INTRODUCTION AUX CLASSES

DÉFINITION D'UNE CLASSE

SYNTAXE

Une **classe** est définie avec le mot-clé `class`, suivi du nom de la classe.

```
class ClassName {
  // code de la classe
}
```

EXEMPLES D'UTILISATION

```
class Voiture {
  // code de la classe Voiture
}
```

CONSTRUCTEUR

SYNTAXE

Un **constructeur** est une méthode spéciale pour initialiser les objets d'une **classe**. Il est défini avec le mot-clé `constructor`.

```
class ClassName {
  constructor( /* paramètre(s) */ ) {
    // initialisation des propriétés
  }
}
```

EXEMPLES D'UTILISATION

```
class Voiture {
  constructor(marque, modele) {
    this.marque = marque;
    this.modele = modele;
  }
}
```

PROPRIÉTÉS D'UNE CLASSE

SYNTAXE

Les **propriétés** sont les variables d'un objet, définies à l'intérieur de la **classe**.

```
class ClassName {
  constructor( /* paramètre(s) */ ) {
    this.propertyName = /* valeur */
  }
}
```

EXEMPLES D'UTILISATION

```
class Voiture {
  constructor(marque, modele) {
    this.marque = marque
    this.modele = modele
  }
}
const voiture1 = new Voiture("Toyota", "Corolla")
console.log(voiture1.marque) // Affiche "Toyota"
console.log(voiture1.modele) // Affiche "Corolla"
```

MÉTHODES DE CLASSE

DÉFINITION D'UNE MÉTHODE

Les **méthodes** sont des **fonctions** définies à l'intérieur d'une **classe**, qui peuvent accéder aux **propriétés** et **méthodes** de cette classe.

```
class Personne {
  constructor(nom, age) {
    this.nom = nom;
    this.age = age;
  }

  afficherNom() {
    console.log("Nom :", this.nom);
  }
}

const personne1 = new Personne("Alice", 30);
personne1.afficherNom(); // Affiche "Nom : Alice"
```

SYNTAXE

```
class MaClasse {
  maMethode() {
    // Code de la méthode
  }
}
```

EXEMPLES D'UTILISATION

```
class Voiture {
  constructor(marque) {
    this.marque = marque;
  }

  demarrer() {
    console.log(`La ${this.marque} démarre`);
  }
}

const maVoiture = new Voiture('Toyota');
maVoiture.demarrer();
```

MÉTHODES STATIQUES

Les **méthodes statiques** sont des méthodes qui appartiennent directement à la **classe** et non à une **instance** de la classe.

```
class MaClasse {
  static maMethodeStatique() {
    console.log("Ceci est une méthode statique");
  }
}

MaClasse.maMethodeStatique(); // Appel de la méthode statique
```

SYNTAXE

```
class MaClasse {
  static maMethodeStatique() {
    // Code de la méthode statique
  }
}
```

EXEMPLES D'UTILISATION

```
class Calculateur {
  static addition(a, b) {
    return a + b;
  }
}

const resultat = Calculateur.addition(5, 3);
console.log(resultat);
```

MÉTHODES D'INSTANCE

Les **méthodes d'instance** sont des méthodes qui appartiennent à une **instance** de la classe et non à la **classe** elle-même.

```
class Exemple {
  constructor(nom) {
    this.nom = nom;
  }

  saluer() {
    console.log(`Bonjour, je suis ${this.nom}`);
  }
}

const exemple1 = new Exemple("Alice");
exemple1.saluer(); // Affiche "Bonjour, je suis Alice"
```

SYNTAXE

```
class MaClasse {
  maMethodeInstance() {
    // Code de la méthode d'instance
  }
}
```

EXEMPLES D'UTILISATION

```
class CompteBancaire {
  constructor(solde) {
    this.solde = solde;
  }

  deposer(montant) {
    this.solde += montant;
  }

  retirer(montant) {
    this.solde -= montant;
  }
}
```

HÉRITAGE ET CLASSES DÉRIVÉES

HÉRITAGE

L'**héritage** en JavaScript permet à une classe d'**étendre** une autre classe et d'**hériter** de ses propriétés et méthodes.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(`${this.name} fait un bruit.`);
  }
}

class Chien extends Animal {
  constructor(name) {
    super(name);
  }
  speak() {
    console.log(`${this.name} aboie.`);
```

SYNTAXE

```
class ClasseDerivee extends ClasseDeBase {
  // code de la classe dérivée
}
```

EXEMPLES D'UTILISATION

```
class Animal {
  constructor(nom) {
    this.nom = nom;
  }

  parler() {
    console.log(`${this.nom} fait du bruit.`);
  }
}

class Chat extends Animal {
  parler() {
    console.log(`${this.nom} fait miaou.`);
  }
}
```

MOT-CLÉ `extends`

Le mot-clé **`extends`** est utilisé pour créer une classe dérivée à partir d'une **classe de base**.

```
class ClasseDeBase {
  constructor() {
    // Code de la classe de base
  }
}

class ClasseDerivee extends ClasseDeBase {
  constructor() {
    super(); // Appeler le constructeur de la classe de base
    // Code de la classe dérivée
  }
}
```

SYNTAXE

```
class ClasseDerivee extends ClasseDeBase {
  // code de la classe dérivée
}
```

Note : L'héritage de classe permet de créer une nouvelle classe qui hérite des propriétés et méthodes d'une autre classe appelée classe de base. La classe dérivée peut alors étendre ou modifier certaines fonctionnalités de la classe de base.

EXEMPLES D'UTILISATION

```
class Oiseau extends Animal {
  voler() {
    console.log(`${this.nom} peut voler.`);
  }
}

let oiseau = new Oiseau("Piaf");
oiseau.parler();  // Piaf fait du bruit.
oiseau.voler();   // Piaf peut voler.
```

MOT-CLÉ `super`

Le mot-clé **`super`** est utilisé pour appeler une méthode de la **classe de base** depuis la **classe dérivée**.

```
class ClasseBase {
  constructor() {
    console.log("Constructeur de la classe de base");
  }
}

class ClasseDerivee extends ClasseBase {
  constructor() {
    super(); // Appelle le constructeur de la classe de base
    console.log("Constructeur de la classe dérivée");
  }
}

const obj = new ClasseDerivee();
```

SYNTAXE

```
class ClasseDerivee extends ClasseDeBase {
  maMethode() {
    super.methodeDeBase();
  }
}
```

EXEMPLES D'UTILISATION

```
class Chien extends Animal {
  constructor(nom, race) {
    super(nom); // Appel du constructeur de la classe de base
    this.race = race;
  }

  parler() {
    super.parler(); // Appel de la méthode parler de la classe de base
    console.log(`${this.nom} appartient à la race ${this.race}.`);
  }
}

let chien = new Chien("Médor", "Bulldog");
chien.parler();  // Médor fait du bruit. Médor appartient à la race Bulldog.
```

ENCAPSULATION EN JAVASCRIPT

GETTERS ET SETTERS

SYNTAXE

Les **getters** et **setters** permettent de contrôler l'accès aux propriétés d'une classe:

```
class MyClass {
  constructor(name) {
    this._name = name;
  }

  get name() {
    return this._name;
  }

  set name(value) {
    this._name = value;
  }
}
```

EXEMPLES D'UTILISATION

```
let obj = new MyClass("John");
console.log(obj.name); // "John"
obj.name = "Jane";
console.log(obj.name); // "Jane"
```

PROPRIÉTÉS ET MÉTHODES PRIVÉES

SYNTAXE

Les propriétés et méthodes **privées** sont déclarées avec un dièse (#):

```
class MyClass {
  #privateProperty = "secret";
  #privateMethod() {
    console.log(this.#privateProperty);
  }

  usePrivateMethod() {
    this.#privateMethod();
  }
}
```

EXEMPLES D'UTILISATION

```
let obj = new MyClass();
obj.usePrivateMethod(); // "secret"
// obj.#privateMethod(); // Erreur: Private field '#privateMethod' must be declared in an enclosing class
```