HTML5 avancé

Javascript – notions avancés

Closure
Principe

- Pour rappel, une variable a une certaine portée de visibilité, appelée *scope*

```
function myFunc(){
    var x=0;
}
console.log(x); //undefined
```

- Ce concept permet de profiter du mécanisme des *closures*
- Le principe est de limiter l'accès à un certain contenu (variables, fonctions)

- Pour cela, une fonction peut en encapsuler une autre

Closure Exemple

- Les closures permettent l'encapsulation

```
function monObjet(value) {
    var x = value;
    return {
        getX : function() {
            return x;
        }
    }
}

var obj = monObjet(3);
console.log(obj.x);          // undefined
console.log(obj.getX());     // 3
```

visibilité limitée à la fonction monObjet

la closure permet d'y accéder

Héritage / Prototype

- Javascript est un langage par prototype, permettant une approche orientée objet
- Les prototypes sont en quelque sorte les cartes d'identités des objets
- Il n'existe pas de 'classes' mais les objets héritent du Prototype de l'objet étendu
- Ce prototype est partagé

```
function Personne() {};
var olivier = new Personne();
olivier.sayHey();                                    // error
Personne.prototype.sayHey = function() {
    console.log('hey !') ;
}
olivier.sayHey() ;                                   // hey !
```

Héritage / Prototype

- Redéfinir une fonction du même nom permet de surcharger celle du parent

```
var Benoit = new Personne();
Benoit.sayHey();                          // hey !
Benoit.sayHey = function(){
    console.log('olá !') ;
}
Benoit.sayHey();                          // olá
Olivier.sayHey();                         // hey !
```

- Mais il reste possible de faire appel à la fonction héritée

```
Benoit.constructor.prototype.sayHey();    // hey !
```

Héritage / Prototype

- En revanche, il est possible d'effectuer des modifications globales

```
Benoit.constructor.prototype.sayHey = function(){
    console.log('Pourquoi ?') ;
}

Olivier.sayHey();
=> Pourquoi ?
```

- L'héritage de prototype permet donc d'enrichir des types

```
Number.prototype.sqrt = function(){ (...) };
var x=4 ;
x.sqrt();
=> 2
```

Object Literals

- Cette notation permet de déclarer des objets d'une manière élégante
- La structure d'un fichier JSON est calquée sur celle-ci

```
var voiture = {
    'marque' : 'mercedes',
    'annee' : '1989'
}

voiture.marque;   //mercedes
voiture.annee; //1989
```