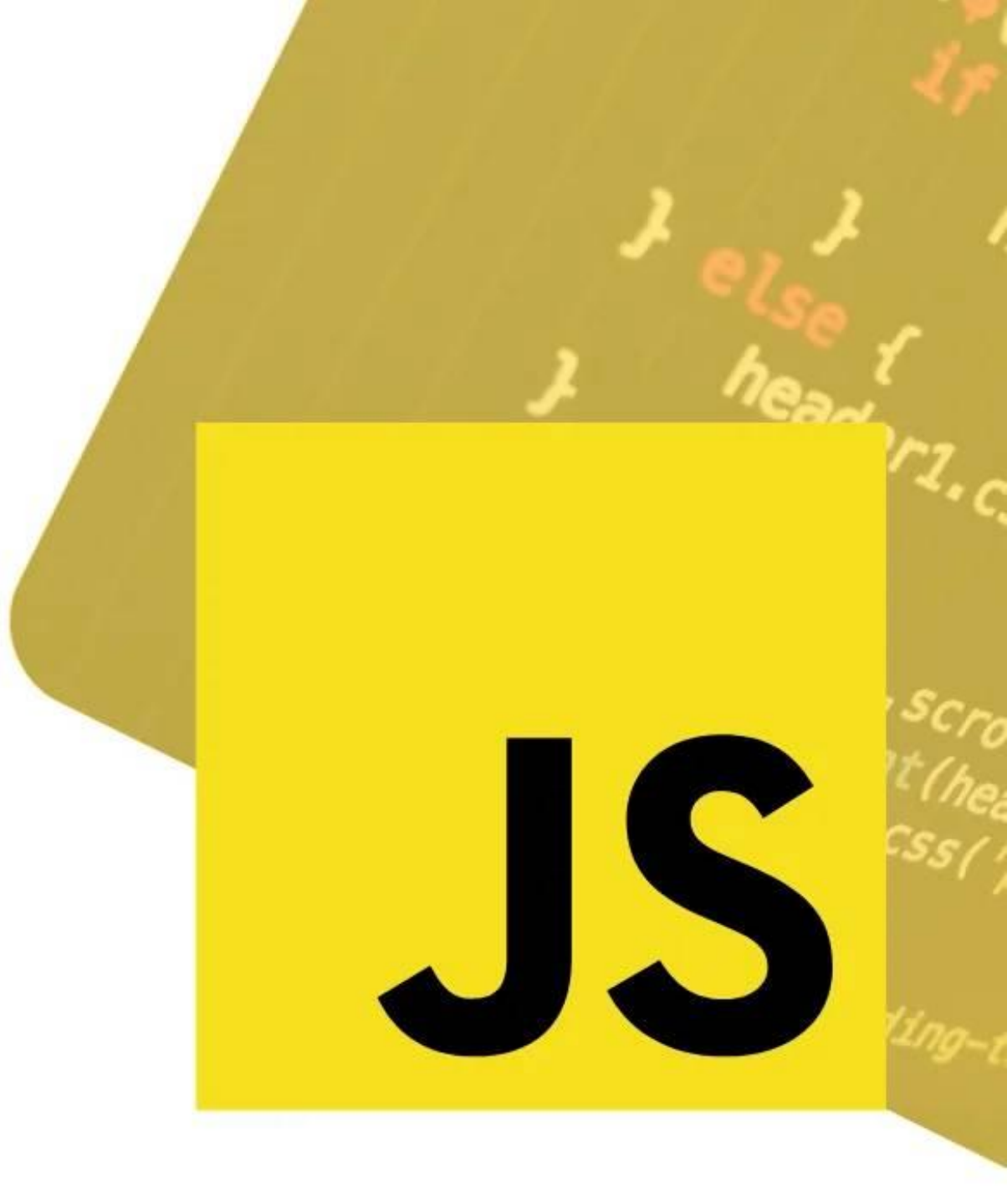


# **INTRODUCTION AUX CLASSES ET AUX OBJETS (ES6)**

The image features a yellow square with the letters 'JS' in a bold, black, sans-serif font. This square is positioned in the lower right area of the frame. Behind it and to the left is a larger, semi-transparent olive-green shape. The background of the entire image is white, with faint, stylized JavaScript code snippets in yellow and orange text visible in the upper right and bottom right corners. The code includes keywords like 'if', 'else', 'head', 'r1.c', 'scro', 't(hea', 'css(', and 'ding-t'.

# Classe VS Prototype

- Il existe deux grands types de langages orientés objet : ceux basés sur **les classes**, et ceux basés sur les **prototypes**
- Le JavaScript est un langage mono-thread **orienté objet** basé sur la **notion de prototypes**
- Il existe de grandes **différences conceptuelles** entre les langages orientés objet basés sur les classes et ceux basés sur les prototypes.
- Dans les langages basés sur les classes, tous les objets sont créés en faisant **une instance** des classes
- Une classe contient toutes les **définitions des propriétés** et **méthodes** dont dispose un objet. On ne peut pas ensuite **rajouter ou supprimer** des membres à un objet dans les langages **basés sur les classes** ;
- Dans les langages basés sur les classes, l'héritage se fait en définissant des **classes mères** et des **classes étendues ou classes enfants**.

```
<?php
class Utilisateur{
    // $user_name et $user_age sont des propriétés
    protected $user_name;
    protected $user_age;

    /* __construct() est une méthode constructeur. Elle va être appelée
    * dès qu'on va instancier la classe c'est-à-dire créer un nouvel
    * objet à partir de la classe. Ici, notre constructeur se charge
    * d'initialiser les propriétés $user_name et $user_age de l'objet créé
    */
    public function __construct($nom, $age){
        $this->user_name = $nom;
        $this->user_age = $age;
    }

    /* Ceci est une autre méthode de la classe qui retourne la valeur de
    * la propriété $user_name de l'objet qui l'appelle */
    public function getNom(){
        return $this->user_name;
    }
}
```

# Les classes avec JavaScript

- JavaScript est toujours un langage orienté objet à **prototypes** et en tâche de fond, il va **convertir** nos « classes » selon son **modèle prototypes**.
- On utilise le mot réservé **\*\*class\*\*** suivi du nom de la classe elle-même pour la définir. La classe est **un modèle** à partir duquel on peut **créer autant d'objets qu'on le souhaite**, appelés **\*\*instances\*\***. Pour créer une nouvelle instance, on utilise le **mot-clé new** suivi du **nom de classe**.
- Lors de l'instanciation avec le **mot-clé new**, une méthode spéciale de la classe est appelée le **\*\*constructeur\*\***. En JS, le constructeur porte le nom réservé **constructor**, et les **\*\*attributs** (ou propriétés)**\*\*** de la classe y sont définis. On attache les attributs à l'instance de classe à l'aide du mot **clé this** :
- **Pas d'encapsulation** : JS n'implémente pas de visibilité **private** (...pas avant ES2019 en tout cas !), tous les attributs et toutes les méthodes sont accessibles dans le script courant, c'est-à-dire **public et accessible de partout**

```
class Ligne{
  /*Nous n'avons pas besoin de préciser "function" devant notre constructeur
  *et nos autres méthodes classe*/
  constructor(nom, longueur){
    this.nom = nom;
    this.longueur = longueur;
  }

  taille(){document.getElementById('p1').innerHTML +=
    'Longueur de ' + this.nom + ' : ' + this.longueur + '<br>'};
}

let geo1 = new Ligne('geo1', 10);
let geo2 = new Ligne('geo2', 5);
geo1.taille();
geo2.taille();
```

```
class Rectangle{

  constructor(w, h){
    // attributs de classe
    this._w = w;
    this._h = h;

    this.name = "Rectangle";
  }

  // ...
}

// création de deux instances
const unRectangle = new Rectangle(30, 50);
const unAutreRectangle = new Rectangle(20, 35);
```

# Encapsulation ? :

- On peut tout de fois faire "comme si" une propriété **était privée**, ne la manipuler directement que **depuis l'intérieur** de la classe, et définir des accesseurs **\*\*setter\*\*** et **\*\*getter\*\*** pour y accéder en **écriture et en lecture**. Il faudra cependant faire attention au nom de ces attributs dans la classe, une technique classique consiste à préfixer leurs noms **par un underscore**.
- On définira les propriétés "**privées**" en les préfixant d'un **underscore**. Elles seront appelées, une fois **les setter et getter** définis, dans le script courant sans **l'underscore** et seront traitées comme des variables attachées à l'objet donc public
- Les **\*\*méthodes\*\*** de classe sont des simples méthodes nommées; voyez l'exemple de la méthode **\*\*dim\*\*** ci-dessous dans la classe Rectangle :

```
class Rectangle{
  // constructeur
  constructor(w, h){
    this._w = w;
    this._h = h;
  }

  // EXEMPLE DE METHODE DE CLASSE
  dim(){
    return `Width : ${this._w} Height : ${this._h}`;
  }

  // getter = accesseur
  get area(){
    return this._w * this._h;
  }

  // setter = mutateur
  set w(w){
    this._w = w;
  }
  // Autre mutateur
  set h(h){
    this._h = h;
  }
}

let r1 = new Rectangle(10, 2);
// 20
console.log(r1.area);

r1.w = 100;
r1.h = 30;

// 3000
console.log(r1.area);

console.log(r1.dim())
```

# Classes étendues et héritage en JavaScript

- Vous pouvez spécialiser une **classe en héritant** d'une classe **parente plus générale**.
- Rappelez-vous du principe de l'héritage en objet : **une classe fille \*\*est une sorte de\*\*** par rapport à la **classe mère**.
- Par exemple, un Lion est une sorte d'Animal. Dans ce cas la **classe Lion** est la **classe fille** de la **classe Animal**. C'est une **spécialisation** de la classe Animal.
- Pour définir une classe étendue vous devez utiliser le mot clé **extends**.
- Le mot clé **\*\*super\*\*** permet de faire passer des valeurs au **constructeur de la classe mère**. Attention, si vous êtes dans une classe dérivée (**fille**) et que vous définissez un constructeur de classe, vous êtes obligé d'utiliser **super** pour accéder au **constructeur de la classe mère**.
- Si toutefois vous ne définissez pas de constructeur dans votre classe dérivée, le constructeur de la classe mère **est automatiquement appelé**.
- Notez enfin que si vous définissez des attributs de classe dans le constructeur de votre **classe dérivée**, vous devez le faire après la méthode **super**, JS bloquera la compilation si ce principe de syntaxe n'est pas respecté.

# Exemple d'héritage simple:

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

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

  speak(){
    return `Name : ${this._name}`;
  }
}

class Lion extends Animal {
  constructor(name) {
    // écrivez super avant la définition des nouvelles propriétés de classe
    super(name);
    this.force = 100;
  }
}

let lion = new Lion("Shere")
lion.name = "Shere Khan";

console.log(lion.speak())
```

# Avenir : Méthodes privées (expérimentales)

- Vous pouvez définir des attributs privés dans une classe JS de la manière suivante; ces attributs ne seront accessibles qu'à l'intérieur de la classe. Attention, les champs ou attributs privés doivent être déclarés en premier dans la déclaration des champs.

```
class Rectangle {  
  #h = 0;  
  #w = 0;  
  constructor(h, w){  
    this.#h = h;  
    this.#w = w;  
  }  
}
```