# LES OBJETS JS

Jusqu'à présent, vous avez utilisé des primitives et des tableaux dans votre code. Et vous avez abordé le codage d'une manière assez procédurale en utilisant des instructions simples, des conditionnels et des boucles for/while avec des fonctions - ce n'est pas exactement orienté objet.

L'utilisation d'objets va vous rendre la vie plus facile - enfin, meilleure au sens de la programmation.

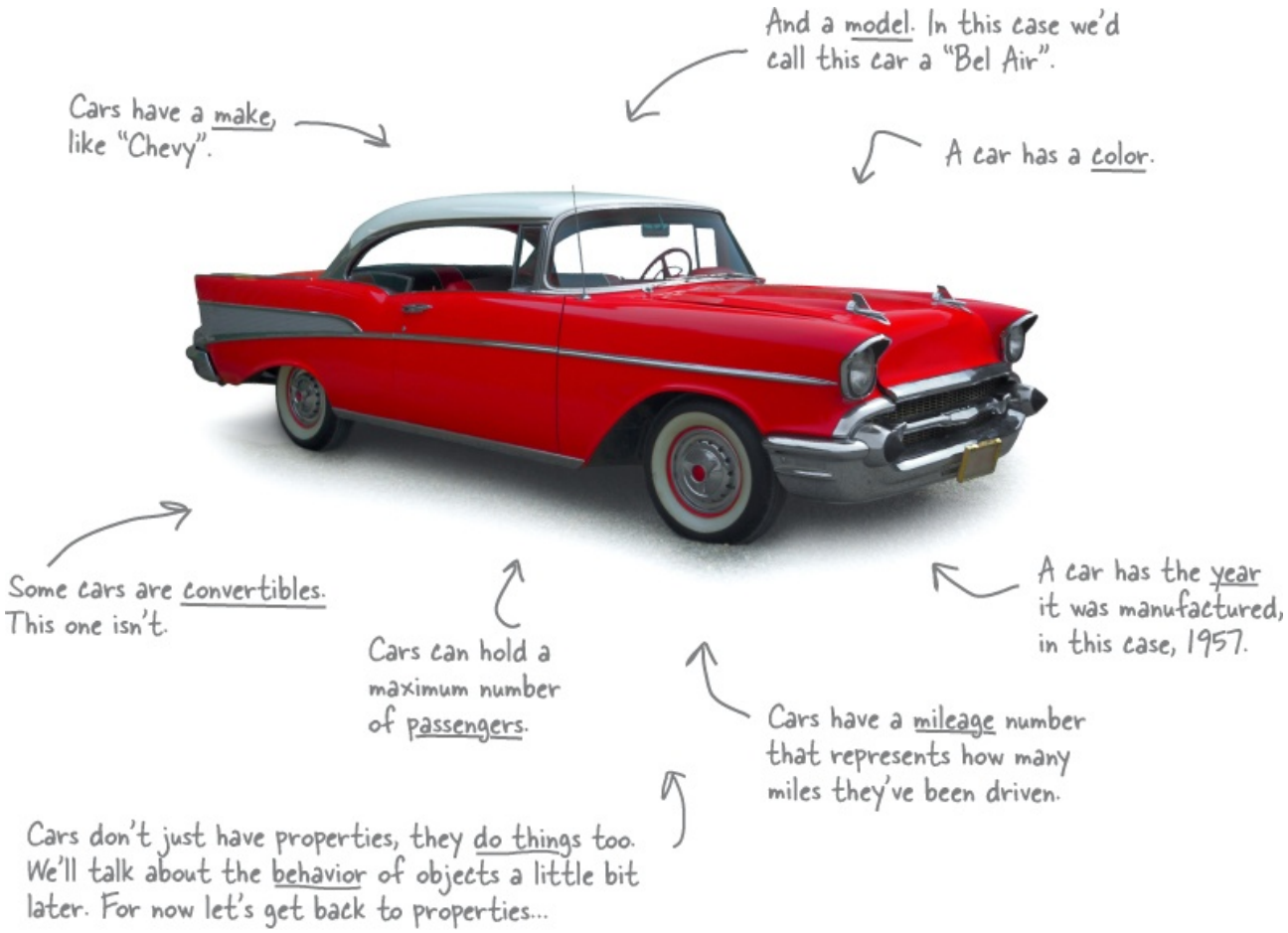



figure extraite de Javascript Head First

Bien sûr, une voiture réelle possède bien plus que ces quelques propriétés, mais pour les besoins du codage, ce sont les propriétés que nous voulons avoir dans notre application. Réfléchissons à ces propriétés en termes de types de données JavaScript:

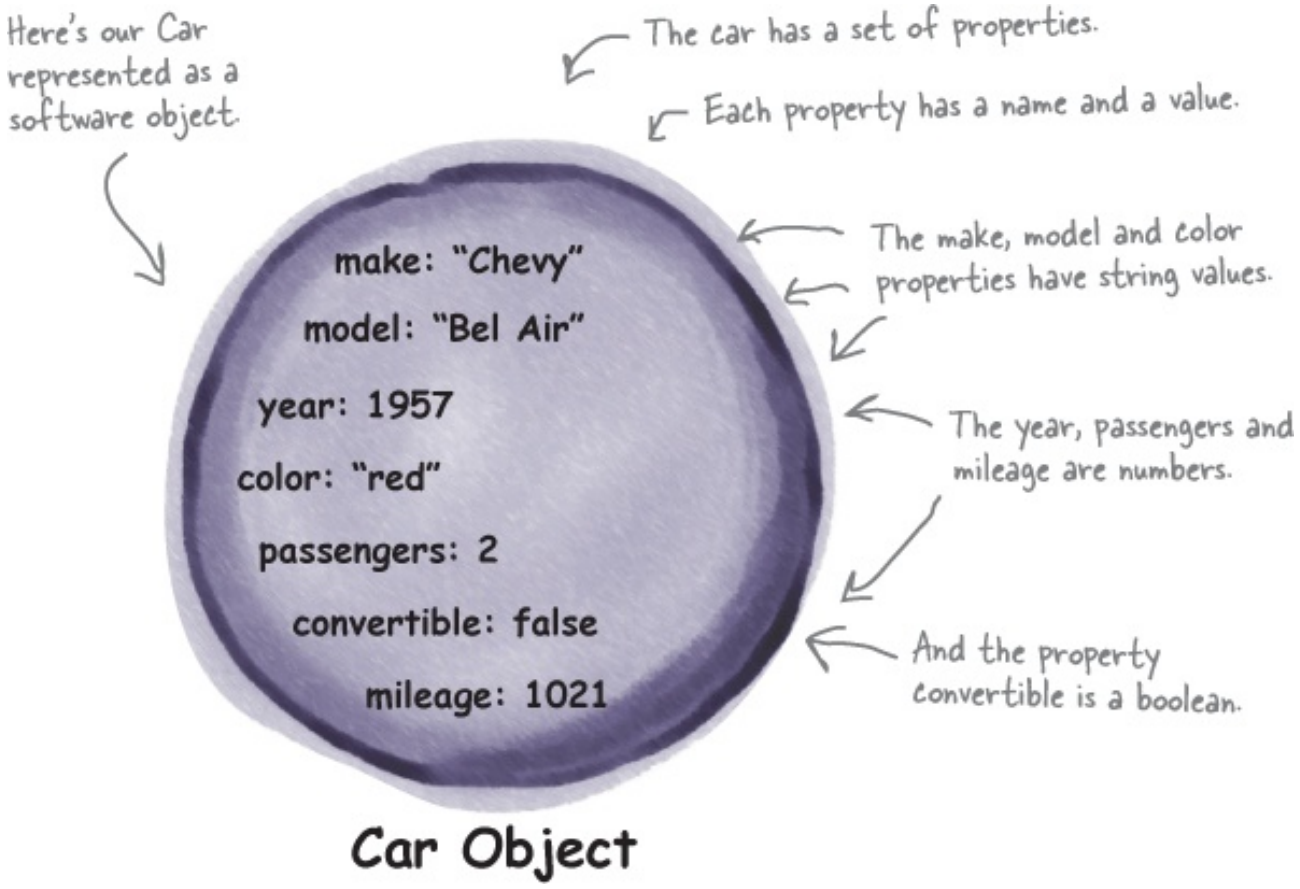



figure extraite de Javascript Head First

Création d'un objet simple en JS et affectation à une variable qui sera une **RÉFÉRENCE** à cet objet

```
const chevy01 = {
          make: 'Chevy',
          model: 'Bel Air',
          year: 1957,
          color: 'red',
          passengers: 2,
          convertible: false,
          mileage: 1021,
}

console.log(typeof chevy01)
```

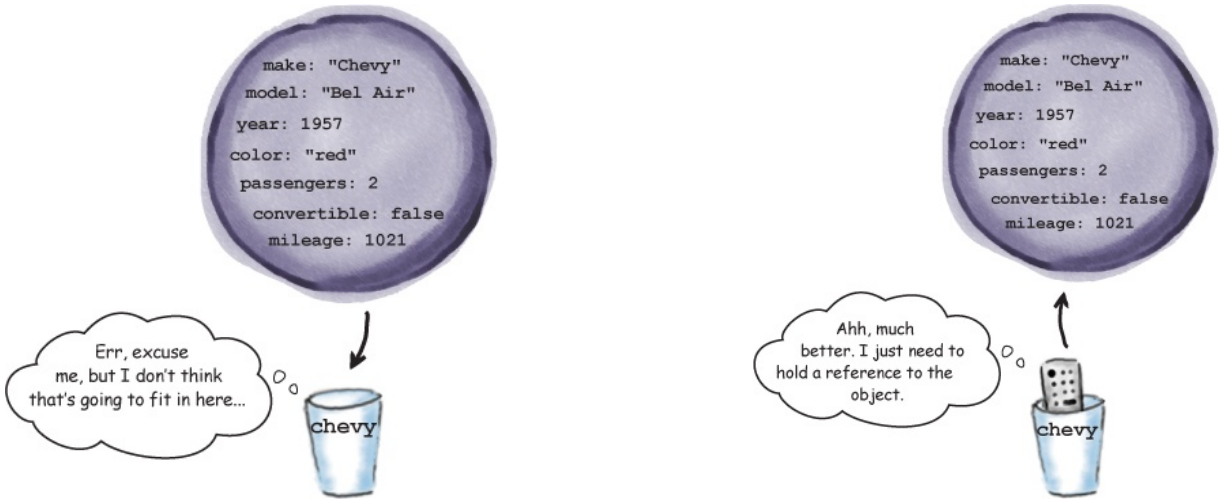
```
console.log(chevy01)

console.log(chevy01.mileage)
```

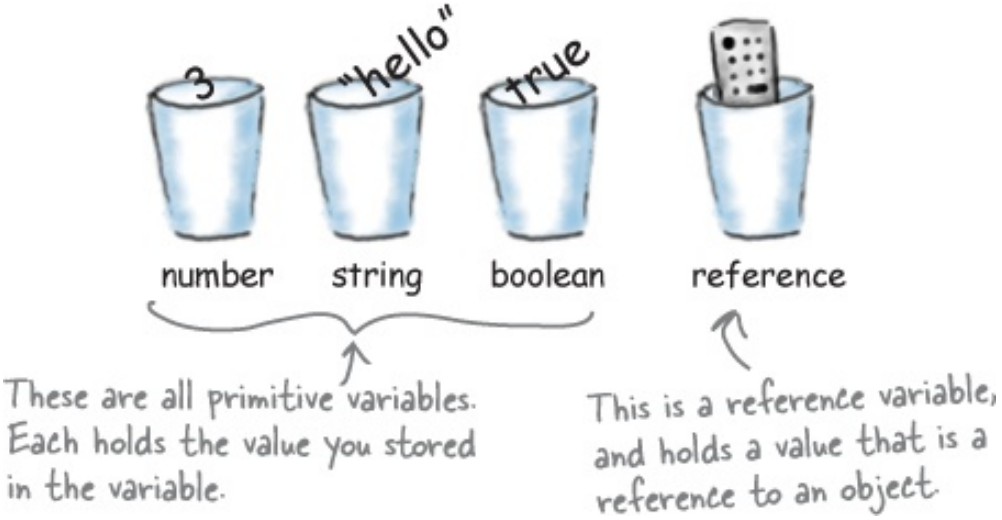
Les variables (ici chevy01) ne contiennent pas réellement d’objets.

Elles contiennent plutôt une référence à un objet.

La référence est comme un pointeur ou une adresse de l'objet réel.

Lorsque nous utilisons la notation par points, l'interprète JavaScript se charge d'utiliser la référence pour obtenir l'objet et accède ensuite à ses propriétés.



Ajoutons un comportement à nos objets

```
var chevy01 = {
          make: 'Chevy',
          model: 'Bel Air',
          year: 1957,
          color: 'red',
          passengers: 2,
          convertible: false,
          mileage: 1021,
          drive: function() {
               console.log("vroum la chevy01")
          },
          addFuel: function(amount) {
               this.fuel = this.fuel + amount ;
          }
}
```

maintenant

```
var chevy01 = {
          make: 'Chevy',
          model: 'Bel Air',
          year: 1957,
          color: 'red',
          passengers: 2,
          convertible: false,
          mileage: 1021,
          started: false ,
          fuel:0,

          start: function() {
               console.log("démarre")
               this.started = true
          },

          stop: function() {
               console.log("stop")
               this.started = false
          },

          drive: function() {
               if(this.started) {
                    console.log("go ahead chevy01 !!!")
```

```
            }
        },
}
```

Changer la fonction drive pour vérifier qu’il y a du fuel avant de démarrer.

```
// JAVASCRIPT.
function Employé (n,b) {
  this.nom = n;
  this.branche = b;
}
```

____

Person une fonction constructeur

```
function Person(nom, lage) {
     this.name = nom ;
     this.age = lage ;
}

let Jam = new Person("Jam",32);
```

Objet avec ses propriétés

```
let fred = new Person ("Fred", 21);

console.log(jam);
console.log(fred);
```

Maintenant rajoutons une méthode à la fonction

```
function Person(name, age) {
     this.name = name ;
     this.age = age ;
     this.present = () => { console.log("Hello my name is:" +
this.name)
}

john.present();
fred.present();

console.log(jam);
console.log(fred);
```

On voit qu'on a 2 fois la même méthode dans la console

Ces 2 méthodes ont été stockées 2 fois en mémoire alors qu'elle fait exactement la même chose c'est bien 2 méthodes différentes.

```
console.log( john.present === franck.present ) ==> false
```

## C'EST DU GACHIS !!!

LES PROTOTYPES permettent de stocker les méthodes

```
Person.prototype.present = function() {
     console.log("Hello my name is:" + this.name)
}
```

Avec les prototype ne pas utiliser de fonction fléchées à cause du **this** (on verra plus tard)

On ne vois plus la méthode present dans l'objet

Elle est ajoutée dans le prototype de la fonction Personne

Ouvrons __proto__

```
console.log(john.__proto__ === Person.prototype)
```

Continuons avec une fonction pour savoir si un objet a une propriété name

```
console.log(john.hasOwnProperty("name") ;
console.log(john.hasOwnProperty("cloud") ;
```

Where is property ? => in __proto__ de __proto__ de Object (objet mère)

```
let myObj = {} ; // creation d'un objet vide.
```

```
console.log(jam.__proto__.__proto__ === myobj.__proto__)
```

---

```
const myArray = [5,6,7]
```

```
console.log(myArray);
```

```
console.log(myArray.__proto__) // =>Array.prototype (Array est un
objet comme Person au-dessus avec ses fonctions)

console.log(myArray.__proto__.__proto__)   // on connait deja
(Objet.prototype)


const myString = "Hello" ;

console.log(myString);
console.log(myString.__proto__)
console.log(myString.__proto__.__proto__)
```

Essayer avec un nombre

Essayer avec fonction

```
function maFonction(){console.log('hello');}

console.log(maFonction)
console.log(maFonction.__proto__)
console.log(maFonction.__proto__.__proto__)
```

Même la fonction descend de l'objet

CHAINE DES PROTOTYPES:

Reprenons Person …

ajouter méthode present au prototype

Ajoutons dans l'objet
presentFrench = function (){}

```
jam.presentFrench()
jam.present();

console.log(john.hasOwnProperty("name") ;
```

Si je donne le même nom  dans une méthode de l'objet

john.present() va s'afficher en français.

cherche 1) dans l'objet 2) prototype de l'objet 3) prototype de Object

exemple:

```
console.log(john.hasOwnProperty("name"));
Person.prototype.hasOwnProperty = function(str) {
     return str +":"+  str
}
console.log(john.hasOwnProperty("name"));
```

ES6 classes

```
class Person {
     constructor(name,age) {
          this.name = name ;
          this.age = age
     }

     present() {
          console.log("Hello my name is:" + this.name)
     }
}
```

**Voir l'inspecteur ==> EXACTEMENT LA MÊME CHOSE**