

Cours de programmation

Les objets :

On l'a déjà dit, JavaScript n'est pas un langage orienté objet. Cependant, JavaScript possède de nombreuses caractéristiques et une syntaxe largement inspirée de la programmation objet. Ainsi, les données autres que simples nombres, chaînes de caractères ou booléens pourront être manipulés selon une syntaxe objet. A la base de la programmation objet se trouve la notion de **classe**. Une classe est en fait une famille d'objets possédant une structure et un comportement commun. Considérons un exemple très simple. Nous souhaitons représenter des individus. Nous allons pour cela créer une classe *individu* définissant les propriétés qui caractérisent les individus. Les caractéristiques que nous retiendrons seront le nom, le prénom et la date de naissance. En JavaScript, les caractéristiques sont appelées les **propriétés**. D'autre part, nous souhaitons disposer d'un outil qui fournisse l'âge d'un individu. En JavaScript, les outils sont appelés les **méthodes**. L'âge sera la différence entre la date actuelle et la date de naissance.

Une fois que l'on a créé une classe, il faut créer les éléments appartenant à cette classe. En JavaScript, on dit que l'on crée une **instance** de classe. On appelle **constructeur** l'outil qui permet de créer un élément particulier d'une classe.

définition d'une classe :

```
function individu(nom, prenom, date_naissance) {  
  this.nom = nom;  
  this.prenom = prenom;  
  this.date_naissance = date_naissance;  
}
```

Une fois le constructeur défini, nous pouvons créer la première instance de la classe *individu*. Dans notre cas, nous aurons :

création d'une instance :

```
niece = new individu('Clementine', 'Gomez', '3/12/95');
```

A présent, nous allons voir comment on accède aux différentes propriétés de cette instance. Les trois propriétés de l'objet *niece* pourront être référencées par *niece.nom*, *niece.prenom* et *niece.date_naissance*. Ainsi, si l'on veut modifier une des propriétés, on utilise la syntaxe :

modification d'une propriété :

```
niece.nom = 'Aubert';
```

Voyons comment ajouter une méthode à un constructeur. Tout d'abord, il faut créer une fonction qui va définir le comportement de cette méthode. Ensuite, on intègre cette méthode au constructeur, comme une propriété dont le nom sera le nom de la méthode et la valeur sera le nom de la fonction préalablement défini.

définition de l'outil :

```
function age() {  
  aujourd'hui = new Date();  
  dateJS = this.date_naissance.split('/');  
  Naissance = new Date(dateJS[2], dateJS[1]-1, dateJS[0]);  
  NombreAnnee =
```

```
Math.floor((aujourd'hui.getTime()-Naissance.getTime()/(24*60*60*1000*365.25));  
    return NombreAnnee;  
}
```

Nouvelle allure du constructeur :

```
function individu(nom, prenom, date_naissance) {  
    this.nom = nom;  
    this.prenom = prenom;  
    this.date_naissance = date_naissance;  
    this.age = age;  
}
```

Maintenant, nous avons terminé. Pour appliquer une méthode à une instance d'objet, nous pourrions utiliser le code suivant.

définition de l'outil :

```
niece = new individu('Clementine', 'Gomez', '3/12/95');  
document.write(niece.age()+ 'an'+((niece.age()>1)?'s.':'/'));
```

La propriété prototype :

Toutes les classes disponibles en JavaScript, qu'elles soient prédéfinies ou définies par l'utilisateur, possèdent une propriété appelée *prototype* qui permet d'enrichir la structure d'une classe en lui ajoutant une nouvelle propriété ou une nouvelle méthode.

Supposons que l'on ait créé une classe Rectangle(). Elle possède deux propriétés : longueur et largeur. On va lui ajouter une propriété couleur. Cela sera possible grâce à la propriété *prototype*.

Exemple :

```
<SCRIPT>  
function Rectangle (a, b)  
{  
    if (a > b) {  
        this.longueur = a;  
        this.largeur = b;  
    }  
    else {  
        this.longueur = b;  
        this.largeur = a;  
    }  
}  
  
MonRectangle = new Rectangle(3, 4);  
Rectangle.prototype.couleur = 'noir';  
document.write('La couleur de mon rectangle est '+Rectangle.couleur+'<BR>');  
</SCRIPT>
```

La fonction **typeof** :

La fonction **typeof** permet comme son nom l'indique de connaître le type d'une entité JavaScript.

La syntaxe est simple : **typeof** (MaVariable);.

Les tableaux :

Nous avons déjà évoqué les tableaux. Mais nous allons voir les tableaux comme objet. En général, l'index d'un tableau est un entier, mais il peut aussi être une chaîne de caractères. Dans ce cas, le tableau est dit associatif. Voici deux exemples, l'un présente un objet classique, l'autre un tableau dont la déclaration est semblable à celle d'une classe (ou objet).

Exemple :

```
<SCRIPT>
var Tableau = new Array();
Tableau[0] = 1;
Tableau[2] = 'chien';
</SCRIPT>
```

Exemple :

```
<SCRIPT>
var Tableau = new Array();
Tableau['nom'] = 'Nicolas';
Tableau['age'] = 4;
</SCRIPT>
```

L'instruction **new** :

L'instruction **new** permet comme vous avez déjà dû le constater de créer un nouvel objet. Elle doit pour cela être associée à un constructeur qui permet d'indiquer à la fois le type de l'objet et ses valeurs initiales. En terminologie objet, on dira que **new** permet de créer une nouvelle instance d'une classe.

L'instruction **this** :

this permet de référencer l'objet courant. Cette instruction est principalement utilisée lors de la définition des constructeurs et des méthodes d'une classe. Dans le cas de la définition d'un constructeur, elle permet de faire référence à la futur instance. Dans le cas de la définition d'une méthode, elle permet de faire référence à l'instance à laquelle s'appliquera cette méthode.

Mais **this** est aussi souvent utilisé lorsqu'on insère du code JavaScript au sein d'une balise HTML. Il permet alors de faire référence à l'objet défini par cette balise.

