



# JavaScript : programmation événementielle et asynchrone

# Chapitre 1: JavaScript et le DOM

## Qu'est-ce que le DOM ?

Document Object  
Model  
**Représentation  
objet** de  
l'arborescence du  
document  
HTML/CSS

Chaque nœud est  
un objet

Les navigateurs  
représentent le  
DOM  
graphiquement

JavaScript peut  
manipuler le DOM

# JavaScript : programmation événementielle et asynchrone

Page 4

## Et l'API du web ?

L'API du web est  
l'ensemble des  
classes des  
éléments  
HTML/CSS

Un élément HTML  
a deux  
représentations :

- ▼ balises, ex. : `<div>`
- ▼ objet, ex. :  
`HTMLDivElement`

En interne, les  
navigateurs  
transforment les  
balises en objets  
avant affichage

JavaScript manipule  
les objets

# JavaScript : programmation événementielle et asynchrone

Page 5

## Sommaire

Manipulation du  
DOM  
Sélection,  
création,  
modification,  
suppression des  
éléments  
HTML/CSS

Programmation  
événementielle  
Événements,  
propagation,  
écouteurs...

Programmation  
asynchrone  
Promesses, API  
Fetch, fonctions  
asynchrones...

# JavaScript : programmation événementielle et asynchrone

Page 6

## Ce qu'il faut **retenir**

- ▼ Le DOM est une représentation objet
- ▼ L'API du web est un ensemble de classes
- ▼ JavaScript est le langage pour accéder à l'API du web et manipuler le DOM



# **Chapitre 2 : L'objet global window**

# JavaScript : programmation événementielle et asynchrone

Page 8

## Objet global window

Créé  
automatiquement par  
JavaScript

Représente la fenêtre  
(ou l'onglet) du  
navigateur



## Ce qu'il faut **retenir**

### ▼ Quelques propriétés et méthodes utiles de l'objet global window :

- ▶ document
- ▶ location
- ▶ alert()
- ▶ confirm()
- ▶ setTimeout()
- ▶ setInterval()
- ▶ clearInterval()



# **Chapitre 3 :**

## **Sélectionner des éléments**

## Sélectionner des éléments

Sélectionner =  
récupérer une  
référence  
Donc une adresse  
mémoire

Parcours dans le DOM  
à la recherche du ou  
des éléments

## Ce qu'il faut **retenir**



- ▼ Une dizaine de méthodes hétérogènes et parfois obsolètes...
- ▼ Deux méthodes modernes, universelles et efficaces :
  - ▶ `querySelector()`
  - ▶ `querySelectorAll()`
- ▼ Elles appartiennent à la classe `Element`



# **Chapitre 4 :** **Créer et supprimer des éléments**

## Créer et supprimer des éléments

- ▼ JavaScript peut **créer, manipuler et accrocher** des éléments HTML au DOM

## Ce qu'il faut **retenir**



- ▼ Créer un élément :
  - ▶ `document.createElement()`
- ▼ Les images se créent avec un constructeur dédié `Image()`
- ▼ Les rangées et cellules de tableau se créent avec des méthodes dédiées
- ▼ Supprimer un élément « e » :
  - ▶ `e.remove()`

# Chapitre 5 :

## Gérer les règles CSS



## Gérer les règles CSS

- ▼ JavaScript peut créer, manipuler et supprimer des règles CSS

## Ce qu'il faut **retenir**

- ▼ Transformation des noms composés en **camelCase**
- ▼ Propriété **classList** pour manipuler les classes de style





# **Chapitre 6 : La programmation événementielle**

# La programmation événementielle

**JavaScript :**  
**programmation**  
**événementielle**  
**et asynchrone**

Page 20

Les événements (clics, saisies...) se propagent dans le DOM

La propagation obéit à des règles précises

## Ce qu'il faut **retenir**



- ▼ Trois phases de propagation :
  - ▶ Capture
  - ▶ Target
  - ▶ Bubbling
- ▼ Phases Capture et Bubbling exclusives l'une de l'autre
- ▼ Choix Capture OU Bubbling à la création de l'écouteur

# Chapitre 7 : Les écouteurs

# JavaScript : programmation événementielle et asynchrone

Page 23

## La programmation événementielle

- ▼ JavaScript peut écouter des événements et réagir à leur survenue

## Ce qu'il faut **retenir**



- ▼ Lorsqu'ils existent, les écouteurs on...  
sont très pratiques
- ▼ Les écouteurs lambda permettent une  
meilleure séparation de la vue et du  
code
- ▼ Seuls les écouteurs indépendants  
peuvent si nécessaire être supprimés





# **Chapitre 8 :**

## **Exercice Pion en cage**

### **Partie 1**

## Exercice Pion en cage : Partie 1

Vous pourrez guider  
au clavier un pion  
dans un damier

D'abord, créer le  
damier et placer le  
pion dans sa position  
initiale...

# JavaScript : programmation événementielle et asynchrone

Page 27

## Ce qu'il faut **retenir**



- ▼ Pour construire une structure à deux dimensions, il faut une boucle imbriquée dans une autre
- ▼ La modélisation front-end implique toujours un double travail :
  - ▶ Gérer les variables du modèle JS
  - ▶ Répercuter leur état dans le DOM



# **Chapitre 9 :**

## **Exercice Pion en cage**

### **Partie 2**

## Exercice Pion en cage – Partie 2

Le damier est  
prêt

Le pion est à sa  
position initiale

Il faut  
maintenant  
écouter les  
flèches du clavier  
et réagir

## Ce qu'il faut **retenir**

- ▼ Les saisies clavier s'écoutent sur window
- ▼ L'événement keydown écoute les frappes clavier
- ▼ Sa propriété key est facile à utiliser
- ▼ Et encore une fois...  
La modélisation front-end implique toujours un double travail :
  - ▶ Gérer les variables du modèle JS
  - ▶ Répercuter leur état dans le DOM





# **Chapitre 10 : Programmation asynchrone**

# JavaScript : programmation événementielle et asynchrone

Page 32

## Les promesses

Un écouteur est  
parfait pour des  
événements  
répétitifs MAIS...

- Un écouteur peut  
rater le  
chargement d'une  
ressource...
- ▼ Si elle est déjà  
présente au  
départ
- ▼ Si elle se charge  
trop rapidement

Une promesse  
est mieux  
adaptée parce  
qu'elle aboutit  
obligatoirement  
une et une seule  
fois



# JavaScript : programmation événementielle et asynchrone

Page 33

## Ce qu'il faut **retenir**

- ▼ Le constructeur Promise prend en paramètre une fonction qui reçoit elle-même deux fonctions en paramètres
- ▼ Lorsque la promesse aboutit, l'une des deux fonctions est appelée, avec en paramètre automatique la valeur d'aboutissement
- ▼ Les méthodes then() et catch() retournent une promesse
- ▼ L'ensemble est conçu pour le chaînage objet



# Chapitre 11 : L'API Fetch

# JavaScript : programmation événementielle et asynchrone

Page 35

## L'API Fetch

Un bon exemple de  
l'utilisation des  
promesses

Ici, pas de  
constructeur Promise :  
la méthode `fetch()`  
retourne une  
promesse exploitable

## Ce qu'il faut **retenir**



- ▼ L'API Fetch basée sur les promesses remplace les techniques AJAX basées sur XMLHttpRequest et les événements
- ▼ La méthode fetch() retourne une promesse avec la ressource en valeur d'aboutissement
- ▼ Gestion simplifiée des données à envoyer
- ▼ Gestion simplifiée des erreurs



# **Chapitre 12: Application Météo Partie 1**

## Application Météo

Créer une vue  
permettant d'afficher  
la météo courante  
dans les capitales de  
l'Union européenne

API OpenWeatherMap  
gratuite

# Application Météo : Partie 1

**JavaScript :  
programmation  
événementielle  
et asynchrone**

Page 39

Analyser la  
documentation OWM

Tester l'API

## Ce qu'il faut **retenir**

- ▼ Nombreuses API proposées sur le web
- ▼ Authentification nécessaire :
  - ▶ API Key
  - ▶ Token
  - ▶ Cookie...
- ▼ Différents formats :
  - ▶ JSON
  - ▶ XML
  - ▶ HTML...







# **Chapitre 13: Application Météo Partie 2**

## Application Météo : Partie 2

**JavaScript :  
programmation  
événementielle  
et asynchrone**

Page 42

Créer la vue  
HTML/CSS

Coder l'envoi à la  
demande

# JavaScript : programmation événementielle et asynchrone

Page 43

## Ce qu'il faut **retenir**



- ▼ En programmation asynchrone, le débogage est nettement plus compliqué
- ▼ L'onglet réseau de la console de développement permet de...
  - ▶ Vérifier la requête elle-même
  - ▶ Vérifier les données envoyées (payload)
  - ▶ Vérifier la réponse reçue
- ▼ Le débogage classique permet ensuite de...
  - ▶ Vérifier le traitement de la réponse
  - ▶ Vérifier le traitement de l'affichage



# **Chapitre 14:**

## **Application Météo**

### **Partie 3**

## Application Météo : Partie 3

**JavaScript :  
programmation  
événementielle  
et asynchrone**

Page 45

Traiter la réponse  
JSON

Afficher

# JavaScript : programmation événementielle et asynchrone

Page 46

## Ce qu'il faut **retenir**



- ▼ Ne pas oublier que l'API Fetch est asynchrone : tous les traitements doivent être encapsulés dans le corps de la lambda de then()
- ▼ Et pour la dernière fois...  
La modélisation front-end implique toujours un double travail :
  - ▶ Gérer les variables du modèle JS
  - ▶ Répercuter leur état dans le DOM



# **Chapitre 15 :**

## **Les fonctions asynchrones**

# Les fonctions asynchrones

**JavaScript :  
programmation  
événementielle  
et asynchrone**

Page 48

JavaScript permet de  
déclarer des fonctions  
asynchrones

Un mécanisme très  
proche des  
promesses...



# JavaScript : programmation événementielle et asynchrone

Page 49

## Ce qu'il faut **retenir**



- ▼ Le mot clé `async` rend une fonction asynchrone
- ▼ Une fonction asynchrone retourne toujours une promesse avec sa valeur de retour encapsulée en valeur d'aboutissement
- ▼ Le mot clé `await` arrête l'exécution pour attendre l'aboutissement de la promesse



# **Chapitre 16 : Application Météo Partie 4**

## Application Météo : Partie 4

- ▼ Remplacer le chaînage des méthodes par des fonctions asynchrones

## Ce qu'il faut **retenir**

- ▼ Conjuguées avec `await`, les fonctions asynchrones remplacent le chaînage des méthodes `then()` et `catch()` par un code synchrone plus habituel...
- ▼ Mais le traitement asynchrone est toujours là, il est simplement masqué par la syntaxe !



# Conclusion

# JavaScript : programmation événementielle et asynchrone

Page 54

## Conclusion

Félicitations, vous  
maîtrisez maintenant  
JavaScript sous tous  
ses aspects !

Mes conseils pour la suite  
:

- ▼ Étudier au moins un framework JS  
VueJS, ReactJS, Angular...
- ▼ Étudier la programmation  
back-end pour devenir un  
développeur full-stack
- ▼ Et pourquoi pas  
l'administration systèmes  
et réseaux pour devenir un  
DevOps ?