUserMessage,
        destElement);
    }
};

// En appelant la méthode init() une fois le DOM chargé, cela permettra notamment
// d'ajouter les écouteurs
// d'évènement sur les boutons like :heart: de chaque destination
document.addEventListener('DOMContentLoaded', destinations.init);

```

## Module slider 1/2

Autre exemple : Mise en module de la fonction generateSliderImages créée en E02.

## Module slider 2/2

```
const slider = {

    // On crée un tableau qui contiendra toutes les images du slider
    // Chaque image est identifiée par son index.
    sliderImages: [],

    // Sert à stocker le nombre d'images du slider
    sliderImagesNumber: 0,

    // On définit une variable qui contiendra l'index de l'image courante,
    // par défaut, c'est la première donc 0.
    currentPosition: 0,

    // La méthode init permet d'initialiser les propriétés du "module" slider
    // et d'ajouter les écouteurs d'évènement
    init: function() {

        // On génère les images du slider avec la fonction créée précédemment
        slider.generateSliderImages();

        // On récupère toutes les slides de la page et on les stocke dans la propriété
        sliderImages pour pouvoir les réutiliser.
        slider.sliderImages = document.querySelectorAll('.slider__img');

        // On stocke le nombre d'images du slider pour ne pas avoir à le recalculer
        plusieurs fois
        slider.sliderImagesNumber = slider.sliderImages.length;

        // On ajoute les écouteurs d'évènement
        slider.addEvents();
    },

    // La méthode addEvents permet d'ajouter tous les écouteurs d'évènements associés
    au slider
    addEvents: function() {

        // On récupère les boutons précédent et suivant
        const sliderButtons = document.querySelectorAll('.slider__btn');

        // On place un écouteur d'évènement sur le bouton précédent
        const previousSliderButton = sliderButtons[0];
        previousSliderButton.addEventListener('click', slider.previousSlide);

        // On place un écouteur d'évènement sur le bouton suivant
        const nextSliderButton = sliderButtons[1];
        nextSliderButton.addEventListener('click', slider.nextSlide);
    }
};
```

```

},

// La fonction previousSlide est appelée
// lorsqu'on clique sur le bouton précédent.
// Elle permet de calculer la nouvelle position de l'image courante
// et de l'afficher.
previousSlide: function() {

    // On détermine la nouvelle position de la slide
    // en diminuant le compteur de 1.
    let newPosition = slider.currentPosition - 1;

    // Si la nouvelle position est inférieure à 0,
    // c'est qu'on est arrivés à la première image.
    // On le place alors sur la dernière pour faire boucler le slider.
    if (newPosition < 0) {
        // On définit la nouvelle position comme étant
        // le nombre de slides moins une (car les index commencent à 0)
        newPosition = slider.sliderImagesNumber - 1;
    }

    // On appelle la fonction qui va modifier l'image courante
    // en lui passant la position de la nouvelle slide à afficher.
    slider.goToSlide(newPosition);
},

// La fonction nextSlide est appelée
// lorsqu'on clique sur le bouton suivant.
// Elle permet de calculer la nouvelle position de l'image courante et de
l'afficher.
nextSlide: function() {

    // On détermine la nouvelle position de la slide en augmentant le compteur de
1
    let newPosition = slider.currentPosition + 1;

    // Si la nouvelle position est supérieure au nombre d'images,
    // c'est qu'on est arrivés à la dernière image.
    // On le place alors sur la première pour faire boucler le slider.
    if (newPosition > slider.sliderImagesNumber - 1) {
        // On définit la nouvelle position comme étant 0
        newPosition = 0;
    }

    // On appelle la fonction qui va modifier l'image courante
    // en lui passant la position de la nouvelle slide à afficher.
    slider.goToSlide(newPosition);
},

// Fonction qui permet de changer l'image courante en fonction

```

```

// de la nouvelle position reçue en argument.
// newPosition = l'index de l'image qu'on veut afficher dans le tableau.
goToSlide: function(newPosition) {

    // On vérifie que la nouvelle image à afficher existe
    if (newPosition >= 0 && newPosition < slider.sliderImagesNumber) {

        // On récupère l'image actuellement affichée
        const currentSliderImage = document.querySelector('.slider__img--
current');

        // /\ Pour éviter toute erreur Javascript, avant d'enlever l'image
courant,
        // on vérifie qu'elle existe car si l'utilisateur modifie le code HTML,
        // il se peut qu'on ne trouve pas d'image avec la classe slider__img--
current,
        // et dans ce cas currentSliderImage vaudrait null
        // et le classList.remove() provoquerait une erreur dans notre script
        if (currentSliderImage) {
            currentSliderImage.classList.remove('slider__img--current');
        } else {
            console.warn('Il n\'y avait aucun slide affiché dans le diaporama');
        }

        // On récupère l'élément correspond à la nouvelle image à afficher
        const newSliderImage = slider.sliderImages[newPosition];

        // On lui ajoute la classe pour l'afficher dans le slider
        newSliderImage.classList.add('slider__img--current');

        // On met à jour la position courante
        slider.currentPosition = newPosition;

    } else {
        console.warn('Le slide à afficher n\'existe pas');
    }
}
};

// Maintenant que le code est dans un "module" (objet), il ne faut pas oublier
// de l'initialiser pour que les interactions soient activées.
document.addEventListener('DOMContentLoaded', slider.init);

```

## Conclusion

- Vous savez ce qu'est un module.
- Vous avez compris l'intérêt d'avoir un code bien organisé.
- Vous savez utilisé les modules.

# Objectif / Niveau

- **Essentiel** : Comprendre l'intérêt d'avoir un code organisé avec des modules.
- **Attendu** : Savoir utilisé les modules.
- **Avancé** : Vous avez le reflexe de toujours utiliser les modules lorsque cela est nécessaire.

## A toi de jouer

Challenge