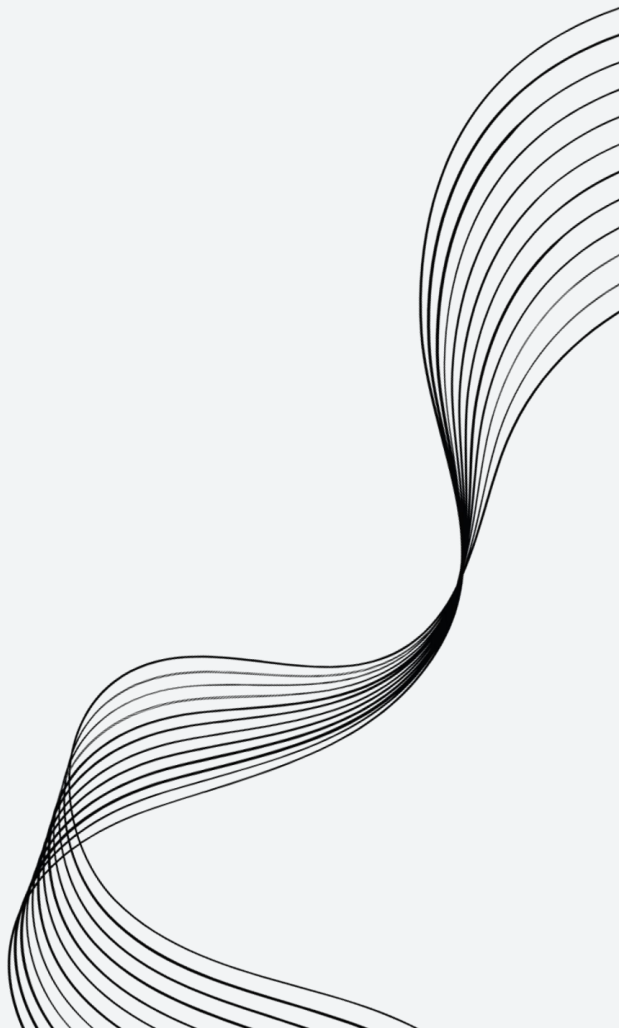
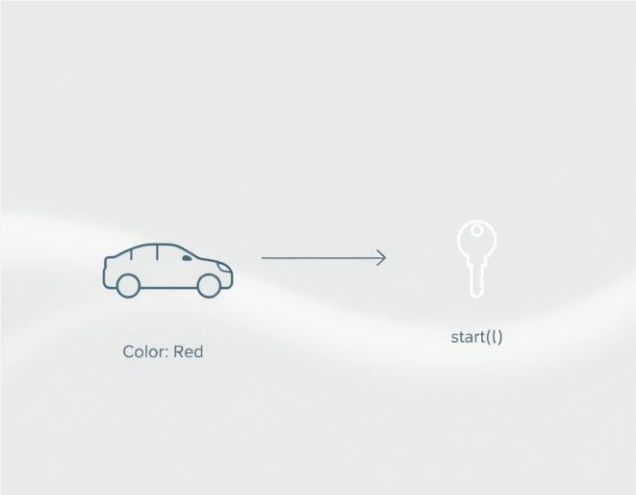
Introduction aux Objets en JavaScript

Une exploration des structures de données fondamentales au cœur de JavaScript.



Qu'est-ce qu'un Objet en JavaScript ? (Analogie)



Imaginez un objet du monde réel, comme une voiture. Cette voiture a des caractéristiques (propriétés) comme sa couleur, sa marque ou son poids. Elle a aussi des actions qu'elle peut effectuer (méthodes), comme démarrer, accélérer ou freiner. En JavaScript, un objet est une collection de ces paires 'caractéristique: valeur' et 'action: fonction'.

Création d'Objets: Notions de Base

La manière la plus simple de créer un objet est d'utiliser la syntaxe littérale, avec des accolades `{}`. À l'intérieur, on définit des paires clé-valeur.

Structure Clé-Valeur

Chaque propriété d'un objet est une 'clé' (un nom que vous donnez) associée à une 'valeur'. Les clés sont des chaînes de caractères, et les valeurs peuvent être de n'importe quel type de données : un nombre, une chaîne de caractères, un booléen, un tableau, ou même un autre objet.

Exemple de Code

```javascript let personne = { nom: "Romain", age: 18, estEtudiante: false, ville: "New York" }; ```
```

Types d'Objets Intégrés (Built-in Objects)

JavaScript fournit plusieurs objets standards qui servent de base pour construire des applications complexes.

Object

L'ancêtre de presque tous les objets en JavaScript. Fournit des fonctionnalités de base.

Array

Un type spécial d'objet pour stocker des collections ordonnées de données.

Date

Pour travailler avec des dates et des heures.

Math

Fournit des propriétés et des méthodes mathématiques (PI, sqrt(), random(), etc.).

String, Number, Boolean

Bien que primitives, elles peuvent se comporter comme des objets grâce à des 'wrappers' d'objets.

Propriétés d'un Objet: Accès et Modification

Une fois un objet créé, vous pouvez facilement lire et changer la valeur de ses propriétés.

Accéder aux Propriétés

On utilise la notation par point (`objet.propriete`) ou par crochets (`objet['propriete']`). La notation par crochets est utile lorsque le nom de la propriété est dans une variable.

```javascript // Notation par point console.log(personne.nom); // "leo" // Notation par crochets console.log(personne['age']); // 108 ```

Modifier les Propriétés

Il suffit d'assigner une nouvelle valeur à la propriété existante.

```javascript // Modification de l'âge personne.age = 31; console.log(personne.age); // 11 // Ajout d'une nouvelle propriété personne.email = "leo@email.com"; console.log(personne.email); ```

Méthodes d'un Objet: Définition et Appel

Une méthode est simplement une propriété dont la valeur est une fonction. Elle définit une action que l'objet peut effectuer.

Définir une Méthode

```javascript
let voiture = {
  marque: "Tesla",
  modele: "Model 3",
  demarrer: function() {
    console.log("La voiture démarre !");
  }
};
```

Appeler une Méthode

On l'appelle comme une fonction, en utilisant la notation par point suivie de parenthèses `()`.

```javascript
voiture.demarrer();
// Affiche dans la console:
// "La voiture démarre !"
```

Le Contexte 'this' dans les Objets

Le mot-clé `this` est une référence spéciale. Dans une méthode d'objet, `this` fait référence à l'objet lui-même, permettant à la méthode d'accéder aux autres propriétés du même objet.

```javascript let utilisateur = {  nom: "Mathias",

 salutation: function()

{  // 'this.nom' fait référence à la propriété 'nom' de l'objet 'utilisateur' console.log("Bonjour, je m'appelle " + this.nom);  } }; utilisateur.salutation(); // Affiche "Bonjour, je m'appelle Mathias" ```
```

Objets Littéraux vs. Constructeurs

Objets Littéraux `{}`

Parfait pour créer un seul objet, unique. C'est simple, rapide et lisible.

```javascript
let chat = {
  nom: "Mimi",
  cri: "Miaou"
};
```

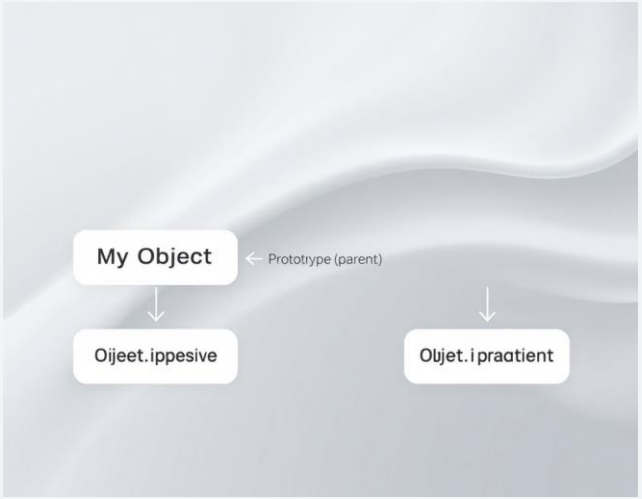
Fonctions Constructeurs

Idéal pour créer plusieurs objets du même type, comme un modèle ou un plan. On utilise le mot-clé `new`.

```javascript
function Animal(nom, cri) {
  this.nom = nom;
  this.cri = cri;
}

let chien = new Animal("Rex", "Wouf");
let oiseau = new Animal("Titi", "Cui-cui");
```

Le Prototypage des Objets (Analogie)



Imaginez une chaîne de prototypes comme une recherche de définitions. Si un objet ne possède pas une propriété ou une méthode, il va la chercher dans son 'prototype' (son modèle parent). Si ce dernier ne l'a pas non plus, il cherche dans le prototype de son parent, et ainsi de suite, jusqu'à atteindre l'objet de base. C'est le principe de la chaîne de prototypes.

Héritage Basé sur les Prototypes

L'héritage prototypal permet à un objet d'accéder aux propriétés et méthodes d'un autre objet. Cela favorise la réutilisation du code.

```javascript
// Objet parent avec une méthode
const animal = {
  parler: function() {
    console.log(this.cri);
  }
};

// Crée un nouvel objet 'chien' qui hérite de 'animal'
const chien = Object.create(animal);
chien.cri = "Wouf";

chien.parler(); // Affiche "Wouf". La méthode 'parler' est héritée de 'animal'.
```

Manipulation: Ajout/Suppression de Propriétés

Ajouter Dynamiquement

On peut ajouter une propriété ou une méthode à un objet à tout moment après sa création.

```javascript
let produit = { nom: "Livre" };

// Ajout d'une propriété
produit.prix = 15;

// Ajout d'une méthode
produit.afficher = function() {
  console.log(this.nom);
};
```

Supprimer avec `delete`

L'opérateur `delete` permet de supprimer une propriété d'un objet.

```javascript
let contact = {
  nom: "Jean",
  email: "jean@test.com",
  temporaire: true
};

delete contact.temporaire;

console.log(contact);
// { nom: "Jean", email: "jean@test.com" }
```

Parcourir les Propriétés d'un Objet

Il existe plusieurs façons d'itérer sur les propriétés d'un objet pour les lire ou les manipuler.

1 **Boucle `for...in`**

Permet de parcourir toutes les propriétés énumérables d'un objet. `for (let cle in objet) { ... }`

2 **Object.keys()**

Retourne un tableau contenant les noms (clés) des propriétés de l'objet. On peut ensuite boucler sur ce tableau.

3 **Object.values()**

Retourne un tableau contenant les valeurs des propriétés de l'objet.

4 **Object.entries()**

Retourne un tableau de paires [clé, valeur] pour chaque propriété de l'objet.

Objets Mutables vs. Immutables

Mutable (Modifiable)

Par défaut, les objets sont mutables. Cela signifie que leur état (leurs propriétés) peut être modifié après leur création. C'est flexible mais peut introduire des bugs si un objet est modifié de manière inattendue.

Immutable (Non-modifiable)

Un objet immutable ne peut pas être changé. Pour le 'modifier', on en crée une nouvelle copie avec les changements. Cela rend le code plus prévisible. On peut utiliser `Object.freeze()` pour rendre un objet immutable.

Cas d'Utilisation Avancés des Objets

- **Namespacing :** Regrouper des fonctions et variables liées dans un objet pour éviter de polluer l'espace de noms global.
- **Objets de configuration :** Passer un seul objet avec de nombreuses options à une fonction, ce qui est plus propre que de nombreux arguments.
- **Structures de données :** Servir de base pour des structures plus complexes comme les dictionnaires (hash maps) ou les files d'attente.
- **Modélisation de données :** Représenter des entités complexes récupérées depuis une API (utilisateurs, produits, articles).

Conclusion et Bonnes Pratiques

Points Clés à Retenir

- **Simplicité avant tout :** Utilisez la syntaxe littérale `{}` pour créer des objets simples.
- **Maîtrisez `this` :** Comprendre comment le contexte de `this` change est crucial, surtout avec les fonctions fléchées.
- **Structure claire :** Donnez des noms de propriétés et de méthodes clairs et cohérents.
- **Préférez l'immutabilité :** Lorsque c'est possible, traitez les objets comme immutables pour un code plus sûr et prévisible.
- **Utilisez le prototypage à bon escient :** L'héritage est puissant mais peut complexifier le code. Les classes (qui sont une surcouche syntaxique) sont souvent plus claires.