

# Module 12

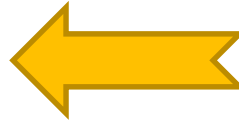
## *Compléments*

Technologies web

HENALLUX — IG2 — 2016-2017

# Compléments

➤ Mode strict et mode normal



➤ JSON

➤ L'orienté objet en ES6

➤ jQuery (aperçu)

➤ Pour aller plus loin...

# Mode normal

Par défaut, Javascript travaille en "**mode normal**" (ou **sloppy mode**).

- De nombreuses "erreurs" sont passées sous silence.

```
let x = 3;  
x.bonjour = "salut";  
x.bonjour           // donne undefined
```

**Mode strict**  
TypeError

- De nombreuses facilités sont offertes pour tirer le plus possible du code.

```
function varOublié () {  
  varLocale = 3;    // déclaré en global  
}
```

**Mode strict**  
ReferenceError: assignment to undeclared variable

- Des conventions "historiques"

```
( function action () {  
  this.machin = 3;    // this = l'objet global  
} )();
```

**Mode strict**  
TypeError: this is undefined

- Problèmes potentiels :
  - Bugs cachés et/ou difficiles à cibler
  - Rétrocompatibilité vs. évolution du langage

# Mode strict

Comment passer en **strict mode** ?

- S'arranger pour que la 1<sup>re</sup> instruction du script (ou du bloc fonctionnel ou du fichier) soit 'use strict'; ou "use strict";

```
<script>
```

```
// Ce commentaire ne compte pas.
```

```
"use strict";
```


```
...
```

```
</script>
```

- Différentes portées possibles :
  - 1<sup>re</sup> instruction d'un script : tout le script
  - 1<sup>re</sup> instruction d'un fichier : tout le fichier
  - 1<sup>re</sup> instruction du corps d'une fonction : la fonction
- Plus d'information sur les différences : voir le site MDN

# Compléments

➤ Mode strict et mode normal


➤ JSON 

➤ L'orienté objet en ES6

➤ jQuery (aperçu)

➤ Pour aller plus loin...

# JSON

- 
- La sérialisation d'objets
  - La solution XML (aperçu)
  - La solution JSON
  - Comment définir précisément une syntaxe ?

Ensuite : *L'orienté-objet en ES6*

# Sérialisation

## Qu'est-ce que la sérialisation ?

- Quand un programme se termine, ses données disparaissent.
- Pour pouvoir les réutiliser plus tard, il faut les stocker quelque part (les rendre « **permanentes** »).
- Quelques cas simples...
  - Tableau de nombres → écrire les nombres
  - Texte → écrire les caractères
  - Mais quid des objets ?
- **Sérialiser** un objet, c'est le transformer en un format mieux adapté pour le stockage à long terme.

# Sérialisation

## Quelques possibilités pour sérialiser des objets...

- **format binaire**

On recopie "bit à bit" l'endroit de la mémoire où l'objet est stocké.

*Avantage* : facile

*Désavantage* : peu/pas portable, peu/pas lisible pour un humain

- **formats standards**

- **XML** (Extensible Markup Language)

- = langage de balisage extensible
    - à la base du HTML
    - peut être adapté à toute une série de domaines

- **JSON** (JavaScript Object Notation)



# Sérialisation en XML

## Exemple de document XML

```
<?xml version="1.0" ?>
```

```
<endangered_species>
```

```
  <animal>
```

```
    <name language="English">Tiger</name>
```

```
    <name language="Latin">panthera tigris</name>
```

```
    <threat>poachers</threat>
```

```
    <weight>500 pounds</weight>
```

```
    <source sectionid="120" newspaperid="21"></source>
```

```
    <picture filename="tiger.jpg" x="200" y="197"/>
```

```
  </animal>
```

```
  <animal>
```

```
    ...
```

```
  </animal>
```

```
</endangered_species>
```

- Balises (selon le domaine)
- Attributs
- Valeurs placées entre les balises
- Structure imbriquée

# Sérialisation en XML

## Exemple de document XML (question à un Web Service)

```
<?xml version="1.0"?>

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

- Structure définie dans des fichiers de définition
- Format standard utilisé dans de nombreux outils du web

# Sérialisation en JSON

## Autre standard qui gagne du terrain : JSON

- JavaScript Object Notation
- basé sur la syntaxe des littéraux de type "objet" en Javascript
- utilisable (et utilisé) pas seulement en Javascript

## Exemple en JSON

```
{  
  "prénom" : "Homer",  
  "nom" : "Simpson",  
  "adresse" : {  
    "numéro" : 742,  
    "rue" : "Evergreen Terrace"  
  },  
  "enfants" : [ "bart", "lisa", "maggie" ]  
};
```

Objet : entre accolades,  
des paires "clef" : info  
séparés par des virgules

Tableaux : entre  
crochets, éléments  
séparés par des  
virgules

# Sérialisation en JSON

## Structure d'un élément au format JSON

- Objet = suite de paires clefs/valeurs entre { }
- Paires écrites au format "clef" : valeur
- Paires séparées par des virgules
- Valeurs autorisées :
  - booléen (true ou false)
  - nombres
  - chaîne de caractères entre guillemets
  - null
  - un objet
  - un tableau, c'est-à-dire
    - une suite de valeurs entre [ ]
    - séparées par des virgules

# S rialisation en JSON

## Utilisation de l'API li e   JSON

- Pour transformer un objet Javascript au format JSON :  
`JSON.stringify(obj)`
- Pour transformer une description JSON en un objet Javascript :  
`let obj = JSON.parse(chaine);`

# Comment définir une syntaxe ?

- **Backus-Naur Form**

- Les définitions de type « BNF »

- **Railroad diagrams**

- Un dessin vaut mieux qu'un long discours ?

# Méthode BNF

## Comment définir précisément une syntaxe ?

Premier standard : Backus-Naur Form (BNF)  
(il en existe plusieurs variantes)

`<chiffre> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

`<préfixe> ::= 0<chiffre> | 0<chiffre><chiffre>  
| 0<chiffre><chiffre><chiffre>`

`<numTelSimple> ::=  
    <chiffre><chiffre>.<chiffre><chiffre>.<chiffre><chiffre>`

`<numTel> ::= [ <préfixe> / ] <numTelSimple>`

# Méthode BNF

## Quelques éléments du format BNF...

- `<chiffre>` symbole non-terminal (qui est défini)
- `3` symbole terminal (à reprendre tel quel)
- `::=` symbole de la définition
- `|` sépare les choix ("ou")
- `[ ... ]` partie optionnelle
- `{ ... }` pour regrouper un ensemble de symboles
- `...*` partie qui peut être répétée 0, 1 ou plusieurs fois
- `...+` partie qui peut être répétée 1 ou plusieurs fois

`<préfixe> ::= 0<chiffre>[<chiffre>[<chiffre>]]`

`<paire> ::= <clef> : <valeur>`

`<objet> ::= { <paire> { , <paire> }* }`



# Méthode BNF

## Définition de la syntaxe JSON en BNF

`<objet> ::= { [ <paire> { , <paire> }* ] }`

`<paire> ::= <chaîne> : <valeur>`

`<tableau> ::= [ [ <valeur> { , <valeur> }* ] ]`

`<valeur> ::= <chaîne> | <nombre> | true | false | null  
| <tableau> | <objet>`

`<chaîne> ::= " <caractère>* "`

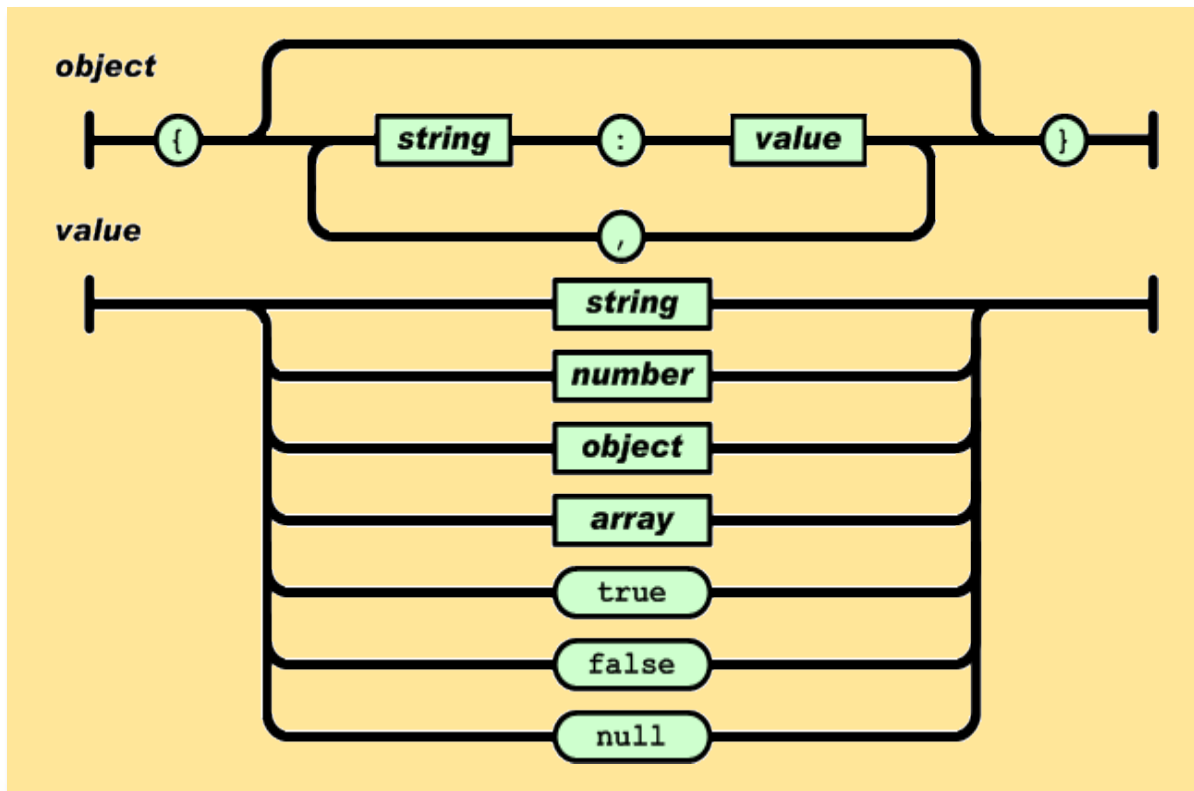
`<caractère> ::= <caractère Unicode standard>  
| \" | \\ | \n | \t | ...`

`<nombre> ::= [-] <chiffre>+ [.<chiffre>*]  
[ {e|E} [+|-] <chiffre>+ ]`

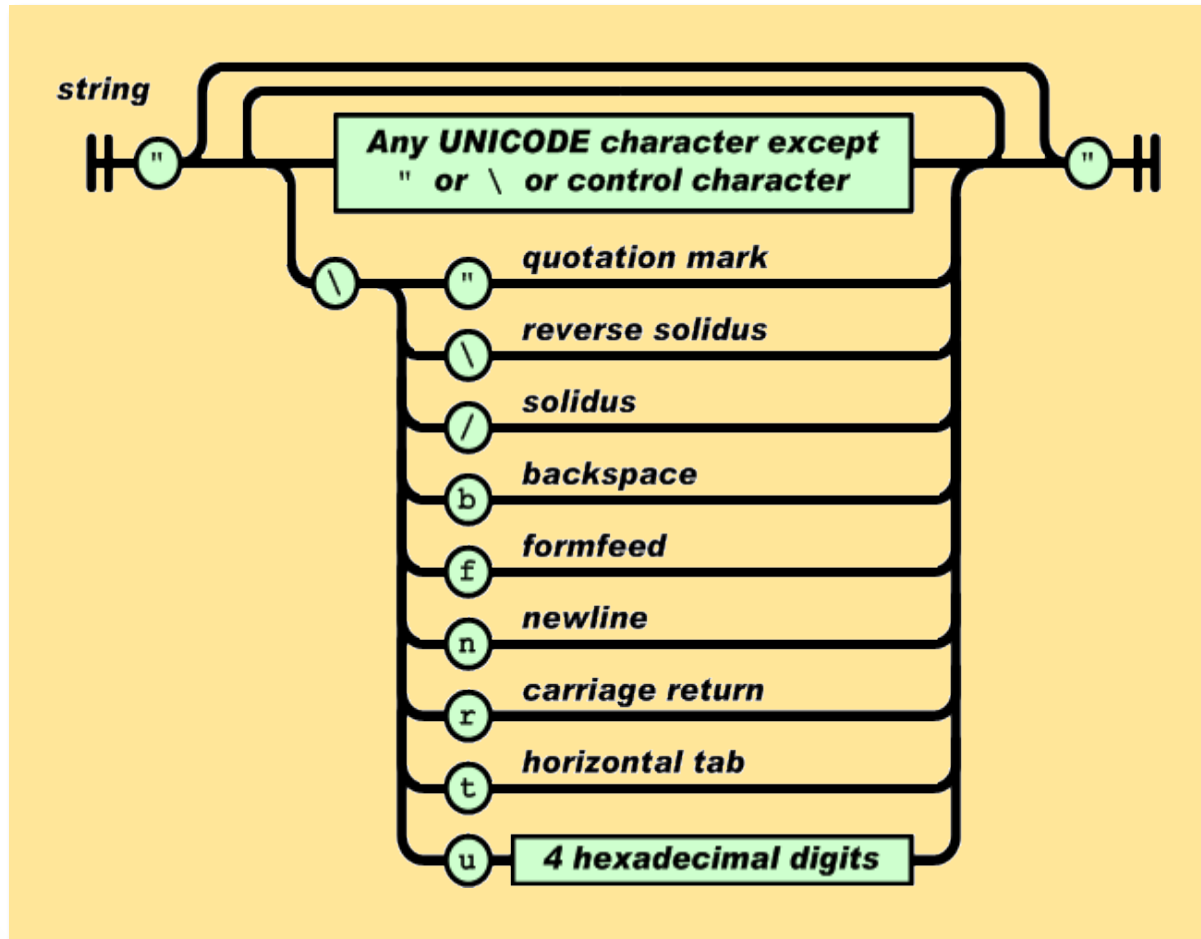
# Railroad diagrams

Comment définir précisément une syntaxe ?

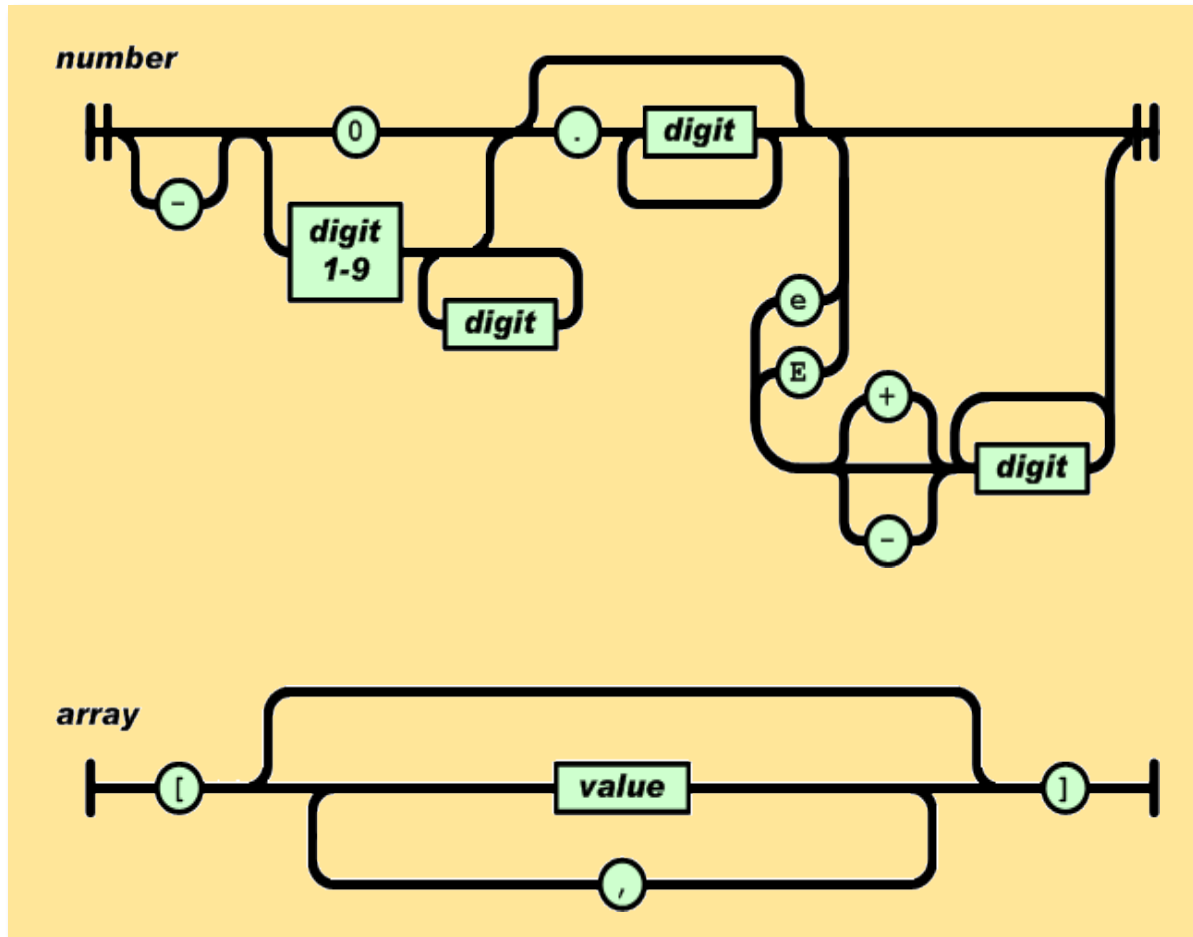
Second standard : diagramme syntaxique (source : json.org)



# Railroad diagrams



# Railroad diagrams



# Railroad diagrams

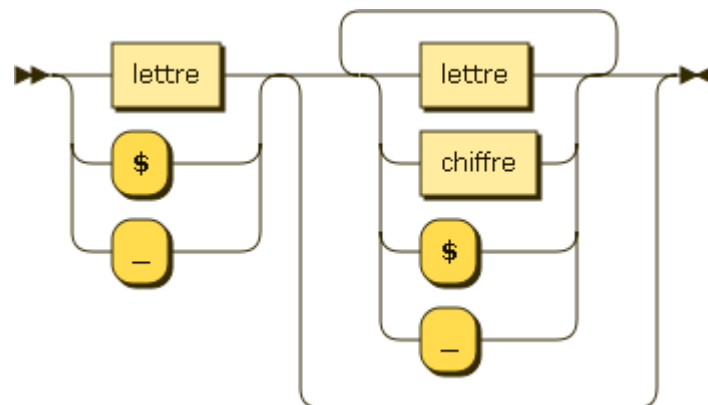
- Autre exemple

$\langle \text{lettre} \rangle ::= a \mid b \mid \dots \mid z \mid A \mid B \mid \dots \mid Z$

$\langle \text{chiffre} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{ident} \rangle ::= \{ \langle \text{lettre} \rangle \mid \$ \mid \_ \}$

$\{ \langle \text{lettre} \rangle \mid \langle \text{chiffre} \rangle \mid \$ \mid \_ \}^*$



# Compléments

➤ Mode strict et mode normal

➤ JSON

➤ L'orienté objet en ES6 

➤ jQuery (aperçu)

➤ Pour aller plus loin...

# Orienté objet en ES6

- **Nouvelle syntaxe**

- simplifiée (à première vue)
- similaire à ce qu'on retrouve dans d'autres langages
- qui cache la mécanique des prototypes

- Principaux éléments

- On définit des **classes** (pas d'attributs, juste des méthodes)

```
class Point { ... }
```

- On définit un **constructeur**.

```
constructor (x,y) { this.x = x; this.y = y; }
```

- On définit des **méthodes**.

```
distance () { return Math.sqrt(x * x + y * y); }  
toString () { return `(${this.x},${this.y})`; }
```

# Exemple

```
class Article {  
  constructor (nom, prix) {  
    this.nom = nom;  
    this.prix = prix;  
  }  
  prixGroupe (nombre) {  
    return nombre * this.prix;  
  }  
  achat (nombre) {  
    console.log(`${nombre} x ${this.nom} : `  
      + this.prixGroupe(nombre));  
  }  
}
```

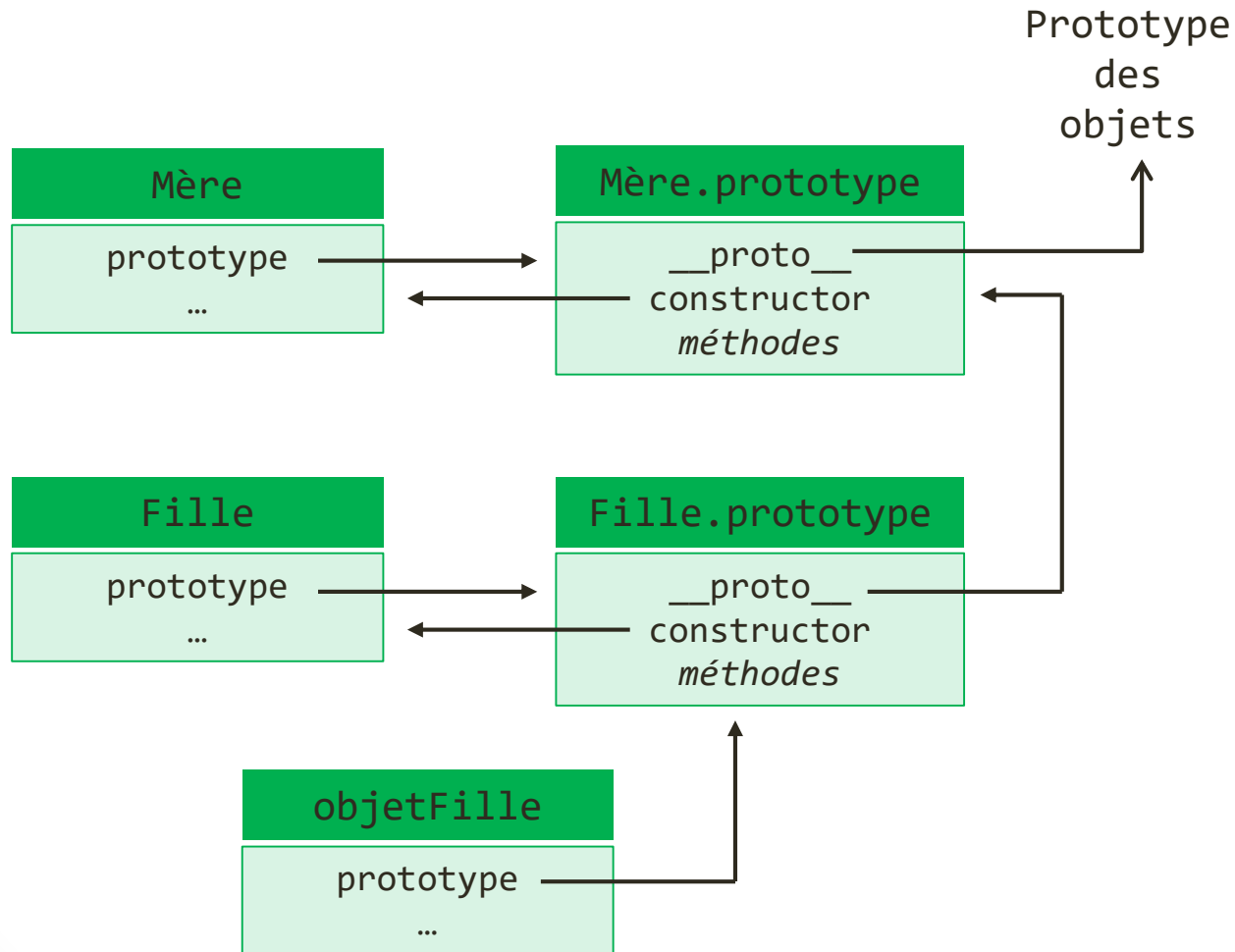
Automatiquement converti en :  
function Article (nom, prix) {...}

Méthodes automatiquement  
placées dans Article.prototype

```
let pc = new Article ("Pain au chocolat", 2.50);  
pc.achat(3);    // affiche : 3 x Pain au chocolat : 7.5
```



# "Sous-classe" en prototypal



# "Sous-classe" en ES6

- La nouvelle syntaxe facilite également "l'héritage entre classes"

- On peut préciser une classe-mère.

```
class PointColoré extends Point { ... }
```

- On peut redéfinir le constructeur.

```
constructor (x,y,couleur) {  
  super(x,y);  
  this.couleur = couleur;  
}
```

- On peut redéfinir des méthodes.

```
toString () {  
  return `${super.toString()} de couleur ${this.couleur}`;  
}
```

# Exemple

```
class Article {
  constructor (nom, prix) { this.nom = nom; this.prix = prix; }
  prixGroupe (nombre) { return nombre * this.prix; }
  achat (nombre) { console.log(`${nombre} x ${this.nom} : `
    + this.prixGroupe(nombre)); }
}

class ArticleSoldé extends Article {
  constructor (nom, prix, réduction) {
    super(nom, prix);
    this.réduction = réduction;
  }
  prixGroupe (nombre) {
    return super.prixGroupe(nombre) * (1 - this.réduction / 100);
  }
}

let pc = new ArticleSoldé ("Pain au chocolat", 2.50, 50);
pc.achat(3);    // affiche : 3 x Pain au chocolat : 3.75
```

En coulisse : même mécanique  
que le diagramme précédent !

# Compléments

➤ Mode strict et mode normal

➤ JSON

➤ L'orienté objet en ES6

➤ jQuery (aperçu) 

➤ Pour aller plus loin...

# jQuery en quelques mots

- **jQuery** est une bibliothèque définissant plusieurs fonctions utilitaires en Javascript permettant de :
  - manipuler le DOM (éléments HTML, événements...)
  - réaliser certains effets et animations standards
  - effectuer des échanges Ajax (voir plus loin),
  - le tout de manière indépendante des navigateurs !

- Rappel Javascript :

## Les identificateurs

- commencent par une lettre, \$ ou \_ et
- sont composés de lettres, de chiffres, de \$ et de \_

- jQuery tire parti du fait que \$ est un identificateur acceptable.

# jQuery en quelques mots

- Où trouver jQuery ?
  - <http://www.jquery.com>
  - Disponible en deux versions :
    - version de développement (code lisible)
    - version minimaliste (code illisible mais beaucoup plus court)
- Inclure jQuery (version locale)  
`<script src="jquery-chez-moi.js"></script>`
- Inclure jQuery (version partagée)  
`<script  
 src="http://code.jquery.com/jquery-latest.pack.js">  
</script>`  
*Pour tirer parti du cache (d'autres adresses standards existent)*

# Sélecteurs jQuery

- Version sans jQuery

```
document.getElementById("ident")
```

```
document.getElementsByTagName("h2")
```

```
document.getElementsByClassName("rouge")
```

- Version avec jQuery (sélecteurs utilisant la syntaxe CSS)

```
$("#ident")
```

```
$("h2")
```

```
$(".rouge")
```

```
$("p strong:first") : tous les premiers éléments "gras" des paragraphes
```

# Actions jQuery

- Un sélecteur jQuery peut retourner un ou plusieurs objets, sur le(s)quel(s) on peut accomplir diverses actions.

Ex : `$("#cadre").hide(); $("h2").hide();`

- La plupart des actions jQuery peuvent être effectuées sur un élément ou sur une liste d'éléments.
- jQuery permet également d'enchaîner les actions (chaque action est une fonction qui renvoie l'objet sur lequel elle porte).

Ex : `$("#cadre").hide().show(); // cache puis affiche`



# jQuery et DOM

- Obtenir des informations sur un élément
  - `.text()` : contenu textuel
  - `.html()` : contenu au format HTML (bases y compris)
  - `.val()` : valeur (d'un élément `<input>` par exemple)
  - `.attr("src")` : valeur de l'attribut "src"
- Donner une valeur / modifier un élément
  - `.text(contenu)`
  - `.html(contenu)`
  - `.val(valeur)`
  - `.attr(nom, valeur)`

# jQuery et DOM

- Modifier plusieurs éléments

- Si la valeur est la même

- `$(".rouge").text("Nouveau contenu");`

- Si la valeur n'est pas forcément la même : on utilise comme argument une fonction qui

- reçoit le numéro d'un élément et son ancienne valeur
    - renvoie la nouvelle valeur.

- `$(".rouge").html(function (i,old) { return old.toUpperCase(); });`

# jQuery et DOM

- Ajout d'éléments à l'arbre HTML
  - `.append(html)` ajoute l'argument à la fin du contenu de l'élément

```
$("p").append(" texte ajouté à la fin de chaque para");
$("#maliste").append("<li>Élément oublié</li>");
```
  - `.prepend(html)` ajoute au début du contenu de l'élément
  - `.before(html)` ajoute avant l'élément

```
$("img").before("<p>para ajouté avant chaque image</p>");
```
  - `.after(html)` ajoute après l'élément

*Note : ces quatre méthodes peuvent prendre plusieurs arguments (plusieurs éléments à ajouter).*

# jQuery et DOM

- Suppression d'éléments de l'arbre HTML
  - `.remove()` enlève l'élément (et ses enfants)
  - `.empty()` enlève les enfants de l'élément

# jQuery et CSS

- Manipulation des classes
  - `.addClass(classe)`
  - `.removeClass(classe)`
  - `.toggleClass(classe)`
  - `.hasClass(classe)`

*Note : l'argument est une chaîne de caractères avec un nom de classe (ou plusieurs noms séparés par des espaces).*

# jQuery et CSS

- Manipulation des propriétés CSS (style inline)
    - `.css("nom")` donne la valeur d'une propriété CSS
    - `.css("nom", "valeur")` donne une valeur à une propriété CSS
- Ex : `$(".rouge").css("font-weight", "bold");`

*Note : cela modifie les éléments (propriétés CSS inline), pas les classes !*

# Effets jQuery

- Afficher / cacher des éléments
  - `.hide()` et `.show()` : cacher/afficher les éléments
    - 1<sup>er</sup> argument : "slow", "fast" ou nombre de millisecondes
    - 2<sup>e</sup> argument : callback à exécuter à la fin
  - `.toggle()` : cache ou affiche selon l'état actuel
- Réaliser des fondus
  - `.fadeIn()`, `.fadeOut()`
    - 1<sup>er</sup> argument : "slow", "fast" ou nombre de millisecondes
    - 2<sup>e</sup> argument : callback à exécuter à la fin
  - `.fadeToggle()`
  - `.fadeTo(vitesse, opacité-cible, callback)`

# Effets jQuery

- Modification progressive du CSS
    - `.animate({paramètres CSS}, vitesse, callback)`
- Ex : `$("#cadre").animate({height:'150px', opacity:0.5});`  
`$("#cadre").animate({height:'+=50px'});`
- *Note : on peut enchaîner plusieurs animations (file d'attente)*



# jQuery et les événements

- Fonctions pour associer un événement à un (ou plusieurs) élément(s) :
  - `.click()`, `.dblclick()` : clic et double clic
  - `.mouseenter()`, `.mouseleave()` : passage de la souris
  - `.load()` : après le chargement (d'une image, du document)
  - ... et bien d'autres ...
- Ces fonctions prennent pour argument la fonction à appeler.  
`$("#monBouton").click(function () { alert("Clic !"); } );`
- La fonction peut faire référence à l'élément via `$(this)`.  
`$( "p" ).mouseenter(function () {  
 $(this).css("color", "red");  
});`

# jQuery et les événements

- Fonction combinant `onmouseenter` et `onmouseleave` :
  - `.hover(fonction1,fonction2)`
- Fonction à exécuter dès que toute la structure du DOM est en place :
  - `$(document).ready(fonction);`
  - Version abrégée : `$(fonction);`

```
$(function () { $("#bouton").click( ... ); });
```

- *Note : `.load(...)` s'exécute lorsque toutes les ressources (images par exemple) ont été chargées.*

# Compléments

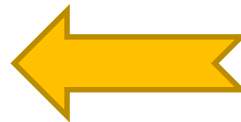
➤ Mode strict et mode normal

➤ JSON

➤ L'orienté objet en ES6

➤ jQuery (aperçu)

➤ Pour aller plus loin...



# Et après ?

- **Les closures**
  - Des fonctions avec des variables locales intégrées
  - Permet de simuler des attributs privés
- **Plus loin dans ES6**
  - Programmation modulaires (export/import entre modules)
  - Les symboles (des noms de propriétés garantis uniques)
  - Les itérateurs (de quoi créer des objets itérables)
  - Les promesses (indiquer à l'avance que faire lorsqu'une opération en cours sera terminée)
- **Bibliothèques Javascript**
  - Plus loin dans jQuery
  - AngularJS, ReactJS, EmberJS, NodeJS, ...