Labo 3 –Javascript orienté objet

Exercice 1 : pour se familiariser avec l'orienté objet en JS

Le but de cet exercice est de se familiariser avec la manière dont Javascript gère l'orienté objet. Dans la plupart des étapes, on vous demande d'introduire des commandes directement dans l'invite de la console… ceci dit, pour éviter les conséquences fâcheuses des erreurs, il vaut peut-être mieux tout d'abord préparer ces commandes dans un fichier texte puis les copier/coller vers la console, histoire d'en conserver une copie.

Attention. Cet exercice vous guide progressivement à travers la complexité de l'orienté objet en Javascript. Dans les premières étapes, on vous demande d'utiliser des procédés qui ne sont pas forcément les meilleurs méthodes pour faire de l'orienté objet. Le but est didactique : cela permet de mettre en évidence les lacunes de ces premières méthodes et de démontrer pourquoi il vaut mieux en utiliser d'autres.

Étape 1

Créez tout d'abord le document suivant, qui servira de base à cet exercice. Examinez son contenu et comparez-le avec ce que le navigateur affiche quand vous l'ouvrez.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
    <style>
      .cadrestd {
        background-color: #EFFFEF;
        border: 1px solid blue;
        padding: 4px;
      }
    </style>
  </head>
  <body>
    <div id="cadre" class="cadrestd">
      <p>Le cadre commence ici.</p>
    </div>
  </body>
</html>
```

Étape 2

Une fois le document chargé dans Firefox, ouvrez la console (Ctrl-Shift-K). Dans cet exercice, vous allez ajouter dynamiquement (après le chargement de la page web) des éléments-paragraphes à l'intérieur du div identifié par « cadre ».

Pour ajouter un paragraphe dans le cadre, vous pouvez utiliser la commande suivante (que vous pouvez entrer dans la console). Comme d'habitude, faites l'effort de recopier le texte à la main plutôt que d'utiliser un simple copier/coller afin de vous familiariser avec Javascript.

```
document.getElementById("cadre").innerHTML +=
  "<p style='color:red; font-weight:bold'>Un autre paragraphe</p>";
```

Pour rappel, la fonction `document.getElementById` permet de retrouver un élément du document HTML et, via sa propriété « innerHTML », on peut accéder à son contenu (au format HTML). Notez au passage l'utilisation de guillemets et d'apostrophes.

Étape 3

Parlons objets… Chacun des paragraphes ajoutés dans le cadre sera représenté par un objet. Cet objet indiquera (a) le contenu du paragraphe à ajouter, (b) la couleur de texte désirée et (c) si le texte doit apparaître en gras ou pas.

On peut créer un premier objet avec les instructions suivantes.

```
var p1 = {};
p1.texte = "Bonjour";
p1.couleur = "purple";
p1.gras = true;
```

Après avoir entré ces lignes, si vous tapez `p1` dans la console, celle-ci vous répondra qu'il s'agit d'un « Object », c'est-à-dire d'un objet héritant directement de `Object`.

Pour en savoir plus, cliquez sur le mot *Object* et un descriptif de l'objet `p1` apparaît à droite. Vous pouvez entre autres y voir les trois propriétés de `p1` (`texte`, `couleur` et `gras`) et un lien vers son prototype `__proto__`, qui n'est autre que « l'objet-père » `Object`.

Étape 4

Pour automatiser l'ajout du texte au cadre, on peut définir une méthode « `ajoute` » dans l'objet `p1`.

Cette méthode « `ajoute` » devra ajouter à l'intérieur du div identifié par « cadre » un paragraphe dont le contenu est `this.texte`, dont la couleur est `this.couleur` et qui, en fonction du booléen `this.gras`, est écrit en gras ou pas.

Vous pouvez vous aider de la commande donnée à l'étape 3 pour construire cette méthode. Utilisez judicieusement les guillemets et les apostrophes (qui sont interchangeables tant en HTML qu'en Javascript). Écrivez la méthode puis exécutez-la.

Attention : comme cette méthode sera également employée avec d'autres paragraphes, il est important qu'elle utilise bien `this.texte`, `this.couleur` et `this.gras` plutôt que le texte, la couleur et le style du paragraphe que vous venez d'afficher !

Étape 5

Créez un objet `p2` qui hérite de `p1` grâce à l'instruction suivante.

```
var p2 = Object.create(p1);
```

Utilisez la console pour voir ce qu'est `p2` (entrez simplement `p2` puis cliquez sur les mots « Object » pour voir son contenu). Utilisez la console pour voir si `p2.texte`, `p2.couleur` et `p2.gras` ont une valeur et pour vérifier si `p2.ajoute()` fonctionne (en entrant ces diverses expressions directement dans la console).

Observez que p2 n'a, en fait, aucun champ propre : aucun attribut, aucune méthode, juste un prototype p2.__proto__. Par contre, lorsqu'il s'agit d'évaluer p2.texte ou une autre propriété de p2, on « remonte » jusqu'à p1. Testez que c'est bien le cas en évaluant p2.texte et p2.couleur via la console.

D'après vous, qu'indiquera la console si vous lui demandez d'évaluer l'expression suivante ? Vérifiez votre réponse.

```
p2.__proto__ == p1
```

Note. Si, à un moment donné, le cadre affiché devient trop rempli, vous pouvez le « vider » en entrant la commande suivante dans la console.

```
document.getElementById("cadre").innerHTML = "";
```

Étape 6

En supposant que p1 corresponde au texte « Bonjour » en pourpre gras et que vous venez de réaliser l'étape 5, tentez de déterminer ce que chacun des groupes d'instructions suivants va faire apparaître (on suppose qu'ils sont exécutés à la suite l'un de l'autre). Vérifiez vos réponses en utilisant la console.

```
p1.couleur = "orange";
p1.ajoute();
p2.ajoute();

p2.texte = "Au revoir";
p1.ajoute();
p2.ajoute();

p1.texte = "Salut";
p1.ajoute();
p2.ajoute();

p2.__proto__.couleur = "red";
p1.ajoute();
p2.ajoute();
```

Étape 7

Définissons maintenant une fonction-constructrice pour les paragraphes à ajouter au cadre ; celle-ci créera automatiquement un objet possédant automatiquement les 3 propriétés données en arguments (texte, couleur et gras) ainsi qu'une méthode ajoute(). Pour suivre les conventions habituelles, on nommera cette fonction « Para » (avec un P majuscule).

Ainsi, l'appel

```
var p = new Para("Salut !", "blue", true);
```

devrait engendrer un objet p avec « Salut ! » pour texte, « blue » comme couleur, pour lequel p.gras est vrai et qui possède une méthode p.ajoute().

Utilisez la fonction-constructrice que vous venez d'écrire pour créer un « paragraphe » appelé `devise` et contenant le texte « Les Lannister paient toujours leurs dettes. » en couleur « darkgreen » et en gras. Puis ajoutez-le au cadre en utilisant la méthode appropriée.

Pour vérifier votre code, utilisez la console pour voir ce que l'objet « devise » a dans le ventre. Observez la présence des 5 propriétés `texte`, `couleur`, `gras`, `ajoute` et `__proto__`. Comme d'habitude, entrez simplement « devise » puis cliquez sur les mots « Object ».

Étape 8

Dans la console, redéfinissez la fonction-constructrice `Para` pour que, désormais, la méthode « ajoute » centre le texte horizontalement. Au niveau du code HTML, cela revient à ajouter la propriété « text-align : center » dans le style du paragraphe.

Créez ensuite un second « Para » appelé « devise2 » et contenant le texte « Winter is coming. » en couleur « cyan » et sans gras.

Exécutez les instructions suivantes.

```
devise2.ajoute();
devise.ajoute();
```

L'une devrait apparaître à gauche et l'autre, centrée… est-ce normal ?

Qu'aurait-on dû faire pour que la modification de la méthode « ajoute » ait un effet rétroactif ?

Étape 9

Comme les deux paragraphes `devise` et `devise2` ont été créés par la même fonction constructrice, ils partagent un protoytpe commun. C'est la fonction constructrice `Para` qui leur a associé ce prototype (sans que vous n'ayez à le demander explicitement). Le prototype en question est un objet associé à la fonction `Para`, à savoir `Para.prototype`.

Pour le vérifier, évaluez les égalités suivantes dans la console.

```
devise.__proto__ == devise2.__proto__

devise.__proto__ == Para.prototype
```

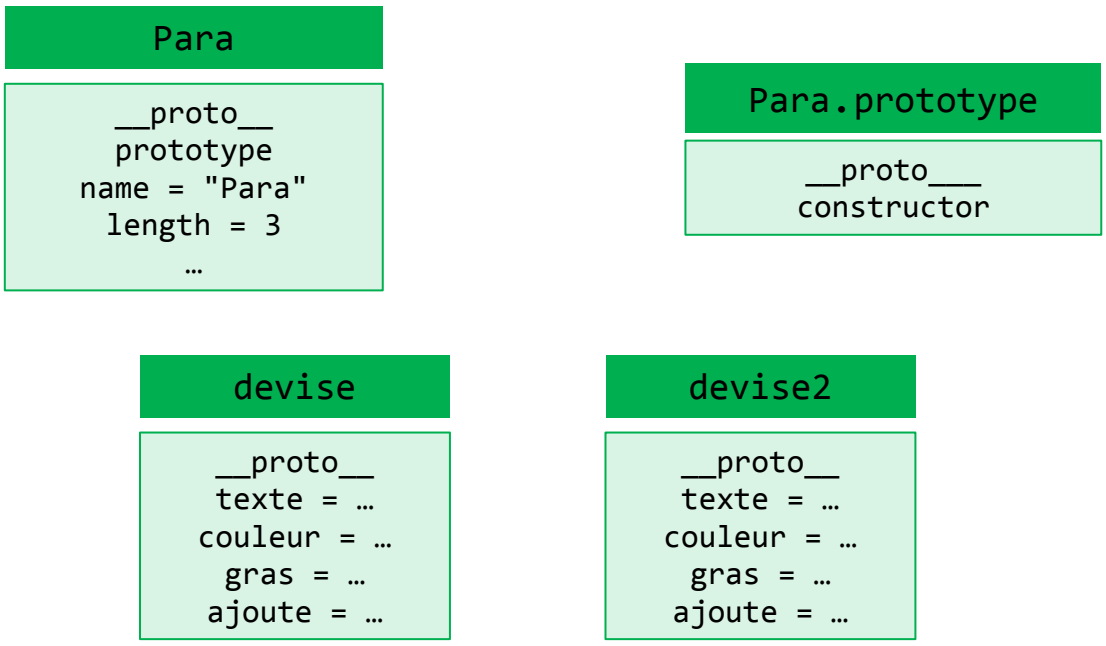
Dans l'objet qui représente la fonction `Para`, on trouve donc un champ appelé « prototype » et qui contient une référence vers cet objet `Para.prototype` qui sert de prototype commun à tous les objets créés par `Para`. De même, dans cet objet `Para.prototype`, on trouve un champ appelé « constructor » qui contient une référence vers l'objet décrivant `Para`.

Testez que c'est bien le cas en évaluant les égalités suivantes.

```
Para == Para.prototype.constructor

devise.__proto__.constructor == Para
```

Pour vous assurer que vous comprenez bien la situation, complétez le diagramme suivant en ajoutant des flèches décrivant le contenu des attributs qui contiennent des références. Si nécessaire, aidez-vous de la console Javascript.



Étape 10

Définissez maintenant une nouvelle fonction constructrice appelée `ParaPro`. Celle-ci accomplira un travail similaire à celui de `Para` mais elle ne placera que trois propriétés sur les objets créés : texte, couleur et gras.

Plutôt que de définir la méthode ajoute dans chacun des objets, créez une définition unique et placez-la dans l'objet qui servira de prototype à tous les « paragraphes pro ».

Pour vérifier votre travail, créez deux « paragraphes pro » appelés `devisepro` et `devisepro2` et contenant les devises données ci-dessus. Observez le contenu des objets `devisepro` et `devisepro2` via la console et exécutez les lignes suivantes.

```
devisepro.ajoute();
devisepro2.ajoute();
```

Dans la console, modifiez la définition de la fonction ajoute pour que celle-ci fasse apparaître le paragraphe en centré.

Ensuite, exécutez à nouveau les deux instructions suivantes.

```
devisepro.ajoute();
devisepro2.ajoute();
```

Cette fois-ci, la modification a été prise en compte rétroactivement. Assurez-vous de bien comprendre pourquoi.

Qu'est-ce qui a changé par rapport au diagramme en haut de cette page ?

Exercice 2 : objets en tant que structures

Les objets Javascript sont avant tout des tableaux associatifs. On peut les utiliser comme l'équivalent des structures du langage C.

Étape 1

On va considérer des structures qui décrivent les pièces d'une maison (pièces qu'on supposera rectangulaires), indiquant leur nom (une chaîne de caractères), leur longueur, leur largeur et leur hauteur.

Définissez tout d'abord trois objets, un pour chacune des pièces suivantes. Comme il y a 3 manières de définir un objet en Javascript, utilisez chaque fois une manière différente (syntaxe orienté objet, syntaxe littérale et syntaxe tableau associatif).

Variable	Nom	Largeur	Longueur	Hauteur
salle1	cuisine	3	4	3
salle2	chambre	4	4	3
salle3	salon	6	8	4

Étape 2

Définissez une fonction appelée `surfaceSol` qui reçoit en argument une salle et renvoie la superficie de son sol (largeur × longueur). Définissez également une fonction `surfaceMurs` qui reçoit en argument une salle et renvoie la superficie de ses murs (la surface à tapisser).

Attention : il s'agira bien de fonctions recevant une salle (c'est-à-dire un objet, ou tableau associatif) comme argument, pas de méthodes !

Étape 3

Chaque salle sera décorée de manière différente. Dans le cadre de cet exercice, on considère que la décoration se limite à un plancher et de la tapisserie. Le tarif de décoration est représenté par un objet indiquant le coût du plancher (au mètre carré) et le coût de la tapisserie (au mètre carré).

Ci-dessus, on a traité les objets-salles comme de simples tableaux associatifs et on les a créés « manuellement » en employant diverses syntaxes. Pour faciliter la création des objets-tarifs (qui, eux aussi sont de simples tableaux associatifs sans méthodes), on décide d'utiliser une fonction constructrice.

Définissez une fonction constructrice `Tarif` acceptant deux arguments (les deux coûts) et plaçant tout simplement ces valeurs dans les attributs `coutPlancher` et `coutTapisserie` de l'objet créé. Utilisez-la ensuite pour créer les deux tarifs suivants.

Variable	Description	Coût plancher	Coût tapisserie
tarif1	moquette + papier peint velours	101,00€/m^2	37,50€/m^2
tarif2	carrelage + papier peint	53,30€/m^2	7,80€/m^2

Étape 4

Créez une fonction `coutDeco(salle,tarif)` indiquant le prix de la décoration d'une salle donnée selon le tarif donné. Ce prix sera calculé en additionnant le prix du plancher au prix de la tapisserie.

Étape 5

On se dit que, plutôt que d'employer des fonctions à gauche et à droite, ce serait quand même plus simple d'utiliser des méthodes et, entre autres, de pouvoir calculer le prix total de la décoration d'une salle en utilisant une syntaxe telle que `salle.cout(tarif)`.

Pour ce faire, et pour éviter de devoir placer la définition de cette méthode « cout » sur chaque objet-salle, il va falloir

1. créer un objet vide `protoSalle` qui servira de prototype à toutes les salles ;

2. ajouter une méthode `cout(tarif)` à cet objet vide *(réutilisez la fonction* `coutDeco` *définie plus tôt… en utilisant la syntaxe adéquate !)* ;

3. puis faire en sorte que `protoSalle` devienne le prototype des 3 objets-salles définis plus haut. *(Pour une des salles, utilisez directement* `__proto__` *; pour une autre, utilisez* `Object.setPrototypeOf(…)` *; pour la troisième, utilisez une méthode au choix)*

Vérifiez que tout fonctionne en évaluant quelques expressions du style `salle.cout(tarif)`.

Étape 6

Observez que cet exercice **ne** montre **pas** la voie à suivre !

Il vaut mieux bien prendre le temps de réfléchir à comment structurer ses objets avant de se mettre à programmer… au lieu de se lancer comme un fou furieux sur le clavier, commencer à écrire du code, et devoir modifier le tout (ou mettre en place des solutions branlantes) quand on se rend compte qu'on a pas choisi la meilleure option dès le départ !

Exercice 3 : les objets prédéfinis

Javascript associe automatiquement un prototype à chaque fonction constructrice, qu'il s'agisse d'un constructeur défini par l'utilisateur ou d'un constructeur prédéfinie comme par exemple `Number` (pour les nombres), `String` (pour les chaînes de caractères) ou encore `Function` (pour les fonctions).

Si on modifie l'objet-prototype associé à un constructeur en lui ajoutant une méthode, celle-ci devient accessible à partir de tous les objets créés par ce constructeur. Cela permet entre autres d'ajouter de nouvelles méthodes aux objets standards.

Étape 1

Dans un premier temps, définissez une fonction `tableMultiplication(n)` qui affiche dans la console la table de multiplication du nombre n. Cette table de multiplication sera produite au format suivant (pour n = 7).

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
```

```
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

Ciblez l'objet-prototype associé à la fonction constructrice `Number`, à savoir l'objet `Number.prototype` et ajoutez-lui une méthode `afficheTable()` dont l'effet sera d'afficher (dans la console) la table de multiplication d'un objet de « type » Number. Comme d'habitude, vous utiliserez « this » pour faire référence à cet objet.

Pour vérifier votre code, entrez les lignes suivantes dans la console.

```
var x = 5;
x.afficheTable();
```

Étape 2

Évaluez l'objet Number.prototype dans la console. Cliquez sur le mot « Object » pour avoir le détail de son contenu.

Observez que la méthode que vous avez ajoutée (`afficheTable`) s'y trouve.

Observez également la valeur de la propriété « constructor ». Cliquez sur le triangle à gauche du mot « constructor » pour observer le contenu de l'objet référencé par cette propriété. Il s'agit bien entendu de la fonction `Number` !

Étape 3

Ciblez maintenant les chaînes de caractères. Votre but est d'ajouter une méthode `afficheNFois(n)` qui devra être accessible à partir de n'importe quelle chaîne de caractères `s` et qui permettra d'afficher dans la console une chaîne constituée de `n` fois la chaîne `s`.

Par exemple, l'exécution de

```
var chante = "la";
chante.afficheNFois(5);
```

devrait afficher « lalalalala » dans la console.

Étape 4

Pourquoi s'arrêter aux chaînes de caractères ? Ciblez maintenant les fonctions. Votre but est d'ajouter une méthode `applique2Fois()` qui devra être accessible à partir de n'importe quelle fonction `f` et qui permettra d'exécuter `f()` deux fois (On supposera que cette méthode ne sera appelée que sur des procédures ne demandant aucun argument).

Ainsi, l'exécution de

```
function ditBonjour () { alert("Bonjour !"); }
ditBonjour.applique2Fois();
```

devrait produire deux popups indiquant « Bonjour ! ».

Étape 5

Votre but est d'ajouter une méthode `afficheJusquaN(n)` qui devra être accessible à partir de n'importe quelle fonction `f` et qui permettra d'afficher les valeurs f(1), f(2), f(3), …, f(n) (On supposera que la méthode `afficheJusquaN` ne sera utilisée que sur des fonctions à un argument numérique).

Par exemple, l'exécution de

```
function carre (x) { return x * x; }
function fib (n) {
  return (n == 1 || n == 2) ? 1 : fib(n-2) + fib(n-1)
}
carre.afficheJusquaN(7);
fib.afficheJusquaN(10);
```

devra afficher le résultat suivant.

```
1 4 9 16 25 36 49
1 1 2 3 5 8 13 21 34 55
```