

Exemple d'application des objets en Javascript

Pourquoi les projets graphe et pendu vont naturellement amener à utiliser des objets ?

- Les données échangées entre client et serveur sont généralement structurées :
 - => Les échanges se font ici en utilisant des notations objets (*JSON : Javascript Object Notation*)
 - => A la réception des données, il faudra les analyser en utilisant les propriétés des objets
- Un nœud est défini par ses coordonnées x et y et un poids :
 - => Il faudra regrouper, pour chaque nœud, les coordonnées et le poids dans une structure
 - => En créant des objets on pourra regrouper toutes les valeurs (*attributs*) et définir la façon commune de les modifier (*méthodes*). Ceci permettra de créer et dessiner facilement les nœuds.
- Pour gérer le graphe à l'écran, il faut pouvoir placer, déplacer et supprimer des nœuds
 - => Pour gérer globalement les nœuds, il faut les ajouter dans une structure unique (*le graphe*)
 - => L'utilisation d'un tableau de nœuds (*collection en Python*) pourrait être envisagé pour parcourir tous les nœuds mais la suppression des nœuds du tableau se compliquerait sensiblement
 - => L'utilisation d'un objet pour stocker les nœuds (*dictionnaire en Python*) est préférable car les nœuds seront facilement insérés (*ajout d'un attribut*) et supprimés (*suppression d'un attribut*)

En Javascript, il y a plusieurs façons de créer des objets :

- On peut utiliser le même format (*Json*) que celui des données échangées comme par exemple :

```
var pointJSON = {                // A noter que c'est déjà un objet fonctionnel
  abs : 0,                       // => attribut avec valeur par défaut utilisable par this.abs
  ord : 0,                       // => attribut avec valeur par défaut utilisable par this.ord
  deplace : function(dx, dy) {  // Méthode de l'objet
    this.abs += dx;
    this.ord += dy;
  },
  parle : parle                  // Autre méthode, décrite ici en dehors de l'objet
}
```

```
function parle() {
  console.log("Je suis le point de coordonnées " + this.abs + " " + this.ord);
}
```

// Et maintenant on peut l'utiliser pour créer d'autres objets par duplication :

```
var p1 = Object.create(pointJSON); // p est obtenu par duplication de l'objet pointJSON
p1.abs = 4;                        // On positionne ici les attributs après duplication
p1.ord = 5;                        // Syntaxe équivalente : p['ord'] = 5; pour utiliser des variables
p1.parle();                        // --> "Je suis le point de coordonnées 4 5"
p1.deplace(2, -1);
p1.parle();                        // --> "Je suis le point de coordonnées 6 4"
pointJSON.parle();                 // --> "Je suis le point de coordonnées 0 0"

p1["couleur"] = "black";           // --> "Ajout d'une propriété à l'objet p1"
console.log(p1.couleur);           // --> "black"
```

- La méthode de création des objets qui nous intéresse davantage se fait, sous Javascript, via l'utilisation de fonctions anonymes.

On fait précéder du mot clé **this** les propriétés et les méthodes pour les rendre publiques. Pour faire la différence avec le mode précédent et comme cela ressemble à une classe, on écrira :

```
var ClassPoint = function (x, y) {      // Écriture à préférer à function ClassPoint(x,y) {
  this.abs = x || 0;                    // => attribut initialisé à la création de l'objet ou 0
  this.ord = y || 0;                    // => attribut initialisé à la création de l'objet ou 0
  this.deplace : function(dx, dy) {    // Méthode de l'objet
    this.abs += dx;
    this.ord += dy;
  }
}
```

```
ClassPoint.prototype.parle = function() { // Prototype de tous les objets de type ClassPoint
  console.log("Je suis le point de coordonnées " + this.abs + " " + this.ord);
}
```

// Et maintenant on peut l'utiliser en créant des objets à partir de cette « classe »:

```
var p2 = new ClassPoint(4, 5); // Instanciation (création/initialisation) par appel du constructeur
p2.parle();                    // --> "Je suis le point de coordonnées 4 5"
p2.deplace(2, -1);
p2.parle();                    // --> "Je suis le point de coordonnées 6 4"
```

```
var p3 = new ClassPoint();     // Instanciation (création/initialisation) par appel du constructeur
p3.parle();                    // --> "Je suis le point de coordonnées 0 0"
```

```
var p4 = new ClassPoint(3);    // Instanciation (création/initialisation) par appel du constructeur
p4.parle();                    // --> "Je suis le point de coordonnées 3 0"
```

```
p3 = p4;
```

Ce qui donne deux références sur le même objet

La mémoire est allouée lors de la création des objets puis libérée « automatiquement » lorsque ceux-ci ne sont plus utilisés. Un objet peut être « ramassé » par le ramasse-miettes quand il n'y a plus de références pointant vers lui. La destruction d'une référence à un objet peut être obtenue par la syntaxe :

```
monObjet = null;
```

A ne pas confondre avec la destruction d'une propriété d'un objet qui se fait par la syntaxe :

```
delete objet.propriete;
```

exemple : delete p1.couleur; ou delete p1["couleur"];