

JS

# La programmation en JavaScript

HTML/CSS/JavaScript (avancé)



# Introduction aux animations CSS

La programmation en JavaScript



# Pourquoi utiliser les animations CSS ?

- Les **animations CSS** permettent d'ajouter du dynamisme aux pages web.
- Elles sont utiles pour améliorer l'expérience utilisateur, attirer l'attention sur certains éléments et rendre l'interface plus fluide.
- **Exemples** d'utilisation :
  - Faire apparaître progressivement un titre ;
  - Ajouter une transition fluide entre différentes sections ;
  - Mettre en avant une interaction au survol.



# Les transitions CSS

- Les **transitions CSS** permettent de modifier progressivement une **propriété CSS** lorsqu'un élément change d'état (exemple : au survol ou au clic) :
- Syntaxe de base :



A dark-themed code editor window with a purple border. Inside, there are three grey dots at the top left and the file name "style.css" at the top right. The main area contains the following CSS code:

```
.element {  
    transition: propriété durée timing-function delay;  
}
```



# Exemple : Modifier la taille et la couleur au survol

...

style.css

```
.button {  
    background-color: #ff6600;  
    color: white;  
    padding: 10px 20px;  
    transition: background-color 0.3s ease-in-out, transform 0.2s ease;  
}  
  
.button:hover {  
    background-color: #ff3300;  
    transform: scale(1.1);  
}
```

Code source



# Explications

- O `background-color 0.3s ease-in-out` : La couleur change en 0.3 secondes avec un effet progressif.
- O `transform 0.2s ease` : L'élément grossit légèrement lors du survol.



# Les animations avec `@keyframes`

- Contrairement aux transitions, les **animations CSS** permettent de modifier un élément indépendamment d'un **événement** utilisateur :

```
● ● ● style.css

@keyframes nom-de-l-animation {
    0% {
        propriété: valeur-de-départ;
    }
    100% {
        propriété: valeur-de-fin;
    }
}

.element {
    animation: nom-de-l-animation durée timing-function delay iteration-count direction;
}
```



# Exemple : Faire apparaître un texte progressivement

```
style.css

@keyframes fadeIn {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}

.text {
  opacity: 0;
  animation: fadeIn 1.5s ease-in forwards;
}
```

[Code source](#)



# Explications

- L'opacité passe de `0` à `1` en `1.5s`.
- `ease-in` pour un effet progressif au démarrage.
- `forwards` permet de conserver l'état final après l'animation



# Exemple : Déplacer un titre de gauche à droite

```
... style.css

@keyframes slideIn {
  from {
    transform: translateX(-100%);
  }
  to {
    transform: translateX(0);
  }
}

h1 {
  animation: slideIn 1s ease-out forwards;
}
```

Code source



# Manipuler les animations CSS avec JavaScript

- JavaScript peut être utilisé pour activer ou désactiver une animation CSS dynamiquement.



# Exemple : Ajouter une classe animée au clic

...

index.html

```
<button id="animate-btn">Animer le titre</button>
<h1 id="title">Titre animé</h1>
```

Code source

...

style.css

```
@keyframes bounce {
  0%, 100% {
    transform: translateY(0);
  }
  50% {
    transform: translateY(-10px);
  }
}

.bounce {
  animation: bounce 0.5s ease-in-out;
}
```

Code source

...

script.js

```
document.getElementById("animate-btn").addEventListener("click", function() {
  document.getElementById("title").classList.toggle("bounce");
});
```

Code source



# Explications

- Le bouton ajoute ou enlève la **classe .bounce** au **h1**.
- L'animation **bounce** fait rebondir le **titre** brièvement.



# Bonnes pratiques

- Utiliser les **transitions** pour des effets simples (changements de couleur, d'opacité, d'échelle).
- Privilégier les **keyframes** pour des animations plus complexes (déplacements, changements successifs).
- Éviter d'animer les **propriétés** qui impactent la mise en page (préférer **transform** et **opacity** plutôt que **width** ou **height**).
- Toujours **tester** sur plusieurs **navigateurs** pour assurer la compatibilité.

JS

# Présentation de JavaScript

La programmation en JavaScript

# Introduction à JavaScript

- Un **langage de programmation** interprété, léger et polyvalent.
- Utilisé principalement pour le développement web côté **client (front-end)**.
- Permet de créer des pages web **interactives** et **dynamiques**.

# Caractéristiques de JavaScript

- Les **types de données** ne sont pas déclarés explicitement (par exemple, `let x = 20;`).
- Interprété par le **navigateur web** de l'utilisateur, ce qui permet une **interaction dynamique** sans recharger la page.

# Utilisation de JavaScript

- Réagit aux **clics, survols, saisies** au clavier, etc. pour interagir avec l'utilisateur
- Ajout, suppression et modification d'**éléments HTML** et **CSS** dans le **DOM** (pour **Document Object Model**) :
  - Pour vérifier les **saisies** utilisateur dans des **formulaires HTML** avant l'envoi au **serveur** ;
  - Pour créer des effets dynamiques comme les **carrousels**, les **menus burgers**, etc.
- Aujourd'hui, on peut utiliser dans des **frameworks** modernes JavaScript comme React, Angular, Vue.js pour construire des **applications web** à page unique (**SPA** pour **Single Page Application**).

# Environnement de développement

- Outils nécessaires :
  - **Éditeur de texte** ou **IDE** (Visual Studio Code, Sublime Text, etc.).
  - **Navigateur web** avec outils de développement intégrés (Chrome, Firefox, etc.).

# Communauté et Ressources

- Documentation et support :
  - [MDN Web Docs](#) pour une documentation complète.
  - Forums, communautés en ligne, et tutoriels pour l'apprentissage continu ([W3Schools](#) ou [CodinGame](#), par exemple).

# Configuration de l'environnement de développement

La programmation en JavaScript

# Utilisation des outils de développement

- **Inspection d'éléments HTML/CSS** (clic droit > **Inspecter** dans la plupart des navigateurs).
- **Console JavaScript** pour tester des **scripts** et voir les logs (`console.log()`) et les erreurs éventuelles (onglet « **Console** » de l'inspecteur du navigateur).
- Débogage de code JavaScript.

# Extensions pour VS Code

- **Live Server** : Permet un rafraîchissement en direct de la page web développée après une modification du code source.
- **ESLint** : Aide à détecter les **erreurs** et les problèmes de format dans le code **JavaScript**.
- **Prettier** : Formateur pour garder votre **code propre et lisible** (raccourci clavier : *Alt + Shift + F*)

# Pratiques recommandées

- **Tester fréquemment le code dans le navigateur :**
  - Vérifiez régulièrement comment votre code fonctionne dans le navigateur et ouvrez la **console** en cas de problème !
- **Se familiariser avec les outils de développement :**
  - Apprenez à utiliser la **console** et à déboguer (remonter à la source) pour résoudre les problèmes.

# Les balises `<script></script>`

- Un **script** correspond à une portion de code insérée dans une page **HTML**.
- Il est défini entre des **balises** `<script></script>` :

```
● ● ●  
<script>  
    // Code source JavaScript  
</script>
```

# Les balises <script></script>

- Il existe plusieurs méthodes pour intégrer un **script** à un **document HTML** :
  - Grâce aux **événements JavaScript** ;
  - Grâce à des **balises <script></script>** placées au sein du **document HTML** ;
  - En plaçant le code **JavaScript** dans un autre fichier (**meilleure pratique**).
- Pour insérer du code **JS** écrit dans un autre **fichier**, il suffit de spécifier le **chemin** pointant vers ce **fichier** (souvent, **script.js**) en **attribut src** des **balises <script></script>** :



```
<script src=".//scripts/script.js">  
</script>
```

JS

# Syntaxe de base

La programmation en JavaScript

# Les instructions et les commentaires

- Les **instructions** sont les commandes que vous demandez à réaliser à JavaScript (**peuvent** être terminée par un point-virgule).
- Comme tout langage de programmation, **JavaScript** peut inclure des **commentaires** dans son code par soucis de lisibilité et de maintenance.
- Commentaire sur une ligne : `// Ceci est un commentaire`
- Commentaire sur plusieurs lignes : `/* Ceci est un commentaire sur plusieurs lignes */`

# Les variables

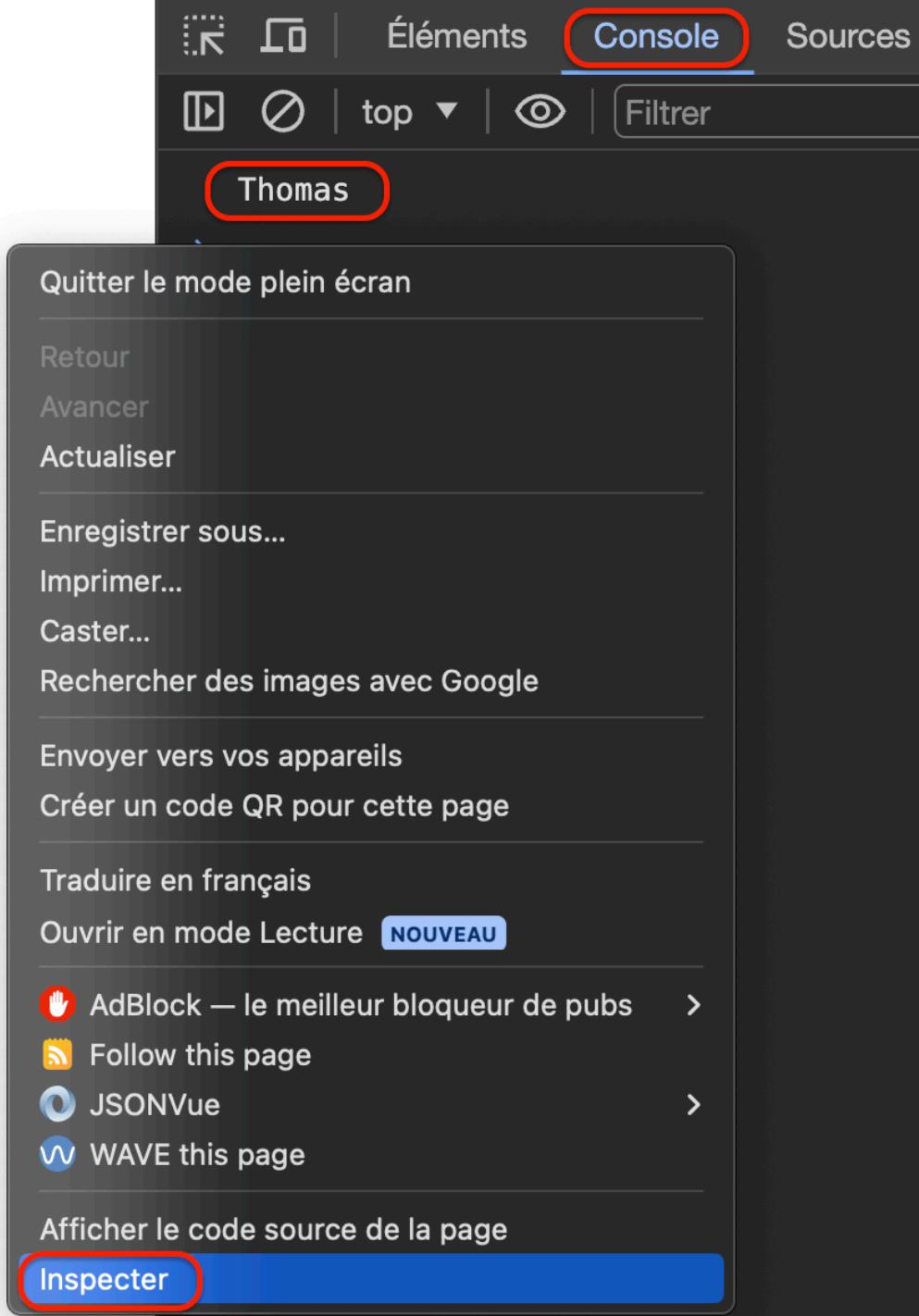
- Une **variable** permet de stocker des données pouvant être **modifiées** au cours de l'exécution du programme. Elle est reconnue par son **nom**.
- Les développeurs peuvent nommer leurs **variables** librement mais doivent respecter certaines **règles** pour que le **nommage** soit validé :
  - Un **nom de variable** doit commencer par une **lettre** (majuscule ou minuscule).
  - Un **nom de variable** peut comporter des **lettres** et des **chiffres**. Les espaces ne sont pas autorisés.
- JavaScript est **sensible à la casse** ainsi, `maVariable` et `MaVariable` sont considérées comme deux **variables** différentes.

# Variables et types de données

- Déclaration de variables :
  - Utilisation de `let` pour une **variable** et `const` pour une **constante**.
  - Exemples : `let age = 25 | const pi = 3.14`
- Types primitifs :
  - Les **nombres** (**entiers** ou à **virgule**),
  - Les **chaines de caractères** (entre **guillemets simples** ‘ ou **doubles** “),
  - Les **booléens** (`true` ou `false`),
  - `undefined`,
  - `null`.
- Structures de données : **Objet** (ou `object`) et **tableau** (ou `array`).

# Débogguer du code JavaScript

- Pour afficher ce que contient une **variable**, il est possible d'utiliser la **méthode `console.log()`**.
- Elle affiche ce qui lui est passé entre parenthèses (variable, texte, nombre, etc.) dans la **console** disponible dans l'inspecteur du **navigateur** : `console.log(firstName);`
- Ce code affiche la valeur contenue dans la **variable `firstName`** en **console**.
- Sur la plupart des navigateurs, la **console** est accessible avec *clic droit > Inspecter*.



# Opérateurs

- Opérateurs arithmétiques :
  - **Addition** (`+`), **soustraction** (`-`), **multiplication** (`*`), **division** (`/`), etc.
- Opérateurs d'affectation :
  - **Affectation de valeur** (`=`), **addition** (`+=`), **soustraction** (`-=`), etc.
- Opérateurs de comparaison :
  - **Égalité** (`==`), **inégalité** (`!=`), **strictement égal** (`===`), **strictement inégal** (`!==`), etc.
- Opérateurs logiques :
  - **ET** (`&&`), **OU** (`||`), **NON** (`!`).

JS

# Structures de contrôle

La programmation en JavaScript

# Instructions conditionnelles

- `if`, `else if` et `else` :
  - Permettent d'exécuter différents blocs de code en fonction de conditions spécifiques.

## ○ Syntaxe :



```
if (condition1) {  
    // Code à exécuter si condition1 est vraie  
} else if (condition2) {  
    // Code à exécuter si condition2 est vraie  
} else {  
    // Code à exécuter si aucune des conditions précédentes n'est  
    // vraie
```

# Exemple concret



```
let age = 20;
if (age < 18) {
    console.log("Vous êtes mineur.");
} else if (age >= 18 && age < 60) {
    console.log("Vous êtes adulte.");
} else {
    console.log("Vous êtes senior.");
}
```

# La boucle `for`

- Utilisée pour **exécuter un bloc d'instructions** un nombre déterminé de fois.
- **Syntaxe :**



```
for (initialisation; condition; incrémentation) {  
    // Code à exécuter à chaque itération  
}
```

# Exemple



```
for (let i = 1; i <= 5; i++) {  
    console.log(i);  
}
```

# Travailler avec des tableaux

La programmation en JavaScript

# Introduction aux tableaux

- Définition :
  - Un **tableau** est une **collection ordonnée de valeurs**, pouvant être de **différents types**.
  - Les **tableaux** sont utilisés pour stocker **plusieurs valeurs** dans **une seule variable**.
- Crédit d'un tableau :
  - **Syntaxe :** `let monTableau = [valeur1, valeur2, valeur3]`
  - **Exemple :** `let fruits = ["pomme", "banane", "cerise"]`

# Accès et modification des éléments

- Accès aux éléments :
  - Utilisation de l'**indice numérique** (de 0 à *taille du tableau - 1*) pour accéder à un élément.
  - Exemple : `console.log(fruits[0]); // affiche « pomme » en console`
- Modification des éléments :
  - Affecter une nouvelle **valeur** à un élément via son **indice numérique**.
  - Exemple : `fruits[1] = "orange"; // remplace "banane" par "orange"`

# Méthodes de base des tableaux

- Ajout et suppression d'éléments :
  - `monTableau.push(element)` : **Ajoute** un élément à la fin du tableau.
  - `monTableau.pop()` : **Supprime** le dernier élément du tableau.
  - `monTableau.unshift(element)` : **Ajoute** un élément au début du tableau.
  - `monTableau.shift()` : **Supprime** le premier élément du tableau.
  - **Exemple** : `fruits.push("mangue"); // ajoute "mangue" à la fin du tableau`
- Longueur d'un tableau :
  - `monTableau.length` : Retourne la longueur (le **nombre d'éléments**) du **tableau**.
  - **Exemple** : `console.log(fruits.length) // Affiche 4 en console`

# Parcourir un tableau

- Utilisation des **boucles** `for` ou `forEach` pour **parcourir** les **tableaux**.

- Exemple avec `for` :



```
for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}
```

- Exemple avec `forEach` :



```
fruits.forEach(function(fruit) {
    console.log(fruit);
});
```

# Trier un tableau

- Pour trier les éléments d'un **tableau** par ordre croissant (**alphabétique** ou **numérique**) JavaScript propose la **méthode sort()** :

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort(); // Trie le tableau par ordre alphabétique
```

- Dans la même idée, vous pouvez utiliser la **méthode reverse()** pour **inverser** le contenu du **tableau** ciblé :

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.reverse(); // Trie le tableau dans le sens inverse
```

# Déclaration et utilisation des fonctions

La programmation en JavaScript

# Fonctions nommées

- Une **fonction** correspond à un **sous-programme** permettant l'exécution d'un ensemble d'**instructions** d'un simple **appel** de celle-ci depuis le **programme principal**
- Elles permettent de **simplifier** le code et ainsi **diminuer la taille d'un programme**
- Une **fonction nommée** est déclarée avec un nom librement défini (en **camelCase**).



```
function nomDeLaFonction(param1, param2) {  
    // Code à exécuter  
}
```

# Exemple concret



```
function additionner(a, b) {  
    return a + b;  
}  
console.log(additionner(5, 3)); // Affiche 8
```

# Passage de paramètres et valeurs de retour

JS

- **Passage de paramètres :**

- Les **fonctions** peuvent recevoir des valeurs en **entrée** (entre les parenthèses) pour effectuer des opérations avec ces valeurs.

- **Valeurs de retour :**

- Le mot-clé `return` permet de renvoyer une valeur depuis la **fonction** vers le programme principal ;
  - Si aucune valeur n'est renvoyée, la **fonction** renvoie la (non-)valeur `undefined`.

# Portée des fonctions

- Les **variables** définies à l'intérieur d'une **fonction** ne sont accessibles que dans cette fonction (portée locale).
- Les **fonctions** peuvent accéder aux **variables** définies en dehors de leur corps (portée globale).
- Cela signifie que, dans l'exemple précédent, les **variables** **a** et **b** ne sont accessibles qu'au sein de la **fonction additionner()**.

# Fonctions anonymes

- Une fonction sans nom, souvent utilisée comme une expression.
- Peut être affectée à une variable.
- **Syntaxe :**



```
const maFonction = function(param1, param2) {  
    // Code à exécuter  
};
```

# Exemple concret



```
const surfaceCercle = function(rayon) {  
    return Math.PI * rayon * rayon;  
};  
console.log(surfaceCercle(4)); // Affiche la surface pour un rayon de 4
```

# Fonctions fléchées (ou arrow functions)

- Introduites plus récemment, elles offrent une **syntaxe plus courte**.
- Pas de mot-clé **function**, et un **return** implicite pour les fonctions sur une unique ligne.
- **Syntaxe :**



```
const nomDeLaFonction = (param1, param2) => {
    // Code à exécuter
};
```

# Exemple concret



```
const multiplier = (a, b) => a * b;  
console.log(multiplier(2, 6)); // Affiche 12
```

JS

# Introduction aux objets

La programmation en JavaScript

# Comprendre les objets

- Un **objet** en JavaScript est une **variable** stockant plusieurs **propriétés**, où chaque **propriété** est une association entre une **clé** (chaîne de caractères) et une **valeur** (type libre).
- Les **objets** permettent de regrouper des données (**propriétés**) et des **fonctions** liées (**méthodes**) dans une même variable :



```
let monObjet = {  
    propriete1: valeur1,  
    propriete2: valeur2,  
    // ...  
};
```



```
let voiture = {  
    marque: "Toyota",  
    modele: "Corolla",  
    annee: 2020  
};
```

# Accès et modification des propriétés

## ○ Accès aux propriétés :

- Utiliser la notation pointée : `monObjet.propriete` (à privilégier) ;
- Ou la notation par crochets : `monObjet[ 'propriete' ]` (! confusion possible avec les tableau) ;
- **Exemple :** `console.log(voiture.modele) // Affiche « Corolla » en console.`

## ○ Modification des propriétés :

- Affecter une nouvelle valeur à une propriété existante ;
- **Exemple :** `voiture.annee = 2021`

## ○ Ajout de nouvelles propriétés :

- Simplement en affectant une valeur à une nouvelle clé (chaîne de caractères) ;
- **Exemple :** `voiture.couleur = "bleue"`

# Parcourir un objet

- La **boucle** `for...in` permet de **parcourir** toutes les **propriétés** d'un **objet**.
- **Exemple :**



```
for (let cle in voiture) {  
    console.log(cle + ": " + voiture[cle]);  
}
```

# Des questions ?