

Cours Javascript & PHP

L3 Informatique - UE Développement Web

David Lesaint

Université d'Angers

Août 2025



FACULTÉ
DES SCIENCES
*Unité de formation
et de recherche*

Plan

1

- UE Développement Web
 - Programme

2

- Javascript
 - Introduction
 - Bases
 - Les fonctions
 - La couche objet
 - Les objets natifs
 - Les classes
 - Les promesses
 - L'API Fetch
 - Web workers

3

- PHP
 - Les modules
 - Sécurité : CORS et CSP
 - Les API
 - L'API DOM
 - Introduction
 - Variables et typage
 - Chaînes de caractères
 - Tableaux
 - Structures de contrôle
 - Fonctions
 - Formulaire
- La couche Objet
 - Réflexion
 - Sérialisation
 - Cookies et sessions
 - Fichiers
 - APIs XML
 - Bases de données
 - Standardisation
 - Gestion de paquets PHP
 - Standard PHP Library
 - L'architecture MVC
 - Compléments

CM UE : Programme

Programme

Partie 1 : Javascript

- 5 CM de 1h20
- 5 TP de 2h40
- CC1 de 1h30 sur machine en semaine 41 coefficient 4/10

Partie 2 : PHP

- 5 CM de 1h20
- 5 TP de 2h40
- CC2 de 1h30 sur machine en semaine 48 coefficient 6/10

Groupes CM et TP

1 groupe CM

3 groupes de TP

- Groupe 1 : Corentin Behuet
- Groupes 2 et 3 : Sid Ali Mahmoudi
- Groupe 4 : D. Lesaint

Supports

Espace Moodle : n° 6924, clé 9dbvtj

- Groupe, calendrier, planning détaillé
- CM : livret (diaporama) + annexes
- TD et TP : livret des énoncés + annexes + corrections
- CC : énoncés + annexes + dépôts

CM JS : Introduction

Développer pour le Web suppose de ...

Maîtriser différents langages et paradigmes de programmation

- Contenu et style des pages web : HTML5, CSS, DOM, SVG ...
- Traitements côté client : JavaScript
- Traitements côté serveur : PHP, Python, Ruby, JSP, ..., MySQL ...
- Echange de données : JSON, XML

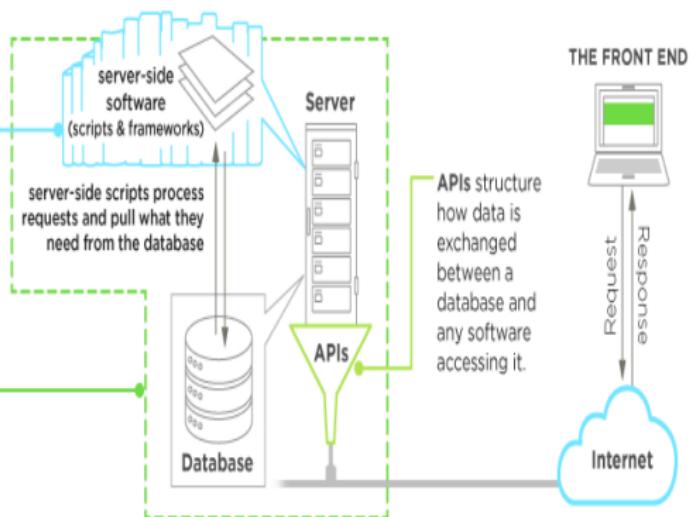
Développement Web : le “back-end”

BACK-END DEVELOPMENT & FRAMEWORKS IN SERVER SIDE SOFTWARE

Upwork®

FRAMEWORKS are libraries of server-side programming languages that construct the back-end structure of a site.

The “STACK” comprises the database, server-side framework, server, and operating system (OS).



APIs structure how data is exchanged between a database and any software accessing it.



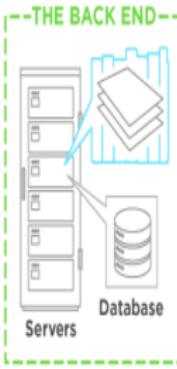
Développement Web : le “front-end”

FRONT-END DEVELOPMENT



- 1 A site is loaded in a browser from the server.


Responsive front-end design allows a site to adapt to a user's device.
- 2 Client-side scripts
Run in the browser and process requests without call-backs to the server
- 3 When a call to the database is required JavaScript and AJAX send requests to the back end.

- 4 The back-end server-side scripts process the request, pull what they need from the database then send it back.
- 5 Server-side scripts process the data, then update the site—populating drop-down menus, loading products to a page, updating a user profile, and more.


Développer pour le Web suppose de ...

Se conformer aux pratiques du Génie Logiciel

- Méthodes de conception objet : patrons, réflexion ...
- Méthodes agiles : développement piloté par les tests, intégration continue ...

Le Génie Logiciel représente l'ensemble des paradigmes, méthodes, techniques et outils destinés à mener à bien le développement d'un logiciel en respectant les contraintes **CQFD** (coûts, qualité, fonctionnalités, délais) du client.

S'adapter à un ensemble d'outils en constante évolution

- Bibliothèques : Bootstrap, jQuery, AngularJS ...
- Cadre : Symfony, Laravel, CodeIgniter ...
- Outils de développement : EDI (VSCode ...), outils de tests (PHPUnit ...), documentation (PHPDocumentor ...)

Périmètre du cours

Langages et outils

- JS, PHP
- JSON, XML

Méthodologie

- Conception orientée objet
- Patrons de conception MVC

ECMAScript

ECMAScript

- Ensemble de normes pour les langages de script (JavaScript, C++) standardisées par ECMA International.
- Spécification ECMA-262.
- **Dernière version : Edition 2025 (ES2025).**
- Quelques moteurs JS :
 - Spidermonkey (C/C++) : Firefox (Mozilla), navigateurs fondés sur KHTML (Konqueror - KDE)
 - V8 : Chrome/Chromium/Node.js (Google)
 - Javascriptcore : Safari (Apple)

Utilisation de JavaScript pour le navigateur

Code source en dur dans éléments `script`

Scripts à placer dans `<head>` ou `<body>`.

Fichiers importés par `scripts` avec attribut de localisation `src`

Import non bloquant si attribut `async` ou `defer` déclaré : exécution du code différée après construction du DOM

- Tous les scripts différés avec `defer` seront exécutés dans l'ordre de leur déclaration.

Modules importés par `scripts` le sont automatiquement en mode différé

Utilisation de JavaScript pour le navigateur

js.html

```
1 <!DOCTYPE html>
2 <html>
3   <title>Résultat : 1 5 6 7 8 3 4</title>
4 <head>
5 <script>console.log('Etape (1)');</script>
6 <script src="etape-2-async.js" async></script>
7 <script src="etape-3-defer.js" defer></script>
8 <script src="etape-4-defer.js" defer></script>
9 <script>console.log('Etape (5)');</script>
10 <script src="etape-6.js"></script>
11 </head>
12 <body>
13   <script>console.log('Etape (7)');</script>
14   <script src="etape-8.js"></script>
15 </body>
16 </html>
```

CM JS : Bases

Spécificités

Spécificités liées à la sécurité

- Ne peut pas lire ou écrire dans le système de fichier de la machine.
- Ne peut exécuter de programme externe.
- Pas de connexion autre qu'avec le serveur web.

Le chemin critique du rendu

Le chemin critique du rendu (Critical Rendering Path)

La séquence d'étapes du navigateur pour afficher initialement le contenu visible d'une page web à partir du document HTML renvoyé et des différentes ressources utilisées (CSS, images, ...)

- ① Analyse du HTML et construction de l'arbre DOM.
- ② Analyse du CSS et construction de l'arbre CSSOM.
- ③ Construction de l'arbre de rendu (render tree).
- ④ Mise en page (layout).
- ⑤ Peinture (painting).

Arbre de rendu, mise en page, peinture

L'arbre de rendu contient les éléments visibles et leurs styles

- Combine les arbres DOM et CSSOM.

La mise en page calcule positions et tailles des éléments visibles

- En fonction de la fenêtre active (viewport) = zone d'affichage disponible
 - Varie selon la taille et l'orientation de l'écran et le niveau de zoom.
 - Configurable avec balise `meta`, p. ex.

```
<meta name="viewport" content="width=device-width,  
initial-scale=1.0, user-scalable=yes">
```
- Selon les règles, requêtes media (@media) et unités CSS utilisées, notamment les unités relatives (% vw vh em rem).

La peinture dessine les pixels pour afficher le contenu visible

Optimisation du chemin critique

Points de blocage du rendu

- L'héritage en CSS impose une analyse complète du HTML.
- Un script JS bloque l'analyseur HTML le temps de son exécution.
- Certains scripts "exigent" que CSSOM ou DOM soient construits.

Charger les scripts non critiques de manière asynchrone

Avec `async` dans les `<script>` de chargement.

Différer le chargement des scripts critiques

Avec `defer` dans les `<script>` de chargement :

- Exécutera les fichiers JS une fois le code HTML analysé.
- Dans l'ordre de leur chargement (l'ordre des `<script>`).
- NB. Les modules sont chargés automatiquement en mode différé.

Les types

JS est un langage dynamiquement typé

7 types de données

6 types primitifs :

- boolean : booléens.
- number : entiers et nombres flottants.
- string : chaînes de caractères.
- Undefined : type d'une variable sans valeur.
- Null : type de ce qui est sans type ni valeur.
- symbol : symboles utilisables comme clés de propriétés anonymes d'objets (ES6).

1 type complexe :

- object : objets.

Valeurs primitives

Valeurs immuables qui ne sont pas des objets (de type object)

- number : 111 3.14
- string : 'Jean Dupont' "Jade Durant"
- boolean : true false
- undefined : valeur d'une variable sans valeur
- null : valeur d'une variable qui n'existe pas

js-core-undefined-null.js

```
1 typeof undefined // undefined
2 typeof null // object
3 null === undefined // false
4 null == undefined // true
5 x=2; y=null; z=undefined; x+y // 2
6 x=2; y=null; z=undefined; x+z // NaN
```

Valeurs non primitives

Objets ou valeurs qu'on peut modifier

- object : {name:"John", age:50}
- les fonctions sont un type d'objet particulier
 - function : function f2c(t) { return (5/9)*(t-32); }
- les tableaux sont de type object
 - tableau : var cars = ["Renault", "Citroen"];

js-core-complex-types.js

```
1 typeof [1,2,3,4]           // object
2 typeof {name:'John', age:34} // object
3 typeof function myFunc() {} // function
```

Les valeurs primitives sont passées et comparées par valeur tandis que les objets sont passés et comparés par référence Chaque type de valeur primitive, sauf null et undefined, possède un objet natif équivalent

Variables

Variables

- Identifiants : peuvent contenir lettres, chiffres, \$ et _ mais ne débutent pas par un chiffre
- Leur typage est dynamique
- Elles peuvent être déclarées à tout moment
- On peut utiliser les mots-clés var ou let pour les déclarer
- **On peut les redéclarer : leur valeur est préservée par défaut**

Eviter le \$ en début de nom de variable (utilisé par de nombreuses librairies JS)

Déterminer le type d'une variable avec `typeof`

js-core-typeof.js

```
1 var b=true;      console.log(typeof b); //boolean
2 var s='coucou';  console.log(typeof s); //string
3 var i=10;        console.log(typeof i); //number
4 var k;           console.log(typeof k); //undefined
5 var tab=[1,2];   console.log(typeof tab); //object
6 var p=null;      console.log(typeof p); //object
7 var reg=/\w+/;   console.log(typeof reg); //object
8 function f() {}  console.log(typeof f); //function
9 console.log(typeof none); //undefined
10 console.log(b); //true
11 console.log(s); //coucou
12 console.log(i); //10
13 console.log(k); //undefined
14 console.log(tab); //Array[1,2]
15 console.log(p); //null
16 console.log(reg); //\w+/
17 console.log(f); //function f() {}
```

Portée de variables

Portée d'une variable

L'espace du script dans laquelle elle va être accessible :

- Espace “global” : l'entièreté du script sauf l'intérieur des fonctions
- Espace “local” : l'intérieur d'une fonction

La portée d'une variable dépend

- D'où elle est déclarée : en dehors ou dans une fonction
- Du type de déclaration : avec `let`, `var` ou sans mot-clé

Règles

- Portée locale pour les variables déclarées dans les fonctions avec `let` ou `var`
- Globale pour les autres

Portée de variables

js-core-variable-scope-1.js

```
1 let xlet = "let x";
2 var xvar = "var x";
3 xglobal = "global x";
4
5 function externe() {
6   console.log(xlet); //let x
7   console.log(xvar); //var x
8   console.log(xglobal); //global x
9   xvar = "xvar modifiée";
10  xglobal = "xglobal modifiée";
11  let flet = "function : let";
12  var fvar = "function : var";
13  fglobal = "function : globale !!!";
14 }
15 externe();
16 console.log(xvar); //xvar modifiée
17 console.log(xglobal); //xglobal modifiée
18 //console.log(flet); // ReferenceError : flet is not defined
19 //console.log(fvar); // ReferenceError : fvar is not defined
20 console.log(fglobal); // function : globale !!
```

Portée de variables : fonctions internes

js-core-variable-scope-2.js

```
1 var xvar = "var x";
2
3 function externe() {
4     let elet = "externe : let";
5     var evar = "externe : var";
6
7     function interne() {
8         console.log(elet); // externe : let
9         console.log(evar); // externe : var
10        console.log(xvar); // externe : var x
11        let illet = "interne : let";
12        var ivar = "interne : var";
13    }
14    interne();
15    //console.log(illet); // ReferenceError : illet is not defined
16    //console.log(ivar); // ReferenceError : ivar is not defined
17 }
18
19 externe();
```

Quelques différences entre let/const et var

Notion de Temporal Dead Zone (TDZ)

Tout accès préalable à l'initialisation d'une

- constante `const` lève une erreur
- d'une variable `let` lève une erreur
- d'une variable `var` renvoie `undefined`

```
1 {  
2 // La TDZ démarre en début de bloc  
3 console.log(euler); // ReferenceError: euler is not defined  
4 console.log(bézout); // "undefined"  
5 console.log(fermat); // ReferenceError: Cannot access 'fermat' before  
initialization  
6 console.log(gauss); // ReferenceError: can't access lexical declaration 'gauss'  
before initialization  
7 euler = 1;  
8 var bézout = 2;  
9 let fermat = 3; // fin de la TDZ pour fermat  
10 const gauss = 4; // fin de la TDZ pour gauss  
11 }
```

Opérateurs

Opérateurs

Affectation	=	+=	-=	*=	/=	%=			
Arithmétiques	+	-	*	/	%	++	-		
Chaînes	+	+=							
Comparaison	==	=====	!=	!==	>	<	>=	<=	? :
Logiques	&&		!						
Binaires	&		~	^	<<	>>	>>>		
Types		typeof		instanceof					

Structures de contrôle

JS possède les structures de contrôle du langage C

- if/else
- switch/case
- for
- for/of pour les tableaux ou objets itérables (conteneurs)
- for/in pour les objets
- do/while
- while, break/continue
- throw/try/catch/finally

Propriétés et fonctions globales

Les propriétés globales de JS

- `Infinity` : une valeur de type `number` représentant $+/-\infty$
- `NAN` : valeur qui "n'est pas un nombre"
- `undefined` : valeur du type `undefined`

Propriétés et fonctions globales

Les fonctions globales

- `eval(string)` : évalue la chaîne comme du code JS
- `Number(var)` : convertit en nombre
- `String(var)` : convertit en chaîne de caractères
- `int parseInt(string[, radix])` : convertit en entier
- `float parseFloat(string)` : convertit en flottant
- `isFinite(var)` : teste si var est un nombre légal et fini
- `isNaN(var)` : teste si var est un nombre illégal
- `encodeURI(uri)` : encode un URI en UTF-8 (sauf caractères réservés, alphabet latin, chiffres, ...)
- `decodeURI(uri)` : décode un URI

Utilisation

js-core-function-global.js

```
1 console.log(parseInt('ff',16)); //255
2 var str=' 256';
3 var x = 1 + str;
4 console.log(x); //'1 256'
5 var x =1 + Number(str);
6 console.log(x); //257
7 console.log(eval('2 + 2 * 8 - 3')); //15
8 //JSON
9 var person=eval("({ name: 'Ada', age : 30 })");
10 console.log(person); // [object Object]
11 console.log(person.name+' '+person.age); // Ada 30
12 var uri=encodeURI('http://www.site.fr/x="10-2<9"');
13 console.log(uri); // http://www.site.fr/x=%2210-2%3C9%22
```

Divers - Débogage

Débogage

- Logging avec `console.debug(...)` ; ou variantes
`console.log()` `console.table()` ...
- Point d'arrêt pour débogage avec `debugger;`

Divers - Mode strict

La déclaration "use strict"; en début de script/fonction lancera une exception dans les cas suivants

- usage de variable/objet non déclaré
- destruction de fonction avec `delete`
- duplication de noms d'arguments de fonction
- écriture sur une propriété *read-only* ou *get-only*
- destruction d'une propriété indestructible
- usage de `eval` et `arguments` comme noms de variables
- création de variables via `eval` dans la portée de l'appel
- utilisation de mots-clés réservés (pour plus tard) : `implements`, `interface`, `let`, `package`, `private`, `protected`, `public`, `static`, `yield`

Déclaration inutile pour les modules (en mode strict par défaut)

Recommandations

- Eviter les variables globales
- Déclarer les variables locales (sinon, portée globale !)
- Placer les déclarations en début de script/fonction
- Initialiser les variables
- Préférer les types primitifs à Object, Number, String, Boolean
- Attention au transtypage automatique
- Préférer les opérateurs de comparaison === et !==
- Affecter des valeurs par défaut aux arguments de fonctions
- Terminer les switch avec default :
- Eviter eval()

CM JS : Les fonctions

Fonctions

Objets "fonction" de type `function`

- Elles peuvent être déclarées à tout moment avec le mot-clé `function`
- Les arguments peuvent ne pas être déclarés
- Type des arguments et valeur retour non déclarés
- Invoquées explicitement, en fonction d'évènements utilisateur, ou auto-exécutées (IIFE)
- Peuvent déclarer et retourner d'autres fonctions
- Variables passées par valeur, objets par référence

Fonctions

js-function.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><meta charset="UTF-8"/></head>
4 <body>
5 <p id="demo"></p>
6 <script>
7 function toCelsius(f) { return (5/9) * (f-32); }
8 document.getElementById("demo").innerHTML =
9   "This JS function<blockquote>
10    + String(toCelsius)
11    + "</blockquote>yields "
12    + String(toCelsius(50))
13    + "&#8451 for 64&#8457";
14 </script>
15 </body>
16 </html>
```

Déclaration optionnelle des arguments

js-function-arguments.js

```
1 function f() {  
2     console.log(arguments.length);  
3 }  
4 f(); //0  
5 f(1); //1  
6 f('riri','fifi','loulou'); //3
```

Les fonctions auto-exécutées (IIFE)

Immediately Invoked Function Expression (IIFE)

On peut déclarer et auto-exécuter une fonction en une instruction avec la syntaxe `(function() { ... })();`

js-function-selfinvoking.js

```
1 //fonction auto-exécutée
2 (function () {
3     console.log("Good "); return "morning";
4 })(); //affiche 'Good'
5
6 //fonction auto-exécutée
7 var f = (function () {
8     console.log("Good "); return "morning";
9 })(); //affiche 'Good'
10 console.log(f); //affiche 'morning'
```

Les fonctions imbriquées

Fonctions imbriquées

- Une fonction peut en contenir d'autres (c'est un objet)
- On peut ainsi définir des fonctions récursives ainsi que des fermetures

js-function-nested.js

```
1 // fonction externe auto-exécutée
2 f = (function() {
3   console.log("Good ");
4   // fonction interne
5   g = function() {
6     console.log("morning ...");
7   };
8   return g;
9 })(); // affiche 'Good'
10 f(); // affiche 'morning ...'
11 f(); // affiche 'morning ...'
```

Les fermetures (closures)

Fermetures

- Une fermeture est une fonction interne retournée par une fonction anonyme auto-exécutée
- La fonction interne a l'accès exclusif aux propriétés et méthodes de la fonction externe !
- Ces propriétés et méthodes sont “privées”

js-function-closure-1.js

```
1 var add = (function () {  
2     var counter = 0;  
3     return function () {return counter += 1; }  
4 })();  
5 add();  
6 console.log(add()); //2  
7 console.log(add.counter); //undefined
```

Les fermetures

js-function-closure-2.js

```
1 var add = (function () {
2   return function () {return this.counter += 1;}
3 })();
4 c1 = add.bind({'counter':0});
5 c2 = add.bind({'counter':20});
6 console.log(c1()); //1
7 console.log(c1()); //2
8 console.log(c2()); //21
9 console.log(c2()); //22
10 console.log(add()); //NaN
```

Les fermetures

js-function-closure-3.js

```
1 var f = function (val) {
2     var p = val;
3     return {
4         setT: function (newval) {p = "GG"+newval;},
5         setP: function (newval) {p = newval;},
6         getP: function () {return p; }
7     }
8 }
9 var some = f("one");
10 console.log(some.getP()); //one
11 some.setP("where");
12 console.log(some.getP()); //where
```

CM JS : La couche objet

La couche objet

Un objet JS est un ensemble cohérent de "propriétés" et "méthodes"

- Chaque propriété est nommée (nom de type `Symbol`) et a une valeur
- Une méthode est une propriété dont la valeur est une fonction (de type `Function`)

JS fournit des objets prédéfinis ("objets natifs") et on peut construire ses propres objets

Objets natifs de JS

Différentes catégories

- **Fondamentaux** : Object, Function, Symbol, Error ...
- **Nombres et dates** : Number, BigInt, Math, Date
- **Collections** : Array, Map, Set ...
- **Données structurées** : JSON ...
- **Contrôle d'abstraction** : Promise, Generator ...
- **Introspection** : Reflect, Proxy
- **Internationalisation** : Intl.DateTimeFormat, Intl.NumberFormat ...
- **Web Assembly** : WebAssembly.Module, WebAssembly.Instance ...

Propriétés et méthodes d'objets

Manipulation de propriétés/méthodes

- Accès à la valeur d'un membre dénommé `x` avec `obj.x` ou `obj["x"]`
 - dans le cas où `x` est une méthode, retourne la fonction correspondante
- Création/modification : `obj.x = ...;`
- Destruction : `delete obj.x;`

Méthodes d'objets et mot-clé `this`

Invocation de méthodes avec `obj.f(...)`

Le mot-clé `this` dans le corps d'une méthode est lié à l'objet sur lequel elle est invoquée

`this` dans le corps d'une fonction `f(...)` fait référence à l'objet

- Que l'on crée avec `new f(...)` quand `f` est utilisée comme constructeur
- ○ que l'on lie avec `f.bind(o)` dans le cas général

js-object-bind.js

```
1 function f(y) {  
2     return this.x + y;  
3 }  
4 let o = {"x":2};  
5 console.log(f.bind(o)(3)); //5
```

Prototypes et héritage

Chaque objet possède un “prototype”

- Le prototype est lui-même un objet créé automatiquement à la construction de l'objet
- Il est stocké dans une propriété spéciale de l'objet
- On y accède avec `Object.getPrototypeOf(o)` (et `o.__proto__` dans la console)

Tout objet hérite des propriétés/méthodes de son prototype

- Propriétés partagées par tous les objets ayant ce même prototype
- Propriétés distinctes des propriétés propres de l'objet (test : `o.hasOwnProperty(p)`)

Prototypes et héritage

Chaîne de prototypes

- Un prototype étant un objet, tout objet a donc une “chaîne de prototypes”
- Lorsqu'on accède à une propriété sur un objet, JS la recherche parmi les propriétés propres de l'objet, puis parmi celles de son prototype, puis celles du prototype de son prototype, . . . jusqu'à la trouver ou pas
- La chaîne se termine toujours sur `null` (le prototype de `Object.prototype`)

Création des prototypes

Tout constructeur `f` (natif ou non) possède une propriété de type objet dénommée `prototype`

- Ajoutée automatiquement au constructeur
- Contient initialement 2 propriétés : `constructor` et `__proto__`
- Accessible via `f.prototype`
- **Affecté automatiquement au prototype de tout objet construit (explicitement ou non) avec le constructeur**

```
» Person.prototype
  ↵ { ...
    | constructor: function Person()
    | <prototype>: { ...
      | _defineGetter_: function _defineGetter_()
      | _defineSetter_: function _defineSetter_()
      | _lookupGetter_: function _lookupGetter_()
      | _lookupSetter_: function _lookupSetter_()
      | __proto__: »
      | constructor: function Object()
```

```
» joe.__proto__
  ↵ { ...
    | constructor: function Person()
    | <prototype>: { ...
      | _defineGetter_: function _defineGetter_()
      | _defineSetter_: function _defineSetter_()
      | _lookupGetter_: function _lookupGetter_()
      | _lookupSetter_: function _lookupSetter_()
      | __proto__: »
      | constructor: function Object()
```

Héritage prototypique

js-object-addprototypeproperty.js

```

1 function Person(first) {
2   this.first = first;
3   this.last = "Dalton";
4 }
5 var joe = new Person('Joe');
6 Person.prototype.display = function() {
7   console.log(this.first + ' ' + this.last
+ ' - ' + this.job);
8 }
9 Person.prototype.job = "outlaw";
10 joe.display(); //Joe Dalton - outlaw
11 var jack = new Person('Jack');
12 jack.last = "Notlad";
13 jack.display(); //Jack Notlad - outlaw

```

```

» joe
↳ ▶ Object { first: "Joe", last: "Dalton" }

» joe.__proto__
← ▷ {...}
  ▶ constructor: function Person()
  ▶ display: function display()
    job: "outlaw"
  ▶ <prototype>: Object { ... }

» jack
↳ ▶ Object { first: "Jack", last: "Notlad" }

» jack.__proto__
← ▷ {...}
  ▶ constructor: function Person()
  ▶ display: function display()
    job: "outlaw"
  ▶ <prototype>: Object { ... }

```

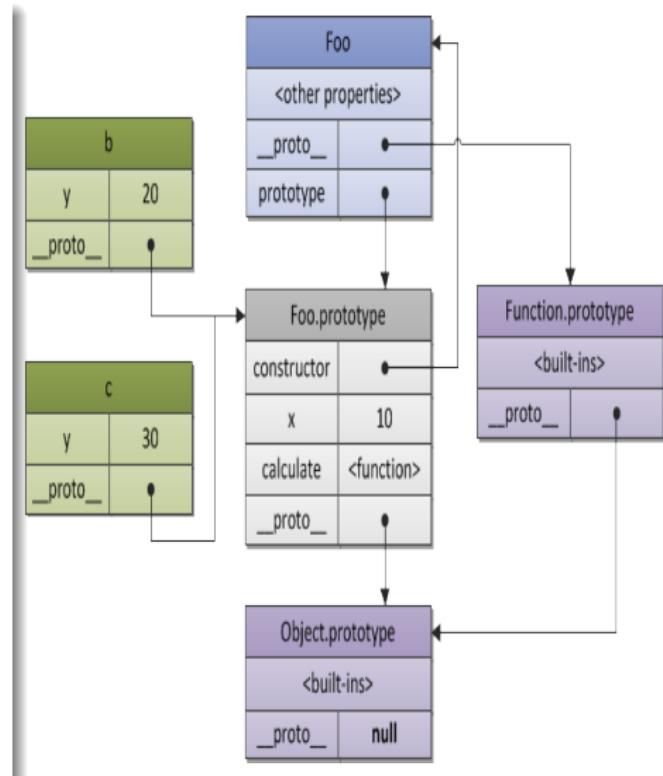
Héritage prototypique

js-object-proto-vs-prototype.js

```

1 // constructeur
2 function Foo(y) {
3   // propriété en propre pour chaque objet construit
4   this.y = y;
5 }
6 // propriété héritée par tout objet construit avec Foo
7 Foo.prototype.x = 10;
8 // méthode héritée par tout objet construit avec Foo
9 Foo.prototype.calculate = function (z) {
return this.x + this.y + z; };
10
11 var b = new Foo(20);
12 var c = new Foo(30);
13 b.calculate(30); //60
14 c.calculate(40); //80
15 console.log(
16   b.__proto__ === Foo.prototype, //true
17   c.__proto__ === Foo.prototype, //true
18   b.constructor === Foo, //true
19   c.constructor === Foo, //true
20   Foo.prototype.constructor === Foo, //true
21   b.calculate === b.__proto__.calculate,
// true
22   b.__proto__.calculate ===
Foo.prototype.calculate //true
23 );

```



Construction d'objets

Plusieurs possibilités pour construire des objets

- ① Créer un objet sous forme littérale JS
- ② Créer un objet à partir d'une chaîne au format JSON
- ③ Appeler un constructeur natif ou une fonction `f` avec le mot-clé `new`
 - `new f(...)`
- ④ Invoquer `Object.create(o)` où `o` sera le prototype de l'objet créé

Construction d'objet littéral avec propriétés seules

js-object-litteral-property.js

```
1 var joe = {  
2     first_name: "Joe",  
3     last_name: "Dalton",  
4     brothers: [{name:"Jack", age:30}, {name: "Averell",age: 33},  
{name: "William",age: 31}]  
5 };  
6 for (let p in joe) {  
7     if (joe[p] instanceof Array) //<=> joe[p].isArray()  
8         for (let i of joe[p])  
9             console.log(i.name);  
10    else  
11        console.log(`nom=${p} valeur=${joe[p]}`);  
12 }
```

Construction d'objet à partir de chaîne JSON

Format JSON (JavaScript Object Notation)

- Format de données texte qui permet la sérialisation et l'échange de données (objets sans méthodes, tableaux)
- Convertible en données JS avec l'objet `JSON`
- Format léger et mise en oeuvre simplifiée par rapport à XML
- Utilisé notamment pour récupérer les données par requêtes HTTP

js-object-json.js

```
1 var json = '{"first_name": "Joe", "last_name": "Dalton",
"brothers": [{"name": "Jack", "age": 30}, {"name": "Averell", "age": 33},
{"name": "William", "age": 31}]}';
2 var joe = JSON.parse(json);
3 for (let p in joe) {
4     if (joe[p] instanceof Array) // <=> joe[p].isArray()
5         for (let i of joe[p])
6             console.log(i.name);
7     else
8         console.log(`nom=${p} valeur=${joe[p]}`);
```

Construction d'objet littéral avec propriétés et méthodes

js-object-literal-method.js

```
1 var joe = {  
2     name: "Joe",  
3     display: function() {  
4         console.log(this.name);  
5     },  
6     brothers: [{  
7         name: "Jack",  
8         age: 30  
9     }, {  
10        name: "Averell",  
11        age: 33  
12    }, {  
13        name: "William",  
14        age: 31  
15    }]  
16 };  
17 joe.display();
```

Construction avec constructeur

Toute propriété `p` initialisée dans le constructeur avec `this.p` est une propriété propre de l'objet créé

js-object-prototype1.js

```
1 function Person(first) {  
2     this.first_name=first;  
3     this.last_name="Dalton";  
4     this.display=function () {  
5         console.log(this.first_name+ ' ' +this.last_name);  
6     };  
7 }  
8 let joe=new Person('Joe');  
9 let jack=new Person('Jack');  
10 joe.display(); //Joe Dalton  
11 jack.display(); //Jack Dalton  
12 jack.last_name = "Notlad";  
13 joe.display(); //Joe Dalton  
14 jack.display(); // Jack Notlad
```

Construction avec Object.create

On communique le prototype de l'objet à créer

Utile pour cloner un objet dont on ne connaît pas le constructeur

js-object-prototype2.js

```
1 function Person(first) {
2     this.first_name = first;
3     this.last_name = "Dalton";
4     this.display = function() {
5         console.log(this.first_name + ' ' + this.last_name);
6     };
7 }
8 let joe = new Person('Joe');
9 let jack = Object.create(joe);
10 jack.display(); // Joe Dalton
11 jack.first_name = "Jack";
12 jack.display(); // Jack Dalton
```

Parcourir les propriétés d'un objet

Itération sur les propriétés “énumérables”, propres ou héritées

Avec `for(let i in obj) { ...obj[i]... } !!pas obj.i!!`

Itérer uniquement sur les propriétés propres énumérables

Avec `Object.keys(obj)` : tableau des noms de ces propriétés

Accéder à toutes les propriétés propres, énumérables ou non

Avec `Object.getOwnPropertyNames(obj)` : tableau des noms de ces propriétés

js-object iterable.js

```
1 tab=[3];
2 console.log(tab.propertyIsEnumerable("0")); //true
3 console.log(tab.propertyIsEnumerable("tab.length")); //1
4 console.log(tab.propertyIsEnumerable("length")); //false
```

Mutabilité des objets

- Les objets sont passés par référence aux fonctions
- Deux objets construits avec le même constructeur et ayant les mêmes propriétés ne sont pas égaux

js-object-equality.js

```
1 function Person(first, last) {  
2     this.firstname=first;  
3     this.lastname=last;  
4 }  
5 var p1=new Person('Joe', 'Dalton');  
6 console.log(typeof p1); //object  
7 var p2=new Person('Joe', 'Dalton');  
8 console.log(p2==p1); //false !!  
9 var p3=p1; console.log(p3==p1); //true  
10 p3.firstname="Jack";  
11 console.log(p3==p1); //true
```

CM JS : Les objets natifs

Objets natifs

JS fournit un constructeur prédéfini (fonction) pour chaque type du langage et d'autres types utiles

Leurs prototypes contiennent des propriétés/méthodes dédiées

- Boolean, Number, String, Symbol
- Object
- RegExp (**expression régulière**), Array, Function

JS fournit aussi des objets globaux qui ne sont pas des fonctions

Semblables à des espaces de noms avec leurs propres constantes et fonctions

- Math, JSON ...

Constructeurs natifs

JS associe automatiquement un objet à chaque littéral

On peut utiliser sur un littéral les propriétés et méthodes du constructeur correspondant

```
var x = "une chaîne";
console.log(x.length);
console.log(x.toUpperCase())
```

Privilégier les littéraux pour accroître performances et lisibilité

- Object : var x1 = {};
- String : var x2 = "";
- Number : var x3 = 0;
- Boolean : var x4 = false;
- Array : var x5 = [];
- RegExp : var x6 = /() /;

L'objet Boolean

Boolean

Permet de représenter des valeurs booléennes

1 attributs :

- TRUE
- FALSE

2 méthodes :

- valueOf()
- toString()

Utiliser Boolean

js-native-boolean.js

```
1 var vrai=Boolean(true) ;
2 var faux=Boolean();
3 var p=Boolean.TRUE;
4
5 console.log(vrai); //true
6 console.log(faux); //false
7 console.log(typeof p); //undefined
8
9 console.log(vrai.toString()); //true
10
11 p=vrai.valueOf();
12 console.log('p=' + p); //p=true
13 p=vrai.toSource();
14 console.log('p=' + p); //p=(new Boolean(true))
```

L'objet Number

Number

Permet de représenter les nombres entiers et les réels

1 attributs :

- MAX_VALUE
- MIN_VALUE
- NaN
- NEGATIVE_INFINITY
- POSITIVE_INFINITY

2 méthodes :

- valueOf()
- toString(radix)
- toFixed(precision) notation avec chiffres après la virgule

Utiliser Number

js-native-number.js

```
1 var n1=new Number(255); // ou n1=255;
2 var n2=new Number(1.5); // ou n2=1.5;
3
4 console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
5 console.log(n1.toString()); // 255
6 console.log(n1.toString(2)); // 11111111
7 console.log(n1.toString(8)); // 377
8 console.log(n1.toString(16)); // ff
9
10 n1=n1*n2;
11 console.log(n1); // 382.5
12
13 console.log(n2.toString(16)); // 1.8
14 console.log(n2.toFixed(2)); // 1.50
```

L'objet String

String

Permet de représenter les chaînes de caractères

① attributs :

- `length` longueur de la chaîne

② méthodes :

- `integer charAt(index)`
- `String concat(s1, s2, ...)`
- `integer indexOf(search[, fromIndex])`
- `integer lastIndexOf(search[, fromIndex])`
- `String slice(begin, end)` **extrait la sous-chaîne**
- `Array split(separator[, limit])`
- `String substr(start, length)` **extrait la sous-chaîne**
- `String toLowerCase()`
- `String toUpperCase()`

Utiliser String

js-native-string.js

```
1 console.log('lassent'.length); //7
2 var text='les aléas de la semaine lassent';
3 console.log(text.indexOf('la'));//13
4 console.log(text.lastIndexOf('la'));//24
5 console.log(text.search('de'));//10
6 console.log(text.split(' ')); //les,aléas,de,la,semaine,lassent
7 console.log(text.slice(16,-8)); //semaine
8 console.log(text.replace('semaine','journée'));//les aléas de la
journée lassent
9 console.log(text.match(/l\w+g/)); //les,la,lassent
```

L'objet String et les expressions régulières

Méthodes en rapport avec les expressions régulières

Permettent de fouiller et modifier des chaînes de caractères à l'aide de regex

- array match(regex)
- boolean search(regex)
- boolean replace(regex, newstr or function)

L'objet RegExp

Regexp

Permet de représenter les expressions régulières

Méthodes :

- `test(string)` retourne `true` si il y a correspondance
- `exec(string)` retourne un tableau qui contient les motifs correspondant à l'expression

Rappel sur les expressions régulières

Une regex suit la syntaxe /pattern/modificateur

L'expression du pattern peut utiliser des

- crochets et parenthèses
- métacaractères
- quantificateurs

Rappel sur les expressions régulières

Modificateurs

q dans une regex /pattern/q prenant la/les valeur(s) :

- g : recherche toutes les correspondances (arrêt à la première par défaut)
- i : recherche insensible à la casse
- m : recherche multi-lignes

Crochets et parenthèses

Correspondance (*match*) avec un ensemble de caractères :

- [abc] : n'importe quel caractère parmi {a, b, c}
- [0-9] : n'importe quel chiffre
- x | y : l'expression x ou l'expression y

Rappel sur les expressions régulières

Méta-caractères

Exemple de méta-caractères :

- \d : un chiffre
- \s : un espace
- \w : un mot
- \b : début ou fin de mot

Quantificateurs

Exemples de quantificateurs :

- x+ : correspondance avec au moins une occurrence de x
- x? : correspondance avec 0 ou 1 occurrence de x
- x{m, n} : correspondance avec $y \in [m, n]$ répétitions de x

Utiliser Regexp

js-native-regexp.js

```
1 var regex1=/\d+;
2 var regex2=new RegExp (" (\d+) ", "g");
3 var regex3=/(\d+)/|(\d+)/|(\d+)/;
4 var dob='30/09/1970';
5 console.log(dob.match(regex1)); //30
6 console.log(dob.match(regex2)); //30,09,1970
7 console.log(regex2); //(\d+)/g
8 console.log(regex3); //(\d+)/|(\d+)/|(\d+)/
9 console.log(regex3.test(dob)); //true
10 console.log(regex3.exec(dob)); //30/09/1970,30,09,1970
11 console.log(dob.replace(regex3,' $3-$2-$1')); //1970-09-30
```

L'objet Math

Math n'est pas une fonction

Math - les propriétés

Permet de réaliser des calculs complexes

- E : constante d'Euler (2,718)
- LN2 : logarithme népérien de 2 (0,693)
- LN10 : logarithme népérien de 10 (2,302)
- LOG2E : logarithme à base 2 de E (1,442)
- LOG10E : logarithme à base 10 de E (0,434)
- PI : valeur du nombre PI (3,14159)
- SQRT1_2 : racine carrée de 1/2 (0,707)
- SQRT2 : racine carrée de 2 (1,414)

L'objet Math

Math - les méthodes

- abs
- acos, asin , atan,, atan2
- ceil, floor
- cos, sin, tan
- exp, log, pow
- min, max
- random : retourne une valeur entre 0 et 1
- round
- sqrt

Utiliser Math

js-native-math.js

```
1 console.log(Math.cos(Math.PI)); // -1
2 console.log(Math.sin(Math.PI)); // 1.2246467991473532e-16
3
4 console.log(Math.sqrt(2)); // 1.4142135623730951
5 console.log(Math.pow(2,10)); // 1024
6
7 console.log(Math.round(Math.sqrt(2))); // 1
8 console.log(Math.random()); // 0.3946604376730234
9
10 console.log(Math.ceil(1.4));
11 console.log(Math.ceil(1.8));
12 console.log(Math.floor(1.4));
13 console.log(Math.floor(1.8));
14 console.log(Math.round(1.4));
15 console.log(Math.round(1.8));
```

L'objet Date

Date - les méthodes

Permet de représenter les dates

- Date()
- Date(milliseconds)
- Date(y, m, d)
- getDay(), getMonth(), getYear(), getFullYear()
- getHours(), getMinutes(), getSeconds()
- toGMTString
- toLocaleString
- toUTCString

Utiliser Date

js-native-date.js

```
1 var today=new Date();
2 console.log(today); //Fri Feb 13 2009 16:27:30 GMT+0100 (GMT+01:00)
3 console.log(today.toLocaleDateString()); //fevrier 13, 2009
4 console.log(today.toUTCString()); //Fri, 13 Feb 2009 15:38:52 GMT
5 var origin = new Date(0);
6 console.log(origin.toGMTString()); // Thu Jan 01 1970 01:00:01 GMT+0100
(GMT+01:00)
7 console.log(new Date(2009,0,15)); // Thu Jan 15 2009 00:00:00 GMT+0100
(GMT+01:00)
8
9 console.log(Date.parse(today.toString())); // 1234539588000
```

L'objet Array

Array - propriétés et méthodes

Permet de représenter et traiter les tableaux

- `length`
- `Array concat(array1, array2)`
- `string join(string)`
- `push` : ajoute un nouvel élément à la fin
- `unshift` : ajoute un nouvel élément au début
- `pop` : supprime et retourne le dernier élément
- `shift` : supprime et retourne le premier élément
- `reverse` : inverse l'ordre des éléments

L'objet Array

Array - autres méthodes

- `Array(n)` : construit un tableau de longueur n
- `Array(n1, ..., nk)` : construit un tableau dont les éléments sont n_1, \dots, n_k
- `slice(begin, end)` : extrait une partie du tableau
- `splice(index, howMany, e1, ..., en)` : ajoute et supprime de nouveaux éléments
- `sort()` : tri des éléments
- `sort(f)` : tri suivant la fonction de comparaison binaire f retournant -1, 0 ou 1

Utiliser Array

js-native-array.js

```
1 var a=[3,8,2,7];
2 console.log(a.length); //4
3 console.log(a.reverse()); //7,2,8,3
4 a.sort(); console.log(a); //2,3,7,8
5 var a=new Array(1,7,3,7,8,2,1);
6 for (i in a) {
7   console.log(i+' '+a[i]); //0 1, 1 7, ...
8 }
9 console.log(a.join(';')); //1;7;3;7;8;2;1
10 var b=a.slice(2,4); console.log(b); //3,7
11 a.splice(1,2,20);
12 console.log(a); //1,20,7,8,2,1
13 function compare(a,b) {
14   return a-b;
15 }
16 a.sort(compare); console.log(a); //1,1,2,7,8,20
```

CM JS : Les classes

Syntaxe des classes

Syntaxe introduite depuis ES2015 pour permettre une programmation orientée objet plus classique

Les classes sont juste une abstraction des mécanismes JS existants fondés sur les prototypes

Caractéristiques

- 1 constructeur par classe
- Champs (propriétés ou méthodes) d'instances ou statiques
- Champs publiques ou privés
- Accesseurs et mutateurs
- Héritage simple
- Mixins pour simuler l'héritage multiple de classes abstraites

Trame syntaxique et correspondance en termes d'objets

js-class-declaration.js

```
1 class A {  
2     constructor() { ... }           // constructeur  
3     afield = "foo";                // propriété d'instance publique  
4     aMethod() { ... }              // méthode d'instance publique  
5     static aStaticField = "bar";    // propriété statique publique  
6     static aStaticMethod() { ... }  // méthode statique publique  
7     static { ... }                // bloc statique d'initialisation  
8     #aPrivateField = "bar";        // propriété d'instance privée  
9 }
```

js-class-declaration-traduction.js

```
1 function A() {  
2     this.aField = "foo";  
3     ...  
4 }  
5 A.prototype.aMethod = function () { ... };  
6 A.aStaticField = "bar";
```

Constructeur

La méthode `constructor` fournit le constructeur d'instances d'une classe

Equivalent aux fonctions constructeurs de JS.

Un seul constructeur autorisé

- Non-obligatoire : un est fourni par défaut
- Nécessairement public : utiliser un booléen statique privé pour simuler un constructeur privé (p. ex. pour classe singleton)
- Si la classe est dérivée, il doit invoquer celui de la classe parente avec le mot-clé `super`

js-class-constructor.js

```
1 class Rectangle {  
2     constructor(hauteur, largeur) {  
3         this.hauteur = hauteur; // propriété propre de chaque instance  
4         this.largeur = largeur; // propriété propre de chaque instance  
5     }  
6 }
```

Construction d'instance

Construction d'instance avec `new maClasse(...)`

- Une classe doit être déclarée avant d'être instanciée : pas de remontée de déclaration (hoisting) à l'instar des déclarations `let` et `const`

`js-class-construction.js`

```
1 const r = new Rectangle(4,5); // ReferenceError: can't access lexical declaration
'Rectangle' before initialization
2 class Rectangle {
3     constructor(hauteur, largeur) {
4         this.hauteur = hauteur;
5         this.largeur = largeur;
6     }
7 }
8 const p = new Rectangle(4,5);
9 console.log(p.hauteur); //4
```

Propriétés d'instances

Une propriété définie pour chaque instance de la classe (de même clé) mais dont la valeur est propre à l'instance

Equivalent aux propriétés propres initialisées dans une fonction constructeur

Déclaration/initialisation dans le constructeur et/ou en dehors mais dans la classe

- Initialisation optionnelle : en dehors du constructeur revient à lui affecter une valeur par défaut que le constructeur peut modifier
- Sans const, let ou var
 - La variabilité d'une propriété se contrôle avec son descripteur
 - Par défaut, une propriété est inscriptible

Préfixer un identifiant de propriété par # rend la propriété privée
Uniquement accessible depuis l'instance

Propriété d'instances

js-class-properties.js

```
1 class Rectangle {
2     height = 0; // propriété publique avec valeur par défaut
3     #width; // propriété privée
4     constructor(height, width) {
5         this.height = height; this.#width = width;
6     }
7 }
8 let r = new Rectangle(2,3);
9 console.log(r.height); //2
10 console.log(r.width); //undefined
11 console.log(new Rectangle(1,1).height==r.height); //false
```

Méthodes

Définies dans la classe par une fonction `f(...){...}`

Equivalent aux méthodes définies sur les prototypes d'objets

- Partagées par toutes les instances de la classe
- Définies sur le prototype de ces instances
- Publiques ou privées (noms préfixés par #)
- De différentes natures : classiques, asynchrones, générateurs ...

Méthodes

js-class-methods.js

```
1 class Rectangle {  
2     constructor(height, width) {  
3         this.height = height;  
4         this.width = width;  
5     }  
6     calcArea() { //méthode publique  
7         return this.height * this.width;  
8     }  
9     *getSides() { //méthode privée (générateur)  
10        yield this.height; yield this.width; yield this.height; yield  
this.width;  
11    }  
12 }  
13 const r = new Rectangle(5, 2);  
14 console.log(r.calcArea()); //10  
15  
console.log(Object.getPrototypeOf(square).hasOwnProperty('calcArea'));  
//true  
16 //syntaxe de décomposition (spread syntax) appliquée à un générateur (objet itérable)  
17 console.log([...square.getSides()]); // [5, 2, 5, 2]
```

Champs statiques

Définis avec `static p;` ou `static f(...){...}`

Equivalent aux propriétés/méthodes propres d'une fonction constructeur

- Champs propres à la classe, pas à ses instances
- Publiques ou privés si préfixés par #

js-class-static.js

```
1 class Point {  
2   constructor(x, y) {  
3     this.x = x;  
4     this.y = y;  
5   }  
6   static displayName = "Point";  
7   static distance(a, b) {  
8     const dx = a.x - b.x;  
9     const dy = a.y - b.y;  
10    return Math.hypot(dx, dy);  
11  }  
12 }  
13 const p1 = new Point(5, 5);  
14 const p2 = new Point(10, 10);  
15 p1.displayName; // undefined  
16 p1.distance; // undefined
```

Accesseurs

Méthodes d'accès à pseudo-propriétés d'objets

Une pseudo-propriété se définit par une fonction accesseur (getter) et/ou mutateur (setter)

- `get prop () { ... } : accesseur de la pseudo-propriété "prop"`
- `set prop (v) { ... } : mutateur de la pseudo-propriété "prop"`

Accesseur/mutateur publics d'une classe C

- Rattachés au prototype `C.prototype` des instances de C
- Automatiquement invoqués sur une instance o à chaque accès à la propriété (sauf si masqués par une propriété d'instance de même clé)
 - o.prop invoque `prop ()` sur o
 - o.prop = v; invoque `prop (v)` sur o

Accesseurs

Intérêt

- Eviter la lourdeur des méthodes et appels du type `o.getProp()`
- Calculer des propriétés à la volée

js-class-accessor-fields.js

```
1 class Color {
2   constructor(r, g, b) {
3     this.values = [r, g, b];
4   }
5   get red() {
6     return this.values[0];
7   }
8   set red(value) {
9     this.values[0] = value;
10  }
11 }
12 const o = new Color(255, 0, 0);
13 o.red = 88; //invocation du setter 'red(88)' sur 'o'
14 console.log(o.red); // invocation du getter 'red()' sur 'o' : 88
15 const p = new Color(0, 0, 0);
16 console.log(p.red); //0
```

Pseudo-propriétés et propriétés

js-class-accessor-fields-1.js

```
1 "use strict";
2 class C {
3   // propriété propre publique "p" masquant getter/setter d'une pseudo-propriété "p"
4   p = 0;
5   // propriété propre privée "#q" distincte de la pseudo-propriété "q"
6   #q = 2;
7   // getter/setter rattachés au prototype C.prototype
8   get q() {
9     console.log("getter");
10    return this.#q;
11  }
12  set q(x) {
13    console.log("setter");
14    return this.#q = x;
15  }
16 const o1 = new C(), o2 = new C();
17 console.log(o1.hasOwnProperty("p")); //true
18 console.log(o1.hasOwnProperty("#q")); //false
19 console.log(o1.hasOwnProperty("q")); //false
20 console.log(o1.q); //getter 2
21 o1.q = 3; //setter
22 console.log(o1.q); //getter 3
23 console.log(o2.q); //getter 2
24 console.log(Object.getOwnPropertyDescriptor(o1, "p")); //Object { value: 0, writable: true, enumerable: true, configurable: true }
25 console.log(Object.getOwnPropertyDescriptor(Object.getPrototypeOf(o1), "q")); //Object { get: q(), set: q(x), enumerable: false, configurable: true }
```

Héritage

Héritage avec `class Child extends Parent {...}`

- Appel recommandé du constructeur parent avec `super(...)`
- Accès à propriétés/méthodes parentes avec `super. ...`

`js-class-inheritance.js`

```
1 class A {
2   constructor(a) {
3     this.a = a;
4   }
5 }
6 class B extends A {
7   constructor(a,b) {
8     super(a);
9     this.b = b;
10  }
11 }
12 const o = new B(0,1);
13 console.log(o);
14 console.log(Object.getPrototypeOf(o)==B.prototype); // true
15 console.log(Object.getPrototypeOf(B.prototype)==A.prototype); // true
```

Blocs d'initialisation statiques

Définis avec `static { ... }`

- Plusieurs blocs autorisés par classe
- Exécutés à l'initialisation de la classe
- Ont accès aux champs statiques, publics ou privés

js-class-static-block.js

```
1 class A {
2     static staticProperty1 = 'Property 1'; // valeur par défaut
3     static staticProperty2;
4     static {
5         this.staticProperty2 = 'Property 2';
6     }
7 }
8
9 console.log(A.staticProperty1); // "Property 1"
10 console.log(A.staticProperty2); // "Property 2"
```

Descripteurs de propriétés

2 types de descripteurs existent

Descripteur de données (data descriptor)

- Pour propriété possédant une valeur accessible ou non en écriture

Descripteur d'accesseur (accessor descriptor)

- Pour pseudo-propriété décrite par accesseur et/ou mutateur

Un descripteur est un objet

- **Accès avec** `Object.getOwnPropertyDescriptor(objet, clé-propriété)`
- **Création/modification avec** `Object.defineProperty(objet, clé-propriété, descripteur)`

Descripteurs de propriétés

Tout descripteur indique si la propriété est

- Configurable : on peut modifier le descripteur lui-même ou supprimer la propriété avec `delete`
- Enumérable : elle apparaît lorsqu'on itère sur les propriétés de l'objet avec `for ... in`

Un descripteur de données peut optionnellement

- Fournir la valeur par défaut associée à la propriété (`undefined` sinon)
- Indiquer si la propriété est inscriptible (`writable`) ou non

Un descripteur d'accesseur peut optionnellement

- Fournir le getter qui renverra la valeur de la pseudo-propriété à chaque appel
- Fournir le setter qui fixera la valeur de la pseudo-propriété à chaque appel

Descripteurs de données

js-class-descriptor-data.js

```
1 "use strict";
2 class C {
3   constructor(v) { this.foo = v; }
4 }
5 const o = new C(0);
6 var d = Object.getOwnPropertyDescriptor(o, "foo");
7 console.log(d);
8 // { configurable: true, enumerable: true, value: 0, writable: true }
9 console.log(o.foo); // 0
10 o['foo'] = 1;
11 Object.defineProperty(o, "bar", {
12   configurable: false, // valeur par défaut
13   enumerable: false, // valeur par défaut
14   value: "2",
15   writable: false // valeur par défaut
16 });
17 for(let k in o){console.log(k+':'+o[k])} // foo:1
18 console.log(o.bar); // 2
19 o['bar'] = 3; // TypeError: "bar" is read-only [sans effet en mode non-strict]
20 delete o.bar; // TypeError: property "bar" is non-configurable and can't be deleted
[sans effet en mode non-strict]
```

Descripteurs d'accesseurs

js-class-descriptor-accessor.js

```
1 "use strict";
2 class C {
3     constructor(v) { this.v = v; }
4     get foo() { return this.v; }
5     set foo(x) { this.v = x; }
6 }
7 const o = new C(1);
8 var d = Object.getOwnPropertyDescriptor(Object.getPrototypeOf(o),
"foo");
9 console.log(d);
10 // { configurable: true, enumerable: false, get: foo(), set: foo(x) }
11 Object.defineProperty(o, "bar", {
12     configurable: true,
13     enumerable: true,
14     get: function () { return this.v; },
15     set: function (x) { this.v += x; },
16 });
17 o.bar = 2;
18 console.log(o.bar); // 3
19 d = Object.getOwnPropertyDescriptor(o, "bar");
20 console.log(d);
```

Expressions de classes

Expressions de classes nommées ou anonymes

Semblables aux expressions de fonctions mais sans remontée de déclaration

Si nommée, permet de faire référence au nom de la classe

- Uniquement dans la classe
- Pas de `self` : en JS, fait référence à la portée globale (fenêtre ou worker)

Expressions de classes

js-class-expression.js

```
1 let RectangleA = class { // expression de classe anonyme
2     constructor(h, l) {
3         this.h = h;
4         this.l = l;
5     }
6 };
7 const a = new RectangleA(4,5);
8 console.log(a.h); //4
9
10 let RectangleB = class Rectangle { // expression de classe nommée
11     constructor(h, l) {
12         this.h = h;
13         this.l = l;
14         this.c = Rectangle.pgcd(this.h, this.l); // référence à la classe
15     }
16     static function pgcd(m,n) {n==0 ? m : Rectangle.pgcd(n,m mod n);}
17 };
18
19 const b = new RectangleB(3,2);
20 console.log(b.l); //2
21 const c = new Rectangle(3,2); // ReferenceError: Rectangle is not defined
```

CM JS : Les promesses

Environnement d'exécution JS

Programmation asynchrone

Fil d'exécution (alias thread, tâche) : Exécution d'un ensemble d'instructions du langage machine d'un processeur au sein d'un processus.

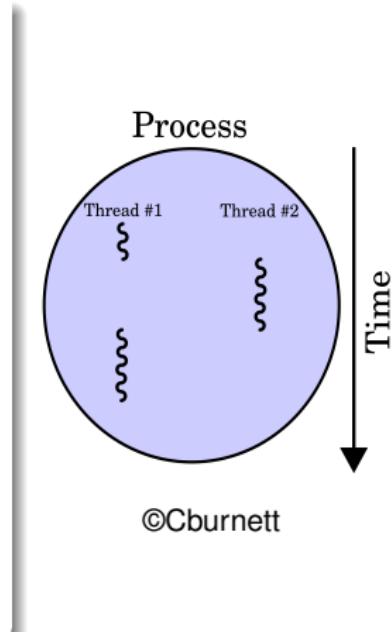
- Possède sa propre pile d'exécution.
- Partage la mémoire virtuelle du processus avec d'autres fils.

Exemples :

- Interaction avec l'utilisateur gérée par un fil.
- Calculs lourds gérés par plusieurs fils.

Avantages :

- Plus de blocage durant les phases de traitement lourd, les requêtes sortantes, etc.
- Parallélisation possible sur les architectures multi-processeurs.



©Cburnett

Programmation asynchrone en JS

Quels besoins ?

De nombreuses API doivent exécuter du code de manière asynchrone :

- Récupération de données sur un serveur (images, polices, scripts, texte).
- Requêter une base de données.
- Accéder au flux vidéo d'une webcam.
- Relayer l'affichage à un casque de réalité virtuelle (VR), etc.

JS est mono-tâche

Un seul fil (main thread) et de possibles situations de blocage :

- Téléchargements.
- Invocations de services distants.
- Longues boucles de calcul, etc.

Blocage : exemple

async-simple-sync.js

```
1 document.querySelector('button').addEventListener('click', f);
2
3 function f(e) {
4     let myDate;
5     Array(10000000).keys().forEach(() => myDate = new Date());
6     console.log(myDate);
7     let pElem = document.createElement('p');
8     pElem.textContent = 'This is a newly-added paragraph.';
9     document.body.appendChild(pElem);
10 }
```

Environnement d'exécution JS

Environnement d'exécution

La pile d'appels (*stack*)

- On y empile les cadres (*frames*).
- Chaque cadre stocke les arguments et variables locales d'un appel de fonction.
- On dépile le cadre du dessus lorsque la fonction associée a terminé.

Le tas (*heap*)

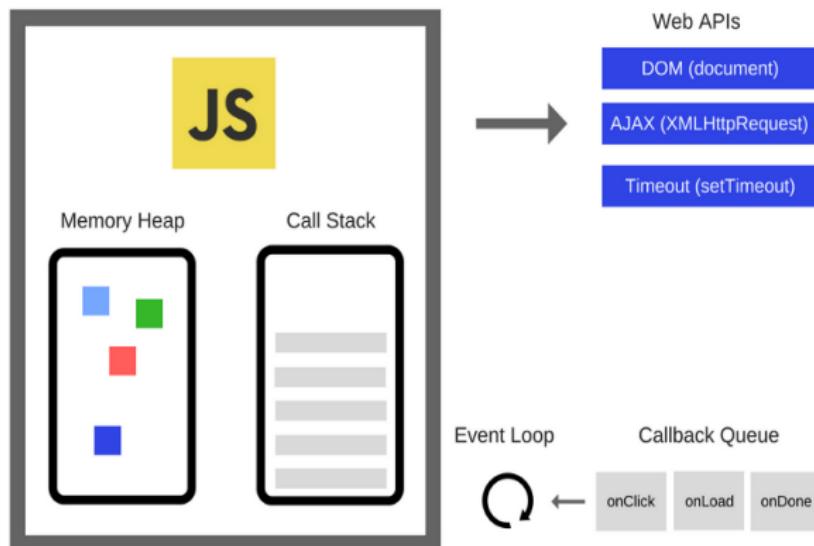
- On y alloue/désalloue de la mémoire pour les objets.

La file de messages (*event queue*)

- Les nouveaux messages y sont ajoutés à la fin dans l'ordre d'arrivée.
- Le premier message sera traité intégralement puis supprimé avant de boucler.

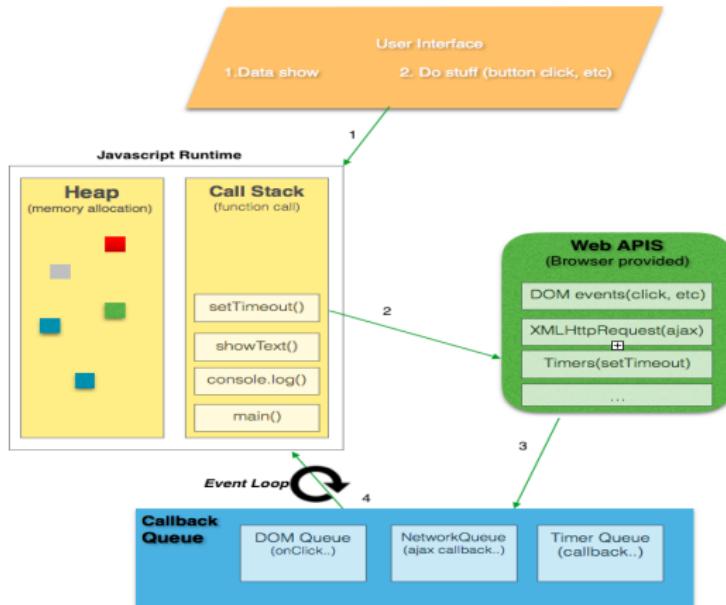
La boucle d'évènements JS (*event queue*)

Pile, tas, file et Web APIs



La boucle d'évènements JS

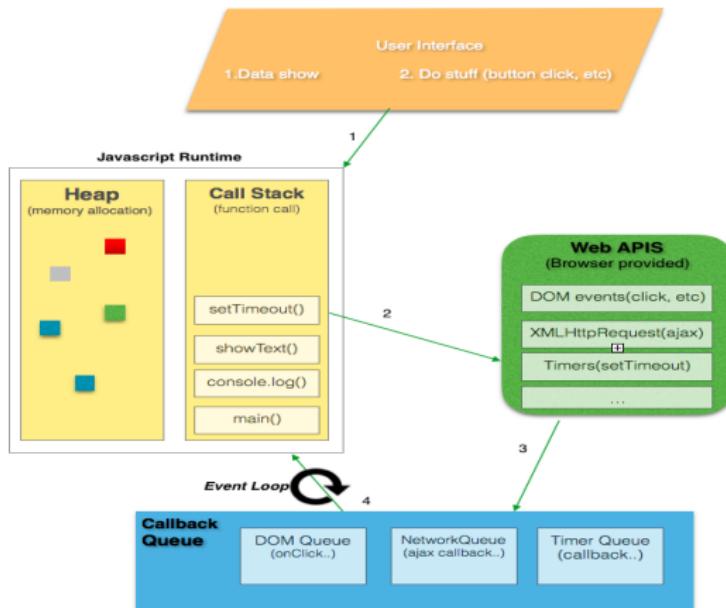
Un message est ajouté à la file lorsqu'un évènement survient et qu'un gestionnaire y a été associé.



©G. Pandvia

La boucle d'évènements JS

La premier message n'est traité et l'est alors intégralement que lorsque la pile est vide.



©G. Pandvia

Exemple avec le minuteur setTimeout ()

WindowOrWorkerGlobalScope.setTimeout () appelée avec 2 arguments

- ➊ Le message = la callback à exécuter quand il sera traité.
- ➋ Le délai minimum à attendre avant que le message ne soit placé dans la file.

async-settimeout.js

```
1 const s = new Date().getSeconds();
2 setTimeout(function() {
3     console.log("Exécuté après " + (new Date().getSeconds() - s) + " secondes.");
4 }, 5000);
5 while (true) {
6     if (new Date().getSeconds() - s >= 2) {
7         console.log("2 secondes se sont écoulées");
8         break;
9     }
10 }
11 // Console avec horodatage :
12 // 10:27:20,999 2 secondes se sont écoulées
13 // 10:27:24,423 Exécuté après 5 secondes.
```

Exemple avec le répétiteur setInterval()

async-setinterval.js

```
1 let intervalId; // variable storing the interval id
2 function flashText() {
3     const oElem = document.querySelector("#my_box");
4     oElem.className = oElem.className === "go" ? "stop" : "go";
5 }
6 document.querySelector("#start").addEventListener("click", function() {
7     // check if an interval has already been set up
8     intervalId ??= setInterval(flashText, 1000);
9 });
10 document.querySelector("#stop").addEventListener("click", function() {
11     clearInterval(intervalId);
12     intervalId = null;
13 });
```

Les promesses

Fonctions de rappels et promesses

Le support de l'asynchronisme en JS

- Historique : les fonctions de rappels (*callbacks*) asynchrones
- Depuis ES2015 (ECMAScript 2015) : les promesses.

De nombreuses API utilisent les promesses

Par exemple, `fetch` (vs. `XMLHttpRequest`).

Les callbacks ne s'exécutent pas toutes de manière asynchrone

Dépend du contexte d'exécution : par exemple, synchrone dans `forEach`.

Fonction de rappel asynchrone

Une fonction `f` passée en paramètre à une fonction `g` laquelle

- S'exécutera en tâche de fond sans bloquer le fil principal.
- Placera `f` dans la file de messages une fois terminée.
- En lui communiquant éventuellement des informations (événement, données, état, erreur . . .)

Exemples : gestionnaires d'évènements DOM avec callback.

async-callback.html

```
1 <!DOCTYPE html>
2 <html>
3 <head><meta charset="UTF-8"><title>MDN : simple callback</title></head>
4 <body>
5   <button id="btn">Click!</button>
6   <script>
7     document.querySelector("#btn").addEventListener('click', () => {
8       document.body.appendChild(document.createElement('p'));
9       document.body.lastElementChild.textContent = 'This is a newly-added
paragraph.';
10    })
11  </script>
12 </body>
13 </html>
```

Les promesses

Nouveau style de programmation asynchrone en JS

Une promesse est un objet JS qui représente le succès ou l'échec d'une opération asynchrone.

Une promesse

- Est un objet `Promise` renvoyé immédiatement mais qui se "résoud" plus tard de manière asynchrone.
- Sa résolution est soit positive, soit négative.

Une promesse est

- Initialement en attente d'être résolue (`pending`).
- Puis résolue (`settled`) : soit tenue (`fulfilled`), soit rompue (`rejected`).

Les promesses

Construction avec `new Promise((resolve, reject)=>{...})`

- Crée 2 propriétés internes désignant l'état courant et le résultat futur de la promesse.
- Exécute la fonction passée en argument (executor) qui elle-même lance l'opération asynchrone (le corps de l'executor).
- `resolve` et `reject` sont 2 fonctions internes (que l'on nomme comme on veut) : l'opération asynchrone appellera l'une ou l'autre.

L'opération asynchrone résoudra la promesse

- Positivement en appelant `resolve`.
- Négativement en appelant `reject`.

La valeur communiquée à `resolve/reject` devient le résultat de la promesse

Les promesses

Le résultat `r` d'une promesse résolue peut être communiqué à des fonctions de rappel anonymes (executor)

En invoquant la méthode `then(succes => {...}, échec => {...})` qui appellera

- la première callback si la promesse est tenue avec `succes == r`
- la seconde sinon avec `échec == r`

`then(s=>{...})` avec une seule callback ne gérera que le succès

La méthode `catch(r => {...})` peut alors être chainée pour gérer le cas où la promesse est rompue.

Chaque bloc `then(...)` ou `catch(...)` peut renvoyer une nouvelle promesse

On peut donc enchaîner les promesses avec une suite mêlant blocs `then(...)` et `catch(...)`

Exemple de promesse avec then

L'invocation de la méthode `then(s=>{ . . . })` avec une seule callback permet de gérer la tenue d'une promesse.

async-promise-then.js

```
1 let myFirstPromise = new Promise((réussir, échouer) => {
2   // réussir(...) est appelée quand l'opération asynchrone a réussi.
3   // échouer(...) est appelée quand l'opération asynchrone a échoué.
4   // Dans cet exemple, on simule l'opération asynchrone avec le minuteur setTimeout(...).
5   // Dans la pratique, ce pourrait être du requêtage HTTP avec XHR ou des écouteurs avec l'API DOM.
6   setTimeout( function() {
7     réussir("Succès !") // Succès, Tout s'est bien passé !
8   }, 2000);
9 })
10
11 myFirstPromise.then((messageRéussite) => {
12   // 'messageRéussite' correspond à l'argument qui a été communiqué à réussir(...) par l'opération async.
13   // Ce peut être un scalaire (chaîne de caractères, etc.) ou un objet.
14   console.log("OK ! " + messageRéussite)
15 });
```

Exemple de promesse avec then/catch

L'invocation de la méthode `catch(s=>{ . . . })` avec une seule callback permet de gérer la rupture d'une promesse.

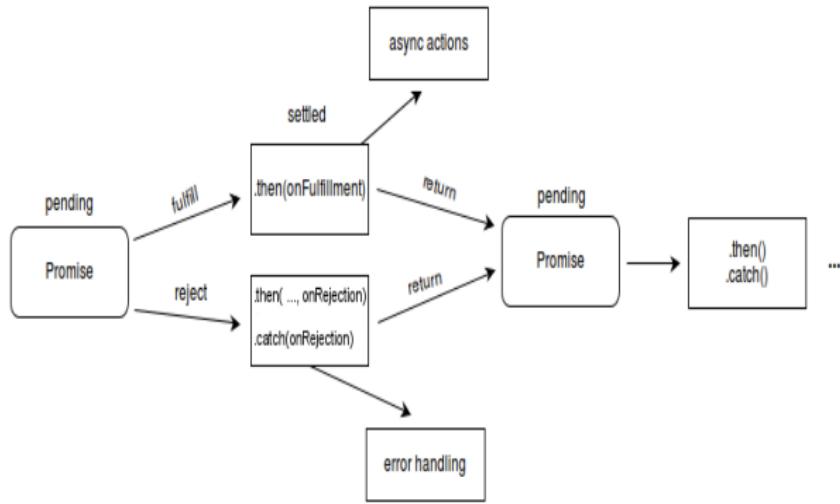
async-promise-then-catch.js

```
1 const oddTime = (date) => {
2   return new Promise((resolve, reject) => {
3     parseInt(date.getTime() / 1000) % 2
4       ? resolve('le nombre de secondes est impair :-)')
5       : reject('le nombre de secondes n\'est pas impair :-(');
6   });
7 }
8 const now = new Date();
9 oddTime(now)
10 .then(msg => console.log(msg), msg => console.error(msg));
11
12 oddTime(new Date(now.getTime() + 1000))
13 .then(msg => console.log(msg))
14 .catch(msg => console.error(msg))
```

Enchaînement de promesses

Chaque bloc `then(...)` ou `catch(...)` peut renvoyer une nouvelle promesse

On peut donc enchaîner les promesses avec une suite mêlant blocs `then(...)` et `catch(...)`



Exemple de chaînage de promesses

Ecriture directe

La valeur renvoyée par une promesse tenue est le paramètre passé à la fonction (executor) du bloc `then()` suivant.

async-promise-chain.js

```
1 fetch('personne.json')
2     .then(response => response.json())
3     .then(myJson => {
4         let div = document.querySelector('#getJSON');
5         div.textContent = myJson.nom + " - " + myJson.surnoms[1];
6     })
7     .catch(e => {
8         console.log('Problem with your fetch operation: ' + e.message);
9     });

```

Promesses vs. callbacks traditionnelles

Différences

- Une promesse ne peut réussir ou échouer qu'une fois.
- On peut ajouter une callback à une promesse même après qu'elle ait réussi ou échoué.
- On peut exécuter plusieurs opérations asynchrones en séquence et dans un ordre précis grâce au chaînage des promesses.

async-promise-vs-callback.js

```
1 let myFirstPromise = new Promise((resolve, reject) => {
2   resolve("Success!");
3 })
4 Array.from(Array(10000000).keys()).forEach(() => myDate = new Date());
5 myFirstPromise.then((successMessage) => {
6   console.log("Yay! " + successMessage)
7 });
```

Exemple de promesses en boucle

async-promise-repeat.js

```
1 const p = () => {
2     return new Promise((resolve, reject) => {
3         Math.floor(Math.random() * 2) == 1 ? resolve('zero') : reject('un');
4     });
5 }
6 function f() {
7     p()
8         .then(function(msg) {
9             console.log(msg);
10            f();
11        })
12        .catch(msg => console.log(msg));
13 }
14
15 f();
```

Tenir plusieurs promesses avec Promise.all (1)

Promise.all() prend un tableau de promesses et en renvoie une

- Qui est tenue quand, et si, toutes les promesses le sont.
- Qui est rompue quand, et si, l'une d'elles l'est.

async-promise-all-part1.js

```
1 // Define function to fetch a file and return it in a usable form
2 function fetchAndDecode(url, type) {
3     // Returning the top level promise, so the result of the entire chain is returned out of the function
4     return fetch(url).then(response => {
5         // Depending on what type of file is being fetched, use the relevant function to decode its contents
6         if(type === 'blob') {
7             return response.blob();
8         } else if(type === 'text') {
9             return response.text();
10        }
11    })
12    .catch(e => {
13        console.log('Problem fetching resource "${url}": ' + e.message);
14    });
15 }
16 // Call the fetchAndDecode() method to fetch the images and the text, and store their promises in variables
17 let coffee = fetchAndDecode('coffee.jpg', 'blob');
18 let tea = fetchAndDecode('tea.jpg', 'blob');
19 let description = fetchAndDecode('description.txt', 'text');
```

Tenir plusieurs promesses avec Promise.all (2)

async-promise-all-part2.js

```
1 // Use Promise.all() to run code only when all three function calls have resolved
2 Promise.all([coffee, tea, description]).then(values => {
3   console.log(values);
4   // Store each value returned from the promises in separate variables; create object URLs from the blobs
5   let objectURL1 = URL.createObjectURL(values[0]);
6   let objectURL2 = URL.createObjectURL(values[1]);
7   let descText = values[2];
8
9   // Display the images in <img> elements
10  let image1 = document.createElement('img');
11  let image2 = document.createElement('img');
12  image1.src = objectURL1;
13  image2.src = objectURL2;
14  document.body.appendChild(image1);
15  document.body.appendChild(image2);
16
17  // Display the text in a paragraph
18  let para = document.createElement('p');
19  para.textContent = descText;
20  document.body.appendChild(para);
21});
```

async et await (ES2017)

async et await

Syntaxe légère pour les promesses ajoutée dans ES2017

async

- Se place avant une déclaration de fonction.
- Rend la fonction asynchrone.
- La fonction renvoie alors une promesse à consommer.
- On peut lui ajouter un bloc `.then() { . . . }` pour la résoudre.

async-async.js

```
1 async function hello1() { return "Hello1" };
2 hello1();
3 let hello2 = async function() { return "Hello2" };
4 hello2();
5 let hello3 = async () => { return "Hello3" };
6 hello3();
7 hello1().then((value) => console.log(value));
8 hello2().then((value) => console.log(value));
9 hello3().then((value) => console.log(value));
```

async et await

await

- S'utilise au sein d'une promesse ou fonction asynchrone (précédée de `async`).
- Se place avant un appel à une fonction asynchrone avec promesse.
- Bloque l'exécution du code qui suit la promesse jusqu'à sa résolution.

async-await.js

```
1 function resolveAfter2Seconds(x) {  
2     return new Promise((resolve) => {  
3         setTimeout(() => {  
4             resolve(x);  
5         }, 2000);  
6     });  
7 }  
8  
9 async function f1() {  
10    const x = await resolveAfter2Seconds(10); // résolution bloquante  
11    console.log("A"); // affiche "A" au bout de 2 secondes  
12    console.log(x); // affiche "10" au bout de 2 secondes  
13 }
```

Promesses vs. async/await

async-await-async-promise.js

```
1 fetch('coffee.jpg')
2 .then(response => response.blob())
3 .then(myBlob => {
4   let objectURL = URL.createObjectURL(myBlob);
5   let image = document.createElement('img');
6   image.src = objectURL;
7   document.body.appendChild(image);
8 })
9 .catch(e => {
10   console.log('Problem with your fetch operation: ' + e.message);
11 });
```

async-await-async.js

```
1 async function myFetch() {
2   let response = await fetch('coffee.jpg');
3   let myBlob = await response.blob();
4   let objectURL = URL.createObjectURL(myBlob);
5   let image = document.createElement('img');
6   image.src = objectURL;
7   document.body.appendChild(image);
8 }
9 myFetch().catch(e => {
10   console.log('Problem with your fetch operation: ' + e.message);
11 });
```

CM JS : L'API Fetch

L'API Fetch

Interface d'accès en JS au protocole HTTP

- Permet de récupérer des données par requêtes asynchrones.
- Utilisable depuis une page web ou dans un service worker.
- Alternative plus puissante que l'API XMLHttpRequest
 - Basée sur les promesses.
 - Fournit une interface aux requêtes, réponses, en-tête et corps des messages HTTP.
- Méthode globale `fetch` à disposition.

fetch()

- Tente de récupérer une ressource web (p. ex. par son URL).
- Permet de paramétriser la requête (en-têtes, corps, etc.).
- Renvoie une promesse qui n'est rompue qu'en cas d'erreur réseau ou absence de réponse, pas en cas d'erreur HTTP (p. ex. 404).
- Et qui se résoud par un objet `Response` modélisant la réponse HTTP
 - `Response.status` fournit le code de la réponse HTTP : Informatif [100–199], Succès [200–299], Redirection [300–399], Erreur client [400–499], Erreur serveur [500–599].
 - `Response.ok` indique si la réponse est fructueuse [200–299].
 - Fournit différentes méthodes renvoyant chacune une promesse permettant d'extraire le corps de la réponse selon son format :
 - `arrayBuffer()` : tableau d'octets.
 - `blob()` : blob (p. ex. images).
 - `formData()` : encodage clé-valeur au format

Arguments de fetch

async-fetch-api.js

```
1 let promise = fetch(url, {  
2     /* indique l'option par défaut  
3     method: "GET", // *GET, POST, PUT, DELETE, etc.  
4     headers: {  
5         "Content-Type": "text/plain; charset=UTF-8" //pour un corps de type chaine  
6     },  
7     body: undefined, // string, FormData, Blob, BufferSource, ou URLSearchParams  
8     // doit correspondre à Content-Type pour requête POST  
9     referrer: "about:client",  
10    // "" (pas de référent) ou une url de l'origine  
11    referrerPolicy: "no-referrer-when-downgrade",  
12    // no-referrer, *no-referrer-when-downgrade, origin,  
13    // origin-when-cross-origin, same-origin, strict-origin,  
14    // strict-origin-when-cross-origin, unsafe-url  
15    mode: "cors", // no-cors, *cors, same-origin  
16    credentials: "same-origin", // include, *same-origin, omit  
17    cache: "default", // *default, no-cache, reload, force-cache, only-if-cached  
18    redirect: "follow", // manual, *follow, error  
19    integrity: "", // ou hash tel "sha256-abcdef1234567890"  
20    keepalive: false, // ou true pour que la requête survive à la page  
21    signal: undefined // ou AbortController pour annuler la requête  
22});
```

fetch : Récupérer du texte brut par méthode GET

async-xhr-fetch-get-text.php

```
1 header("Content-Type: text/plain; charset=UTF-8");
2 $txt = "Le Lorem Ipsum est simplement du faux texte employé ...";
3 echo $txt;
```

async-fetch-get-text.js

```
1 let ft = function() {
2     let elt = document.querySelector('#gettext');
3     let myRequest = new Request('async-xhr-fetch-get-text.php');
4     fetch(myRequest)
5         .then(function(response) {
6             if (!response.ok) {
7                 throw new Error("HTTP error, status = " + response.status);
8             }
9             return response.text();
10        })
11        .then(function(myText) {
12            elt.textContent = myText;
13        })
14        .catch(function(error) {
15            let p = document.createElement('p');
16            p.appendChild(document.createTextNode('Error: ' + error.message));
17            document.body.insertBefore(p, elt);
18        });
19 }();
```

fetch : Récupérer du JSON par méthode GET (1)

async-xhr-fetch-get-json.php

```
1 header("Content-Type: application/json; charset=UTF-8");
2 $tab = [
3     'nom' => 'Clémenceau',
4     'surnoms' => [
5         'Le tigre',
6         'Le père La Victoire',
7         'Le tombeur de gouvernements',
8         'Le premier flic de France'
9     ]
10 ];
11 echo json_encode($tab);
```

fetch : Récupérer du JSON par méthode GET (2)

async-fetch-get-json.js

```
1 let fJ = function() {
2     let elt = document.querySelector('#getJSON');
3     let myRequest = new Request('async-xhr-fetch-get-json.php');
4     fetch(myRequest)
5         .then(function(response) {
6             if (!response.ok) {
7                 throw new Error("HTTP error, status = " + response.status);
8             }
9             return response.json();
10        })
11        .then(function(myJson) {
12            elt.textContent = myJson.nom + " - " + myJson.surnoms[1];
13        })
14        .catch(function(error) {
15            let p = document.createElement('p');
16            p.appendChild(document.createTextNode('Error: ' + error.message));
17            document.body.insertBefore(p, elt);
18        });
19    }();
```

fetch : Récupérer du XML par méthode GET

async-xhr-fetch-get-xml.php

```
1 header("Content-Type: application/xml; charset=UTF-8");
2 echo file_get_contents("bibliotheque.xml");
```

async-fetch-get-xml.js

```
1 let fx = function() {
2     let elt = document.querySelector('#getxml');
3     let myRequest = new Request('async-xhr-fetch-get-xml.php');
4     fetch(myRequest)
5         .then(function(response) {
6             if (!response.ok) {
7                 throw new Error("HTTP error, status = " + response.status);
8             }
9             return response.text();
10        })
11        .then(function(myXml) {
12            xml = (new window.DOMParser()).parseFromString(myXml, "text/xml");
13            let eltxml = xml.querySelector("titre");
14            elt.textContent = eltxml.textContent;
15        })
16        .catch(function(error) {
17            let p = document.createElement('p');
18            p.appendChild(document.createTextNode('Error: ' + error.message));
19            document.body.insertBefore(p, elt);
20        });
21 }();
```

fetch : Récupérer des données binaires par méthode GET

async-xhr-fetch-get-blob-1.php

```
1 // ouvre un fichier en mode binaire
2 $name = 'tea.jpg';
3 $fp = fopen($name, 'rb');
4 // envoie les en-têtes
5 header("Content-Type: image/jpg");
6 header("Content-Length: " . filesize($name));
7 // envoie le contenu du fichier, puis stoppe le script
8 fpassthru($fp);
9 exit;
```

async-fetch-get-blob.js

```
1 let fb = function() {
2     let elt = document.querySelector('#getimg > img');
3     let myRequest = new Request('flowers.jpg');
4     fetch(myRequest)
5         .then(function(response) {
6             if (!response.ok) {
7                 throw new Error("HTTP error, status = " + response.status);
8             }
9             return response.blob();
10        })
11        .then(function(myBlob) {
12            let objectURL = URL.createObjectURL(myBlob);
13            elt.src = objectURL;
14        })
15        .catch(function(error) {
16            let p = document.createElement('p');
```

fetch : Envoyer des données par méthode GET

async-xhr-fetch-get.php

```
1 header("Content-Type: application/json; charset=UTF-8");
2 $tab = [
3     'surname' => $_GET["surname"],
4     'name' => $_GET["name"],
5     'age' => 10
6 ];
7 echo json_encode($tab);
```

async-fetch-get.js

```
1 let fg = function() {
2     let elt = document.querySelector("#xget");
3     let url = new URL(window.location.href.replace(/[^\\/]*/$,
4         "async-xhr-fetch-get.php"));
5     const params = new URLSearchParams();
6     params.set('surname', 'foo');
7     params.set('name', 'bar');
8     url.search = params.toString();
9     fetch(url, {
10         method: 'GET'
11     })
12     .then((response) => response.json())
13     .then((myJson) => {
14         console.log('Success:', myJson);
15         elt.textContent = myJson.surname + " - " + myJson.name + " - " + myJson.age;
16     })
17     .catch((error) => {
18         console.error('Error:', error);
19     });
20 }();
```

fetch : Envoyer des données par méthode POST (1)

async-xhr-fetch-post.php

```
1 header( "Content-Type: application/json; charset=UTF-8" );
2 $tab = [
3     'surname' => $_POST[ "surname" ],
4     'name'      => $_POST[ "name" ],
5     'age'       => 10
6 ];
7 echo json_encode( $tab );
```

async-fetch-post-urlsearchparam.js

```
1 let fps = function () {
2     let elt = document.querySelector("#xposturlsearchparam");
3     let usp = new URLSearchParams();
4     usp.append("surname", "foo");
5     usp.append("name", "bar");
6     fetch('async-xhr-fetch-post.php', {
7         method: 'POST', //or 'PUT'
8         headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
9         body: usp
10    })
11    .then((response) => response.json())
12    .then((myJson) => {
13        console.log('Success:', myJson);
14        elt.textContent = myJson.surname + " - " + myJson.name + " - " + myJson.age;
15    })
16    .catch((error) => {
17        console.error('Error:', error);
18    });
19 }();
```

fetch : Envoyer des données par méthode POST (2)

async-xhr-fetch-post.php

```
1 header( "Content-Type: application/json; charset=UTF-8" );
2 $tab = [
3     'surname' => $_POST[ "surname" ],
4     'name'      => $_POST[ "name" ],
5     'age'       => 10
6 ];
7 echo json_encode( $tab );
```

async-fetch-post-formdata.js

```
1 let fpf = function () {
2     let elt = document.querySelector("#xpostformdata");
3     let form = new FormData();
4     form.append("surname", "foo");
5     form.append("name", "bar");
6     fetch('async-xhr-fetch-post.php', {
7         method: 'POST', //or 'PUT'
8         // let the browser automatically set the header based on the data to send
9         // headers: { 'Content-Type': 'multipart/form-data' },
10        body: form,
11    })
12        .then((response) => response.json())
13        .then((myJson) => {
14            console.log('Success:', myJson);
15            elt.textContent = myJson.surname + " - " + myJson.name + " - " + myJson.age;
16        })
17        .catch((error) => {
18            console.error('Error:', error);
19        });
}
```

fetch : Envoyer des données par méthode POST (3)

async-xhr-fetch-post-json.php

```
1 header( "Content-Type: application/json; charset=UTF-8" );
2 $jsonData      = file_get_contents( 'php://input' );
3 $tab           = json_decode( $jsonData, true );
4 $tab[ 'age' ]  = 12;
5 echo json_encode( $tab );
```

async-fetch-post-json.js

```
1 let fpj = function () {
2     let elt = document.querySelector("#xpostjson");
3     fetch('async-xhr-fetch-post-json.php', {
4         method: 'POST', //or 'PUT'
5         headers: { "Content-Type": "application/json;charset=UTF-8" },
6         body: JSON.stringify({ surname: "foo", name: "bar" }) //ne pas encadrer les clés !
7     })
8     .then((response) => response.json())
9     .then((myJson) => {
10         console.log('Success:', myJson);
11         elt.textContent = myJson.surname + " - " + myJson.name + " - " + myJson.age;
12     })
13     .catch((error) => {
14         console.error('Error:', error);
15     });
16 }();
```

CM JS : Web workers

Web Workers et iframes d'origine multiple

Un Web Worker se construit avec `Worker(f)`

- Exécute le fichier JS `f` passé en argument.
- S'exécute sur un fil différent du fil principal.
- A sa propre pile, tas, et file de messages.
- Ne peut pas accéder au DOM et à l'objet `window`.
- Communique par messages avec le fil principal ou autres workers avec `postMessage()`.
- Peut utiliser XHR et websockets (canal bidirectionnel avec un serveur).

Autres types de workers

- SharedWorker : worker partagés entre scripts s'exécutant dans différentes fenêtres ou iframes.
- ServiceWorker : “proxy” entre navigateur, applis web et réseau (cache, notifications push ...).

Exemple de Web Worker

Récupération des données d'un message via gestionnaire
onmessage (propriété de worker) et event.data

worker-simple-sync.js

```
1 const btn = document.querySelector('button');
2 const worker = new Worker('worker-date.js');
3 btn.addEventListener('click', () => {
4     worker.postMessage('Go!');
5     let pElem = document.createElement('p');
6     pElem.textContent = 'This is a newly-added paragraph.';
7     document.body.appendChild(pElem);
8 });
9 worker.onmessage = function(e) {
10     console.log(e.data);
11 }
```

worker-date.js

```
1 onmessage = function(e) {
2     console.log(e.data); // Go !
3     let myDate;
4     Array.from(Array(10000000).keys()).forEach(() => myDate = new Date());
5     postMessage(myDate);
6 }
```

Exemple de Web Worker : suite de Fibonacci (1)

Interception d'erreur via gestionnaire onerror (propriété de worker)

worker-fibonacci.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="UTF-8" />
6     <title>MDN : Test threads fibonacci</title>
7 </head>
8
9 <body>
10    <div id="result"></div>
11    <script>
12        var worker = new Worker('worker-fibonacci.js');
13        worker.onmessage = function(event) {
14            document.getElementById('result').textContent = event.data;
15            console.log('Got: ' + event.data + '\n');
16        };
17        worker.onerror = function(error) {
18            console.error('Worker error: ' + error.message + '\n');
19            throw error;
20        };
21        worker.postMessage('5');
22    </script>
23 </body>
24
```

Exemple de Web Worker : suite de Fibonacci (2)

worker-fibonacci.js

```
1 let results = [];
2
3 function resultReceiver(event) {
4     results.push(parseInt(event.data));
5     if (results.length == 2) {
6         postMessage(results[0] + results[1]);
7     }
8 }
9
10 function errorReceiver(event) {
11     throw event.data;
12 }
13
14 onmessage = function(event) {
15     let n = parseInt(event.data);
16     if (n == 0 || n == 1) {
17         postMessage(n);
18         return;
19     }
20     for (let i = 1; i <= 2; i++) {
21         let worker = new Worker('worker-fibonacci.js');
22         worker.onmessage = resultReceiver;
23         worker.onerror = errorReceiver;
24         worker.postMessage(n - i);
25     }
26 };
```

CM JS : Les modules

Les modules

Motivations

Diviser le code en modules exportables et importables

- Fonctionnalité historiquement supportée dans Node.js et bibliothèques/frameworks JS (RequireJS, Webpack ...)
- Supportée dorénavant dans les navigateurs

Améliorer les performances de chargement

Extension **.js** ou **.mjs** pour les fichiers correspondant aux modules

Les fichiers **.mjs** doivent être servis avec

Content-Type: text/javascript; charset=utf-8

Exemple avec un canevas HTML

Arborescence de fichiers avec 2 modules

index.html

main.js

modules/

canvas.js : fonctions pour gérer le canevas (`create()`,
`createReportList()`)

square.js : constante (`name`) et fonctions de dessin/écriture
(`draw()`, `reportArea()`, `reportPerimeter()`)

Voir code source

Export des fonctionnalités d'un module

Avec l'instruction `export` devant chaque élément que l'on souhaite exporter ou en fin de fichier

square.js

```
1 export const name = "square";
2 export function draw(ctx, length, x, y, color) {
3     ctx.fillStyle = color;
4     ctx.fillRect(x, y, length, length);
5     return {
6         length: length,
7         x: x,
8         y: y,
9         color: color,
10    };
11 }
12 function reportArea(...) {...}
13 function reportPerimeter(...) {...}
14 //OU
15 // export { name, draw };
```

Import des fonctionnalités d'un module

Avec l'instruction `import ... from ...`

- En listant les fonctionnalités que l'on souhaite importer par leurs identifiants
- Suivi du chemin du fichier du module relativement à la racine du site

main.js

```
1 import { create, createReportList } from './modules/canvas.js';
2 import { name, draw, reportArea } from "./modules/square.js";
3 let myCanvas = create("myCanvas", document.body, 480, 320);
4 let reportList = createReportList(myCanvas.id);
5 let square1 = draw(myCanvas.ctx, 50, 50, 100, "blue");
6 reportArea(square1.length, reportList);
7 ...
```

Les variables importées sont en lecture seule à l'instar des variables
`const`

Chargement de module en HTML

Avec `<script type="module" src="main.js"></script>`

Différences entre modules et scripts classiques

- Les modules utilisent le mode strict
- Ils sont chargés en mode différé : `defer` n'est pas nécessaire
- Ils sont exécutés une seul fois même si référencés par plusieurs `<script>`
- Les fonctionnalités importées ne sont disponibles que dans la portée du script (pas accessibles en console notamment)

Renommage

Renommage de fonctionnalités importées ou exportées avec `as`
Pour éviter les collisions de noms

`js-modules-renaming.js`

```
1 import {
2     name as squareName,
3     draw as drawSquare,
4 } from "./modules/square.js";
5 import {
6     name as circleName,
7     draw as drawCircle,
8 } from "./modules/circle.js";
9 ...
```

Objet module

Import des fonctionnalités dans un objet

import * as Module from ... crée un objet Module dont les champs sont les fonctionnalités importées

- Pour éviter les conflits de noms en important différents modules dans différents objets agissant chacun comme un espace de noms

js-modules-object-module.js

```
1 import * as Canvas from "./modules/canvas.js";
2 import * as Square from "./modules/square.js";
3 let myCanvas = Canvas.create("myCanvas", document.body, 480, 320);
4 let square1 = Square.draw(myCanvas.ctx, 50, 50, 100, "blue");
5 ...
```

Export/import de classes

js-modules-class-export.js

```
1 class Square {  
2     constructor(ctx, listId, length, x, y, color) { ... }  
3     draw() { ... }  
4     ...  
5 }  
6 export { Square };
```

js-modules-class-import.js

```
1 import { Square } from "./modules/square.js";  
2 ...  
3 let square1 = new Square(myCanvas.ctx, myCanvas.listId, 50, 50,  
100, "blue");  
4 square1.draw();  
5 ...
```

Aggrégation de modules

On peut combiner différents modules en un seul, et ce récursivement

main.js > shapes.js > square.js, circle.js

js-modules-aggregation-square.js

```
1 class Square {...}  
2 export { Square };
```

js-modules-aggregation-shapes.js

```
1 export { Square } from './shapes/square.js';  
2 export { Circle } from './shapes/circle.js';
```

js-modules-aggregation-main.js

```
1 import { Square, Circle } from './modules/shapes.js';
```

Chargement dynamique de modules

Chargement à la demande avec promesse et fonction `import()`

```
import (".../unmodule.js").then( (module) => { ..  
module utilisable }
```

`js-modules-import-promise-main.js`

```
1 let squareBtn = document.querySelector(".square");  
2 squareBtn.addEventListener("click", () => {  
3     import("./modules/square.js").then((Module) => {  
4         let square1 = new Module.Square(...);  
5         square1.draw();  
6     });  
7 });
```

Chargement dynamique de modules

Export de module asynchrone avec await

- Les fonctionnalités importées seront utilisables une fois l'export résolu
- Import non bloquant pour les autres

js-modules-export-await-module.js

```
1 const colors = fetch("../data/colors.json").then((response) =>
response.json());
2 // grâce à await, les imports devront attendre que le fichier JSON ait été téléchargé
3 // avant de pouvoir utiliser 'colors'
4 export default await colors;
```

js-modules-export-await-main.js

```
1 // non bloquant : les autres modules sont téléchargés en parallèle
2 import colors from './modules/getColors.js';
3 import { Canvas } from './modules/canvas.js';
4 // pas d'exécution avant que tous les modules aient été chargés
5 let circleBtn = document.querySelector('.circle');
6 let square1 = new Module.Square(myCanvas.ctx, myCanvas.listId, 50,
50, 100, colors.blue);
```

CM JS : Sécurité : CORS et CSP

Origine de page web

L'origine d'une page web (origin)

- Définie par le schéma (protocol), l'hôte (domain) et le port de l'URL utilisée.
- Certaines opérations requièrent une même origine (same-origin policy).
- Relaxable avec CORS.

Origine identique (same origin)

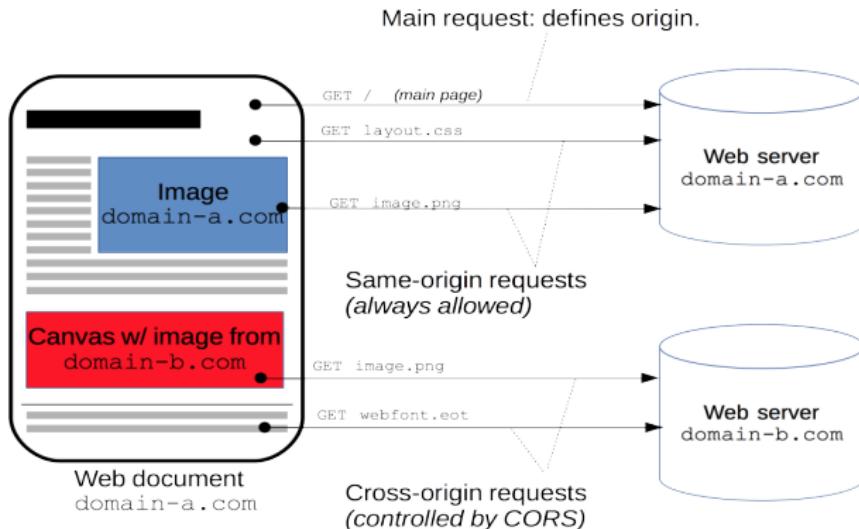
http://example.com/app1/index.html **et** http://example.com
http://example.com/app1/index.html **et**
http://example.com:80

Origines différentes

http://example.com/app1 **et** https://example.com/app2
http://www.example.com **et** http://example.com:8080

Cross-Origin Resource Sharing (CORS)

Mécanisme HTTP permettant à une page web d'accéder à des ressources d'origine différente



©MDN

Fonctionnement de CORS

Ajout d'en-têtes HTTP

- Permettant au serveur d'indiquer les origines autorisées.
- Imposant au navigateur de négocier avec le serveur (preflight) en cas de requêtes à effet de bord (GET, POST).

Exemples de requêtes cross-origin

- Invocations de XHR ou fetch.
- Téléchargement de polices en CSS :
`@font-face { src:
url(...); }`
- Téléchargement de textures avec WebGL.
- Téléchargement d'images/vidéos dans un canevas.
- CORS est intégré à XHR/Fetch : pas d'en-têtes à programmer.
- Les erreurs CORS ne sont pas relayées à JS.

CORS : exemple

Code JS servi par `https://foo.example`

`security-cors-simple-request.js`

```
1 const xhr = new XMLHttpRequest();
2 const url = 'https://bar.other/resources/public-data/';
3 xhr.open('GET', url);
4 xhr.onreadystatechange = someHandler;
5 xhr.send();
```

Ressource accessible depuis n'importe quel domaine

`https://bar.other` renvoie `Access-Control-Allow-Origin: *`

Ressource accessible uniquement depuis `https://foo.example`

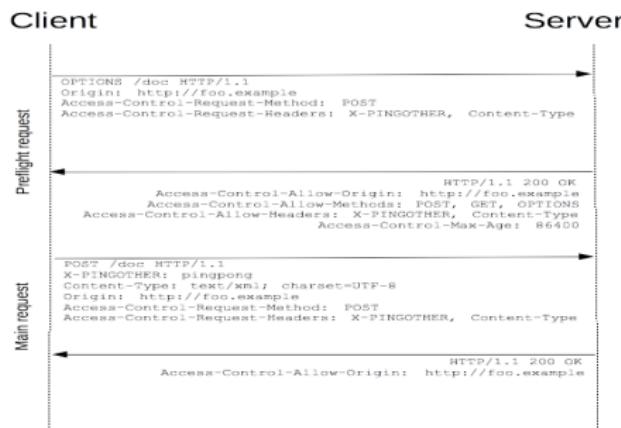
`https://bar.other` renvoie `Access-Control-Allow-Origin:`

`https://foo.example`

CORS : preflight request

security-cors-preflight-request.js

```
1 const xhr = new XMLHttpRequest();
2 xhr.open('POST', 'https://bar.other/resources/post-here/');
3 xhr.setRequestHeader('X-PINGOTHER','pingpong');
4 xhr.setRequestHeader('Content-Type','application/xml');
5 xhr.onreadystatechange = handler;
6 xhr.send('<person>Arun</person>');
```



©MDN

Referrer

- Adresse de la page web visitée précédente sur laquelle un lien a été suivi.
- Utilisé par les serveurs pour la connexion, le cache, l'analyse de navigation, etc

Content-Security Policy (CSP)

Mécanisme HTTP pour se protéger d'attaques XSS ou d'injection de données

- Un navigateur implémentant CSP peut fonctionner avec un serveur sans support CSP et inversement.
- Les navigateurs utilisent par défaut la *same-origin policy*.

Principes

- Le serveur communique au navigateur les domaines de confiance (whitelisting) selon le type de ressources : scripts, polices, images, ...
- Le navigateur n'exécute/ne charge que les scripts/fichiers des domaines autorisés.

Par défaut, CSP n'autorise pas scripts et styles en ligne (`<script>`, `<style>`, `onclick` ...) et la fonction JS `eval()`.

Politiques CSP

2 alternatives pour “activer” CSP dans un serveur

Le serveur communique sa politique *policy*

- Par ajout aux réponses HTTP de l'en-tête

Content-Security-Policy: *policy*

- Par insertion dans les pages HTML servies de

```
<meta http-equiv="Content-Security-Policy"  
content="policy">
```

Politique CSP : une suite de directives restreignant chacune l'accès à un certain type de ressources

default-src 'self'

default-src 'self' *.trusted.com

default-src 'self'; img-src *; media-src medial.com
media2.com; script-src userscripts.example.com

default-src https://onlinebanking.jumbobank.com

CSP : exemple (1)

security-csp.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Tests CSP (fichier importé en PHP)</title>
7   <script defer src="security-csp.js"></script>
8 </head>
9
10 <body>
11   <button id="btn" onclick="alert('clic - inline HTML')>Click!</button>
12   <script>
13     document.querySelector("#btn").addEventListener('click', () => {
14       alert('Clic - inline JS!');
15       let pElem = document.createElement('p');
16       pElem.textContent = 'This is a newly-added paragraph.';
17       document.body.appendChild(pElem);
18     })
19   </script>
20   
21   
22 </body>
23
24 </html>
```

CSP : exemple (2)

security-csp.php

```
1 // pas de blocage : renvoie les erreurs à security-csp-parser.php
2 header("Content-Security-Policy-Report-Only: default-src 'self'; report-uri
security-csp-parser.php;");
3 $htmlfile = file_get_contents("security-csp.html");
4 $html = <<<HTML
5 $htmlfile
6 HTML;
7 echo $html;
```

security-csp-strict.php

```
1 $imgdom = "https://upload.wikimedia.org";
2 // ne bloquera pas l'exécution de security-csp.js
3 // ni de le chargement de l'image de upload.wikimedia.org
4 header("Content-Security-Policy: default-src 'self'; img-src $imgdom");
5 $htmlfile = file_get_contents("security-csp.html");
6 $html = <<<HTML
7 $htmlfile
8 HTML;
9 echo $html;
```

OAuth2

Délégation d'autorisation

Pour accorder un accès limité sur une ressource à une application tierce.

Exemples : utiliser l'API d'un site pour le compte d'un utilisateur (Google APIs ...).

Côté serveur

Quelques outils de développement et de tests

- PHP : Guzzle (HTTP Client), Curl
- Clients lourds : SoapUI, JMeter, Postman . . .
- Shell : curl + jq

The screenshot shows a browser-based API testing interface. The URL is `https://datanova.legroupe.laposte.fr/api/records/1.0/search/?dataset=laposte_hexasmal&q=THOUARCE`. The request method is GET. The response is displayed in JSON format, showing the following structure:

```

1  {
2     "nhits": 1,
3     "parameters": {
4         "dataset": "laposte_hexasmal",
5         "timezone": "UTC",
6         "q": "THOUARCE",
7         "rows": 10,
8         "format": "json"
9     },
10    "records": [
11        {
12            "datasetid": "laposte_hexasmal",
13            "recordid": "hbfb44dfafab976e9e4155a561850df026d3fb61",
14            "fields": {
15                "code_postal": "49380",
16                "code_commun": "49245",
17                "libelle_d_acheminement": "BELLEVIGNE EN LAYON",
18                "libell_d_acheinement": "BELLEVIGNE EN LAYON",
19                "ligne_5": "THOUARCE",
20                "nom_de_la_commune": "BELLEVIGNE EN LAYON",
21                "od": "47.2592682331,",
22                "geometry": {
23                    "type": "Point",
24                    "coordinates": [
25                        -0.517923916545,
26                        47.2592682331
27                    ]
28                },
29                "record_timestamp": "2020-02-24T23:35:00+00:00"
30            }
31        }
32    ]
33 }
34
  
```

Below the JSON response, there are tabs for Headers (27), Attachments (0), SSL Info (2 certs), Representations (5), Schema, and JMS (0). The status bar at the bottom indicates a response time of 1262ms (642 bytes).

CM JS : Les API

Les APIs du Web

API = Interface de Programmation Applicative

Ensemble normalisé de classes, méthodes ou fonctions qui sert de façade par laquelle un logiciel abstrait et offre des services à d'autres logiciels.

Web APIs disponibles en JS côté client

API de navigateur

- Intégrées au navigateur Web, eg. API de géolocalisation.

API de parties tierces

- Accessibles depuis un site Web, eg. API Twitter.

Relations entre JS, APIs et autres artefacts JS

JavaScript

- Langage haut niveau intégré aux navigateurs.
- Egalement disponible dans d'autres environnements (par ex. Node.js).

API du navigateur

- Intégrées dans le navigateur au dessus de JS (par ex. API DOM).

API tierces

- Intégrées à des plateformes tierces pour utiliser leurs fonctionnalités dans vos propres pages Web (par ex. Twitter).

Bibliothèques JS

- Fichier(s) JS contenant des fonctions utiles à l'écriture de fonctionnalités courantes (par ex. React).

Modèles/cadriels JS

- Paquets HTML/CSS/JS/... à installer pour écrire une application Web entière (par ex. Angular). Contrairement aux bibliothèques, les modèles appellent le code développé (**Inversion de Contrôle**).

APIs de navigateur les plus couramment utilisées

Manipulation HTML et CSS

- DOM, CSSOM (pendant du DOM pour le CSS).

Récupération de données du serveur (AJAX)

- XMLHttpRequest, Fetch.

Dessin et manipulation de graphiques

- Canvas, WebGL.

Audio et vidéo

- HTMLMediaElement, Web Audio, Web RTC.

Périphériques

- Géolocalisation, Notifications, Vibrations, ...

Stockage de données côté client

- Web Storage (paires clé-valeur), IndexedDB (données structurées et indexées dont fichiers/blobs).

Autres

- Intl (internationalisation), WebWorkers (calcul parallèle)

Fonctionnement des APIs

Caractéristiques communes

- Fondées sur des objets.
- Points d'entrée identifiables.
- Utilisent des évènements pour réagir aux changements d'état.
- Peuvent imposer des mécanismes de sécurité supplémentaires.

Les APIs sont fondées sur des objets

Interagissent avec le code en utilisant un/des objets(s) JS qui servent de conteneurs pour :

- Les données utilisées par l'API : contenues dans des propriétés d'objet.
- Les fonctionnalités mises à disposition : contenues dans des méthodes d'objets.

Exemple : objets de l'API Géolocalisation

- `Geolocation` : contient 3 méthodes de récupération des données géographiques.
- `Position` : contient un objet `Coordinates` représentant la position de l'appareil horodatée.
- `Coordinates` : contient latitude, longitude, vitesse, direction du mouvement, ...

Points d'entrée des APIs

Cas simples

- Propriété `navigator.geolocation` qui renvoie l'objet `Geolocation` pour l'API de géolocalisation.
- Objet `Document` pour l'API DOM.

Cas complexes

- Appel de `getContext()` sur une référence à l'élément `<canvas>` (`HTMLCanvasElement`) sur lequel on veut dessiner.

Les APIs utilisent des évènements

Exemple avec XMLHttpRequest

Objet représentant une requête HTTP au serveur pour récupérer une ressource (texte, JSON, XML, ...).

- Offre différents gestionnaires d'évènements : eg. `onload` en cas de réponse récupérée avec succès.

apis-xhr.js

```
1 var request = new XMLHttpRequest();
2 request.addEventListener("load", function() {
3   console.log(JSON.stringify(request.response));
4 });
5 request.open('GET', "apis-xhr.php");
6 request.responseType = 'json';
7 request.send();
```

apis-xhr.php

```
1 class A
2 {
3   public $name;
4   function __construct ($n)
5   {
6     $this->name = $n;
7   }
8 }
9 $a = new A ("Cesar");
10 echo json_encode ($a);
```

APIs et règles de sécurité

Les API sont sujettes aux mêmes règles que JS

Exemple : la règle **same-origin policy**.

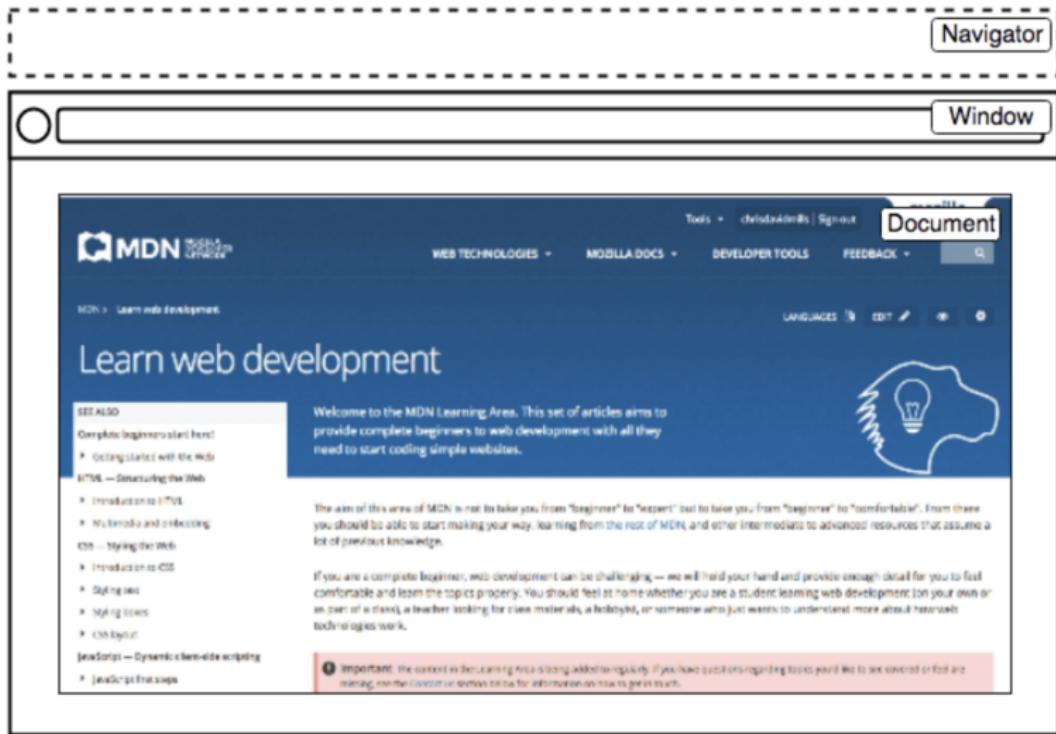
Les API peuvent imposer des règles supplémentaires

- Usage imposé du protocole HTTPS.
- Autorisation d'activation : géolocalisation, notifications, etc.
- Etc.

CM JS : L'API DOM

L'API DOM

Les parties importantes d'un navigateur



Trois types d'objets JS fondamentaux (alias interfaces)

navigator

- L'état et l'identité du navigateur (alias user-agent).
- Pour récupérer des données de géolocalisation, le langage préféré de l'utilisateur, le flux média de la webcam, ...

window

- La fenêtre (ou onglet) dans lequel la page est chargée.
- Pour redimensionner la fenêtre, lui associer des gestionnaires d'évènements, stocker des données spécifiques à la page ...

document

- Le document chargé.
- Pour obtenir une référence d'élément HTML, en changer le contenu texte, en modifier le style, pour créer/ajouter/supprimer des éléments.

L'objet window

Objet global représentant la fenêtre du navigateur

Il n'est pas nécessaire de le spécifier, sauf lorsqu'on manipule des (i)frames.

dom-window.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>L'objet window</title>
6 </head>
7 <body>
8   <script>
9     // 2 instructions équivalentes
10    alert("Hello world!");
11    window.alert("Hello world!");
12  </script>
13 </body>
14 </html>
```

L'objet window

Les fonctions globales ne sont pas des méthodes de window
isNaN(), parseInt(), parseFloat() ...

Une variable déclarée sans var est une propriété de window.

dom-window-1.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>L'objet window</title>
4 </head>
5 <body>
6   <script>
7     let texte = 'globale';
8     (function () { // IIFE (Immediately-Invoked Function Expression)
9       let texte = 'locale';
10      glob = "autre globale"; //déconseillé !
11      alert(window.texte); // 'globale'
12      alert(texte); // 'locale'
13    }) ();
14    alert(texte); // 'globale'
15    alert(glob); // 'autre globale'
16  </script>
17 </body></html>
```

Le DOM

API pour manipuler documents XML et HTML

- Offre une représentation structurée et orientée objet des éléments, de leurs attributs et de leur contenu.
- Permet l'ajout et la suppression d'objets.
- Permet la modification des propriétés de ces objets à l'aide de méthodes.
- Permet de répondre aux évènements utilisateur.

Mise en oeuvre

- API standardisée autour de JS dans tous les navigateurs.
- API dépendante du langage de scripts côté serveur pour la manipulation de documents XML
 - eg. les classes PHP `DOMDocument`, `DOMElement` etc.

Les normes du DOM

Recommandations du W3C

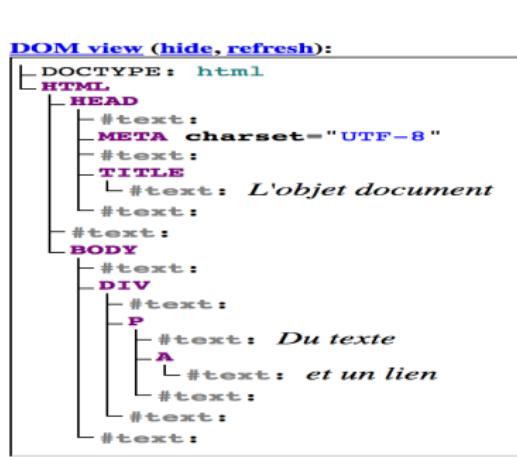
- 1998 : DOM Level 1 (Core + HTML).
- 2000 : DOM Level 2 (`getElementById`, modèle d'évènements, espaces de noms XML, CSS).
- 2004 : DOM Level 3 (support XPath, sérialisation en XML).
- 2015 : DOM Level 4 du **WHATWG**.
- 2015-... : DOM *Living Standard* du WHATWG.

Des bibliothèques permettent d'assurer la compatibilité du code JS pour différents navigateurs (eg. **jQuery**)

L'arbre DOM d'un document HTML

Arborescence de noeuds de différents types représentant

- Les éléments HTML.
- Leur contenu : noeud(s) texte et/ou noeud(s) élément.



dom-document.html

```
1 <!DOCTYPE html>
2 <html><head>
3 <meta charset="utf-8" />
4 <title>L'objet document</title>
5 </head>
6 <body>
7   <div>
8     <p>
9       Du texte
10      <a>et un lien</a>
11    </p>
12  </div>
13 </body></html>
```

Les attributs d'éléments sont des noeuds non-connectés.

Types de noeuds DOM

La propriété `nodeType` renvoie le type du noeud.

Constante	Valeur	Description
<code>Node.ELEMENT_NODE</code>	1	Un noeud <code>Element</code> tel que <code><p></code> ou <code><div></code> .
<code>Node.TEXT_NODE</code>	3	Le <code>Text</code> actuel de l' <code>Element</code> ou <code>Attr</code> .
<code>Node.PROCESSING_INSTRUCTION_NODE</code>	7	Une <code>ProcessingInstruction</code> d'un document XML tel que la déclaration <code><?xml-stylesheet ... ?></code> .
<code>Node.COMMENT_NODE</code>	8	Un noeud <code>Comment</code> .
<code>Node.DOCUMENT_NODE</code>	9	Un noeud <code>Document</code> .
<code>Node.DOCUMENT_TYPE_NODE</code>	10	Un noeud <code>DocumentType</code> c'est-à-dire <code><!DOCTYPE html></code> pour des documents HTML5.
<code>Node.DOCUMENT_FRAGMENT_NODE</code>	11	Un noeud <code>DocumentFragment</code> .

Depuis le DOM-4, suppression des constantes

2 (ATTRIBUTE), 4 (CDATA_SECTION), 5 (ENTITY_REFERENCE), 6 (ENTITY) et 12 (NOTATION).

Noms de noeuds DOM

La propriété `nodeName` renvoie le nom du noeud.

Interface	<code>nodeName</code>
<code>Attr</code>	Identique à <code>Attr.name</code>
<code>CDATASection</code>	"#cdata-section"
<code>Comment</code>	"#comment"
<code>Document</code>	"#document"
<code>DocumentFragment</code>	"#document-fragment"
<code>DocumentType</code>	Identique à <code>DocumentType.name</code>
<code>Element</code>	Identique à <code>Element.tagName</code>
<code>Entity</code>	nom de l'entité
<code>EntityReference</code>	nom de la référence d'entité
<code>Notation</code>	nom de la notation
<code>ProcessingInstruction</code>	Identique à <code>ProcessingInstruction.target</code>
<code>Text</code>	"#text"

Types et noms de noeuds DOM

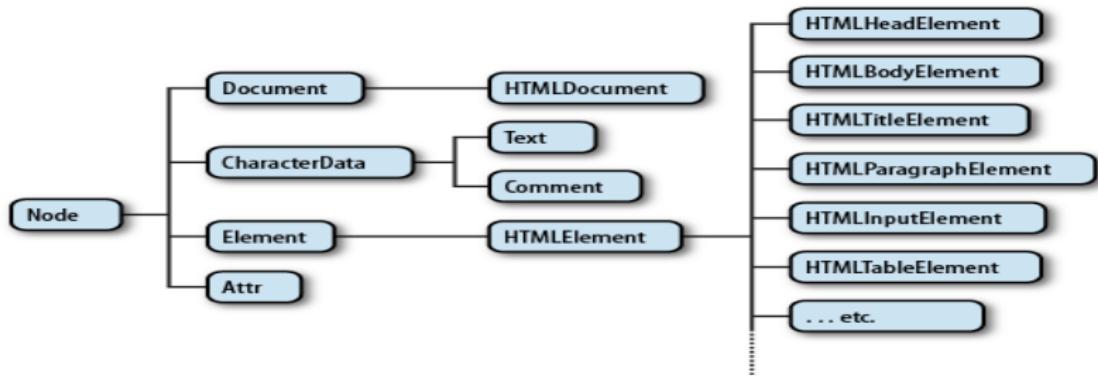
dom-nodetype.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8" />
5  <title>Les propriétés nodeType/nnodeName</title>
6  </head>
7  <body>
8    <div id="myDiv">Du texte.</div>
9    <script>
10      let myDiv = document.getElementById("myDiv");
11      alert(myDiv.nodeType); //1
12      alert(myDiv.nodeName); //DIV
13      let myDivId = myDiv.getAttributeNode("id");
14      alert(myDivId.nodeType); //2
15      alert(myDivId.nodeName); //ID
16      let myDivText = myDiv.firstChild;
17      alert(myDivText.nodeType); //3
18      alert(myDivText.nodeName); //#text
19    </script>
20  </body>
21  </html>
```

Interfaces DOM

JS fournit une hiérarchie d'**interfaces**

Propriétés et méthodes auxquelles se conforment les différents noeuds d'un DOM selon leur type.



Chaque objet modélisant un noeud du DOM implémente les propriétés et méthodes de sa chaîne d'interfaces.

Héritage d'interfaces : exemple



dom-interfaces.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8"/>
3 <title>Interfaces DOM</title>
4 </head>
5 <body>
6   <table id="myTable">
7     <tr><td>A1</td><td>A2</td></tr>
8     <tr><td>B1</td><td>B2</td></tr>
9   </table>
10  <script>
11    let table = document.querySelector('#myTable');
12    // HTMLTableElement interface : attribut rows
13    table.rows[0].style.backgroundColor = "red";
14    // HTMLElement interface : attribut title
15    table.title = "Une aide";
16    // Element interface : attribut class
17    table.className = "mesTables";
18    // Node interface : attribut baseURI
19    alert('Base URL de la table :' + table.baseURI);
20  </script>
21 </body></html>
```

Les types d'objets importants

document (**interface Document**)

- L'objet correspondant au noeud racine du DOM.

element (**interface Element**).

- Un objet correspondant à un noeud du DOM.

nodeList (**interface NodeList**).

- Une collection d'**elements** statique ou dynamique.
- Traversable avec `for`, `for...of` et méthode `forEach`.
- Convertible en tableau avec `Array.from()`.

attribute (**interface Attr**).

- Un objet correspondant à un attribut d'**element**.

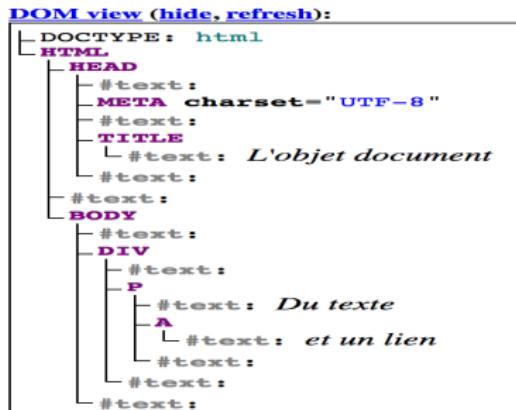
namedNodeMap (**interface NamedNodeMap**).

- Une collection dynamique d'**attributes**.
- Traversable avec `for`.
- Convertible en tableau avec `Array.from()`.

L'objet document

Est un objet propriété de window

- Représente le noeud racine du DOM.
- Parent de l'élément HTML.
- Consulter l'outil DOM de Firefox ou [Live DOM Viewer](#).



dom-document.html

```
1 <!DOCTYPE html>
2 <html><head>
3 <meta charset="utf-8" />
4 <title>L'objet document</title>
5 </head>
6 <body>
7   <div>
8     <p>
9       Du texte
10      <a>et un lien</a>
11    </p>
12  </div>
13 </body></html>
```

Accès aux éléments HTML

Avec les méthodes d'objets Document ou Element :

`getElementById(identifiant)`

- Renvoie l'élément d'attribut `id` égal à `identifiant`.

`getElementsByName(balise)`

- Renvoie le tableau (`HTMLCollection`) des éléments de balise `balise`.

`getElementsByName(nom)`

- Renvoie le tableau (`HTMLCollection`) des éléments de formulaires (`HTML 5`) d'attribut `name` égal à `nom`.

`querySelector(sélecteur)`

- Renvoie le 1er élément correspondant au sélecteur CSS.

`querySelectorAll(sélecteur)`

- Renvoie le tableau d'éléments (`HTMLCollection`) correspondant au sélecteur CSS.

Accès aux éléments HTML

dom-getelements.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Accès aux éléments HTML</title>
6 </head>
7 <body>
8   <div id="myDiv">
9     <p>
10       Du texte <a>et un lien</a>
11     </p>
12   </div>
13   <div>
14     <p>Que du texte
15   </div>
16   <script>
17     let div = document.getElementById("myDiv");
18     console.debug(div);
19     let divs = document.getElementsByTagName("div");
20     let i=1;
21     Array.from(divs).forEach(function(div) {
22       alert("Element n\u00b0"+(i++)+" : "+div.innerHTML);
23     });
24     for(let div of divs) {
25       console.debug("Element n\u00b0"+" : "+div.textContent);
26     }
27   </script>
28 </body>
29 </html>
```

Accès aux éléments HTML avec sélecteurs CSS3

Séquence	Signification
<code>*</code>	tout élément
<code>E</code>	tout élément de type E
<code>E[foo]</code>	tout élément E portant l'attribut "foo"
<code>E[foo="bar"]</code>	tout élément E portant l'attribut "foo" et dont la valeur de cet attribut est exactement "bar"
<code>E[foo~="bar"]</code>	tout élément E dont l'attribut "foo" contient une liste de valeurs séparées par des espaces, l'une de ces valeurs étant exactement égale à "bar"
<code>E[foo^="bar"]</code>	tout élément E dont la valeur de l'attribut "foo" commence exactement par la chaîne "bar"
<code>E[foo\$="bar"]</code>	tout élément E dont la valeur de l'attribut "foo" finit exactement par la chaîne "bar"
<code>E[foo*="bar"]</code>	tout élément E dont la valeur de l'attribut "foo" contient la sous-chaîne "bar"
<code>E[lang = "en"]</code>	tout élément E dont l'attribut 'lang' est une liste de valeurs séparées par des tirets et commençant (à gauche) par "en"
<code>E:root</code>	un élément E, racine du document
<code>E:nth-child(n)</code>	un élément E qui est le n-ième enfant de son parent
<code>E:nth-last-child(n)</code>	un élément E qui est le n-ième enfant de son parent en comptant depuis le dernier enfant
<code>E:nth-of-type(n)</code>	un élément E qui est le n-ième enfant de son parent et de ce type
<code>E:nth-last-of-type(n)</code>	un élément E qui est le n-ième enfant de son parent et de ce type en comptant depuis le dernier enfant
<code>E:first-child</code>	un élément E, premier enfant de son parent
<code>E:last-child</code>	un élément E, dernier enfant de son parent
<code>E:first-of-type</code>	un élément E, premier enfant de son type
<code>E:last-of-type</code>	un élément E, dernier enfant de son type
<code>E:only-child</code>	un élément E, seul enfant de son parent
<code>E:only-of-type</code>	un élément E, seul enfant de son type

Accès aux éléments HTML avec sélecteurs CSS3

E:empty	un élément E qui n'a aucun enfant (y compris noeuds textuels purs)
E:link E:visited	un élément E qui est la source d'un hyperlien dont la cible n'a pas encore été visitée (:link) ou a déjà été visitée (:visited)
E:active E:hover E:focus	un élément E pendant certaines actions de l'usager
E:target	un élément E qui est la cible de l'URL d'origine contenant lui-même un fragment identifiant.
E:lang(c)	un élément E dont le langage (humain) est c (le langage du document spécifie comment le langage humain est déterminé)
E:enabled E:disabled	un élément d'interface utilisateur E qui est actif ou inactif.
E:checked E:indeterminate	un élément d'interface utilisateur E qui est coché ou dont l'état est indéterminé (par exemple un bouton-radio ou une case à cocher)
E:contains("foo")	un élément E dont le contenu textuel concaténé contient la sous-chaîne "foo"
E::first-line	la première ligne formatée d'un élément E
E::first-letter	le premier caractère formaté d'un élément E
E::selection	la partie d'un élément E qui est actuellement sélectionnée/mise en exergue par l'usager
E::before	le contenu généré avant un élément E
E::after	le contenu généré après un élément E
E.warning	<i>Uniquement en HTML.</i> Identique à E[class~="warning"].
E#myid	un élément E dont l'ID est égal à "myid".
E:not(s)	un élément E qui n'est pas représenté par le sélecteur simple s
E F	un élément F qui est le descendant d'un élément E
E > F	un élément F qui est le fils d'un élément E
E + F	un élément F immédiatement précédé par un élément E
E ~ F	un élément F précédé par un élément E

Accès aux éléments HTML avec sélecteur CSS

dom-selecteur.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Accès aux éléments HTML avec sélecteurs CSS</title>
4 </head>
5 <body>
6   <div id="menu">
7     <div class="item">
8       <span>Elément 1</span>
9       <span>Elément 2</span>
10    </div>
11    <div class="publicité">
12      <span>Elément 3</span>
13      <span>Elément 4</span>
14    </div>
15  </div>
16  <script>
17    let query = document.querySelector("#menu .item span");
18    let queryAll = document.querySelectorAll("#menu .item span");
19    alert(query.innerHTML); // Elément 1
20    alert(queryAll.length); // 2
21    // Elément 1 - Elément 2
22    alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML);
23  </script>
24 </body></html>
```

Accès aux attributs d'éléments HTML

Avec les méthodes d'objets Element :

getAttribute(att)

- Renvoie la valeur (string) de l'attribut de nom att.

setAttribute(att, val)

- Fixe à val la valeur de l'attribut de nom att.

dom-attributs-acces.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Accès aux attributs d'éléments HTML</title>
4 </head>
5 <body>
6   <a id="myLink" href="http://www.anywhere.com">
7     Un lien modifié dynamiquement.
8   </a>
9   <script>
10    let myLink = document.querySelector("#myLink");
11    let href = myLink.getAttribute("href");
12    alert(href);
13    myLink.setAttribute("href", "http://es6-features.org");
14  </script>
15 </body></html>
```

Accès direct aux attributs d'éléments HTML

Par une propriété d'objets `Element` de même nom que l'attribut

dom-attributs-acces-direct-1.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Accès direct aux attributs d'éléments HTML</title>
6 </head>
7 <body>
8   <a id="myLink" href="">un lien</a>
9   <script>
10     let myLink = document.querySelector("#myLink");
11     let href = myLink.href;
12     alert(href);
13     myLink.href = "http://es6-features.org";
14   </script>
15 </body>
16 </html>
```

Accès direct aux attributs d'éléments HTML

CamelCase utilisée pour les propriétés correspondant aux

- Attributs à nom composé (eg. `readonly`) ou comportant un tiret (eg. `background-color`).
- Attributs dont le nom est réservé en JS : `class` (`className` et `classList`), `for` (`htmlFor`).

dom-attributs-acces-direct-2.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Accès direct aux attributs d'éléments HTML</title>
4 <style>
5 .bleu { background:lightblue;} .rouge { color:red;} .noir { color:black;}
6 </style>
7 </head>
8 <body>
9   <a id="myLink" class="rouge" href="">Un lien</a>
10  <script>
11    let myLink = document.querySelector("#myLink");
12    alert("Prêt ?");
13    myLink.style.textTransform = "uppercase";
14    myLink.classList.add("bleu");
15    myLink.classList.remove("rouge");
16    myLink.classList.toggle("noir");
17  </script>
18 </body></html>
```

Récupérer le code HTML

Sous forme de chaîne avec propriété `Element.innerHTML`

dom-innerhtml.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Accès au code HTML</title>
6 </head>
7 <body>
8   <div id="myDiv">
9     <p>
10       Du texte <a href="#myDiv">et un lien</a>
11     </p>
12   </div>
13   <p>Autre texte</p>
14 <script>
15   let div = document.querySelector("#myDiv");
16   alert(div.innerHTML);
17 </script>
18 </body>
19 </html>
```

Ajouter ou éditer du code HTML

Avec Element.innerHTML

dom-innerhtml-1.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Edition/ajout de code HTML</title>
6 </head>
7 <body>
8   <div id="myDiv">
9     <p>
10       Du texte <a>et un lien</a>
11     </p>
12   </div>
13   <script>
14     let div = document.querySelector("#myDiv");
15     div.innerHTML = "<h1>Un titre écrase le paragraphe</h1>";
16     div.innerHTML += "<h1>Et un second titre en ajout</h1>";
17   </script>
18 </body>
19 </html>
```

Ajouter ou éditer le contenu textuel d'éléments

Avec Element.textContent

dom-textcontent.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Edition/ajout de code HTML</title>
6 </head>
7 <body>
8   <div id="myDiv">
9     <p>
10       Du texte <a>et un lien</a>
11     </p>
12   </div>
13   <script>
14     let div = document.querySelector("#myDiv");
15     div.textContent = "ABC";
16     div.innerHTML += "<h1>Et un second titre en ajout</h1>";
17   </script>
18 </body>
19 </html>
```

Naviguer dans le DOM : parent

La propriété `Node.parentNode` permet d'accéder au parent d'un noeud

dom-parent.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>La propriété parentNode</title>
6 </head>
7 <body>
8   <blockquote>
9     <p id="myP">Un paragraphe.</p>
10    </blockquote>
11    <script>
12      let bq = document.querySelector("#myP").parentNode;
13      alert(bq); // object HTMLQuoteElement
14    </script>
15 </body>
16 </html>
```

Naviguer dans le DOM : enfants

Element .firstChild et lastChild donnent accès aux premier et dernier enfants d'un noeud

dom-firstchild.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8"/>
3 <title>Création d'éléments</title>
4 <link rel="stylesheet" href="dom.css"></head>
5 <body>
6   <div>
7     Voici <p id="myP">du texte <a>et un lien</a></p>
8   </div>
9   <script>
10    let div = document.querySelector("div");
11    alert(div.firstChild.nodeName); // #text
12    alert(div.firstElementChild.nodeName); // P !!
13  </script>
14 </body></html>
```

firstElementChild et lastElementChild donnent accès aux premier et dernier enfants qui sont des éléments HTML

Naviguer dans le DOM : contenu texte

Node.nodeValue et CharacterData.data donnent le contenu texte d'un noeud textuel

dom-nodevalue.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Les propriétés nodeValue/data</title>
4 </head>
5 <body>
6   <div>
7     <p id="myP">Du texte <a>un lien</a> et <strong>en emphase</strong></p>
8   </div>
9   <script>
10  let myP = document.querySelector("#myP");
11  alert(myP.firstChild.nodeValue); // Du texte
12  alert(myP.lastChild.firstChild.data); // en emphase
13  alert(myP.nodeValue); // null
14  </script>
15 </body></html>
```

Naviguer dans le DOM : tableau des enfants

Node.childNodes renvoie le tableau des enfants d'un noeud

Node.children renvoie le tableau des éléments enfants d'un noeud

dom-childnodes.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Les propriétés childNodes et children</title>
6 </head>
7 <body>
8     <div>
9         <p id="myP">Du texte <a>un lien</a></p>
10    </div>
11    <script>
12        let myP = document.querySelector("#myP");
13        for (child of myP.childNodes) {
14            if (child.nodeType === Node.ELEMENT_NODE) {
15                alert("Noeud element : " + child.firstChild.data); // un lien
16            } else {
17                alert("Noeud texte : " + child.data); // du texte
18            }
19        }
20        console.log(myP.children); // HTMLCollection { 0: a, length: 1 }
21    </script>
22 </body>
```

Naviguer dans le DOM : adelphes

Node.nextSibling et previousSibling donnent accès au noeud suivant et précédent

dom-nextsibling.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Les propriétés nextSibling/previousSibling</title>
4 </head>
5 <body>
6   <div>
7     <p id="myP">Du texte <a>un lien</a></p>
8   </div>
9   <script>
10    let child = document.querySelector("#myP").lastChild;
11    while (child) {
12      if (child.nodeType === 1) {
13        alert(child.firstChild.data); //élément HTML
14      } else {
15        alert(child.data); //noeud texte
16      }
17      child = child.previousSibling;
18    }
19  </script>
20 </body></html>
```

nextElementSibling et previousElementSibling donnent accès aux éléments HTML suivant et précédent

Naviguer dans le DOM

Attention aux noeuds texte “vides” (espaces, CR ...) !

dom-noeuds-texte-vides.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Les noeuds texte vides : retours à la ligne, espaces ...</title>
4 </head>
5 <body>
6   <div>
7     <p id="myP1"><a>Une ancre</a></p>
8     <p id="myP2">
9       <a>Une ancre</a>
10      </p>
11    </div>
12    <script>
13      let child1 = document.querySelectorAll("p")[0].firstChild;
14      let child2 = document.querySelectorAll("p")[1].firstChild;
15      if (child1.nodeType !== child2.nodeType) {
16        alert(child2.nodeType); // 3 (Node.TEXT_NODE)
17      }
18    </script>
19 </body></html>
```

Créer et insérer des éléments

En 3 temps

- ① Création d'un élément avec la méthode
`Document.createElement(tag)`.
- ② Affectation des attributs avec la méthode
`Element.setAttribute(att, val)`.
- ③ Insertion de l'élément dans le document avec l'une des méthodes :
 - `Node.appendChild(tag)`
 - `Node.insertBefore(node, referenceNode)`

Création d'un noeud texte avec la méthode

`Document.createTextNode(contenuTexte)`.

Créer et insérer des éléments

dom-createelement.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Création et ajout d'éléments</title>
4 </head>
5 <body>
6   <div>
7     <p id="myP">Du texte </p>
8   </div>
9   <script>
10    let myP = document.querySelector("#myP");
11    let newLink = document.createElement("a");
12    alert(newLink.isConnected); //false
13    newLink.id = "myLink";
14    newLink.href = "http://www.lemonde.fr";
15    newLink.title = "Le Monde";
16    newLinkText = document.createTextNode("Le site du Monde");
17    newLink.appendChild(newLinkText);
18    myP.appendChild(newLink);
19    alert(newLink.isConnected); //true
20  </script>
21 </body></html>
```

Cloner des éléments

Avec `Node.cloneNode(flag)`

- Si `flag=true`, clone aussi enfants et attributs du noeud.
- Si `flag=false`, ne clone ni enfants ni attributs.

Les gestionnaires d'évènement enregistrés ne sont pas clonés.

dom-clonenode.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Clonage de noeuds</title>
4 </head>
5 <body>
6   <div>
7     <p id="myP">Du texte </p>
8   </div>
9   <script>
10  let myP = document.querySelector("#myP");
11  let p1 = myP.cloneNode(true);
12  let p2 = myP.cloneNode(false);
13  document.querySelectorAll("body")[0].appendChild(p1);
14  document.querySelectorAll("body")[0].appendChild(p2);
15  </script>
16 </body></html>
```

Remplacer des éléments

Avec `Node.replaceChild(newChild, oldChild)`

Remplace l'enfant `oldChild` par `newChild`.

dom-replacechild.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Remplacement de noeud enfant</title>
4 </head>
5 <body>
6   <div><p>Du texte</p></div>
7   <script>
8     let myDiv = document.querySelector("div");
9     let myS = document.createElement("span");
10    myS.innerHTML = "Un &lt;span&gt; en remplacement d'un &lt;p&gt;";
11    myDiv.replaceChild(myS,myDiv.firstChild);
12  </script>
13 </body></html>
```

Supprimer des éléments

Avec `Node.removeChild()` sur noeud parent

dom-removechild.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Suppression de noeud enfant</title>
4 </head>
5 <body>
6   <div><p>Du texte</p></div>
7   <script>
8     let myP = document.querySelector("div > p");
9     myP.parentNode.removeChild(myP);
10    /* alternative
11    let myDiv = document.querySelector("div");
12    myDiv.removeChild(myP);
13    */
14  </script>
15 </body></html>
```

Insérer des éléments

Avec `Node.insertBefore()` sur noeud adelphe

dom-insertbefore.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Insertion d'éléments</title>
4 </head>
5 <body>
6   <div><p>Du texte</p></div>
7   <script>
8     let myDiv = document.querySelector("div");
9     let newP = document.createElement("p");
10    newPText = document.createTextNode("Voici ");
11    newP.appendChild(newPText);
12    if(myDiv.lastChild);
13      myDiv.insertBefore(newP, myDiv.lastChild);
14    </script>
15 </body></html>
```

Manipuler le CSS

Cascading Style Sheets (CSS)

Langage de feuilles de style

Pour décrire la présentation d'un document écrit dans un langage de balisage (HTML, XHTML, SVG, XML).

Objectifs

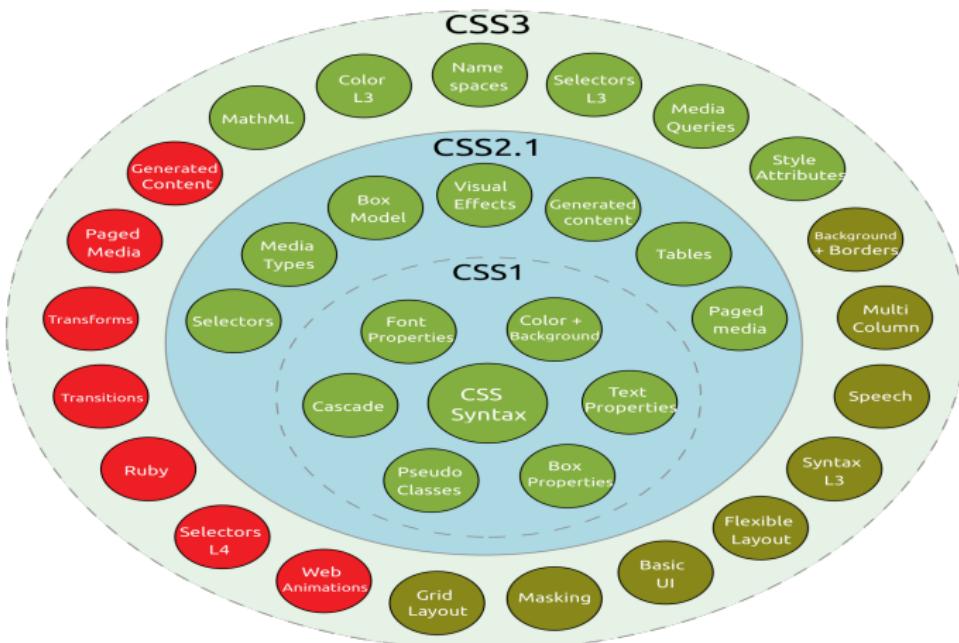
- Séparer la forme du contenu : mise en page, couleurs, polices ...
- Apporter plus de flexibilité et de contrôle sur la présentation.
- Permettre la réutilisation de fichiers CSS de mise en forme dans différentes pages HTML.
- Réduire la complexité des pages HTML (eg. répétition).
- Adapter la présentation aux capacités du terminal et à différents rendus (écran, impression, ...).

Historique

Les versions CSS

- CSS 1 : 1996
 - Polices, couleurs, espacements, alignements, marges . . .
 - Statut : Obsolète.
- CSS 2 : 1998-2011
 - Positionnement, types média . . .
 - Statut : Recommandation.
- CSS 3 : 1999-...
 - Décomposition en ~50 modules.
 - Statut : Normalisation en cours de chaque module.

De CC 1 à CSS 3



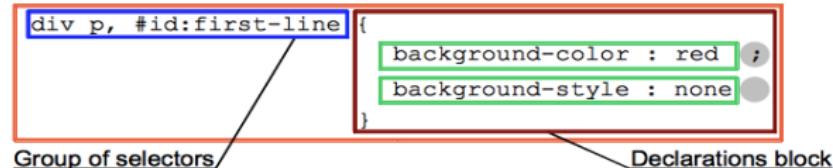
By Krauss - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=44954967>

Déclarations, blocs et règles CSS

Déclaration = paire propriété:valeur

- Propriétés et valeurs sont insensibles à la casse.
- Toute déclaration invalide est ignorée (eg. font-size:red).

A CSS ruleset (or rule):



Bloc = liste entre {} de déclarations séparées par des ;

Règle = sélecteur(s) précédant un bloc de déclarations

- Le moteur de rendu CSS appliquera les déclarations du bloc aux éléments satisfaisant le groupe de sélecteurs.
- Toute règle comportant un sélecteur invalide est ignorée.

Instruction CSS

Une règle ou une @-règle (@-rule)

Différents types de @-règles :

- Méta-données relatives au fichier, eg. @charset, @import.
- Règles conditionnelles, eg. @media, @document.
- Règles descriptives, eg. @font-face.

Chaque @-règle a sa propre syntaxe et sémantique.

Editer le CSS d'éléments

Avec

- L'attribut `style` d'un élément HTML donné.
- Des règles placées dans un élément `style` de l'en-tête du document HTML.
- Des règles placées dans un fichier CSS importé dans le document HTML.

Editer le CSS d'éléments

css-feuille.css

```
1 @charset "UTF-8";
2
3 div {
4     margin: auto;
5     border: solid 1px green;
6 }
```

css-application.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Editer le CSS d'éléments HTML</title>
7     <link rel="stylesheet" href="css-feuille.css">
8 </head>
9 <style>
10    div {
11        background-color: orange;
12    }
13 </style>
14 </head>
15
16 <body>
17     <div style="text-align: center;">Je suis un DIV.</div>
18 </body>
19
20 </html>
```

Lecture de propriété CSS en JS

Avec `window.getComputedStyle(element)` qui renvoie en lecture seule un objet contenant toutes les propriétés CSS de l'élément après application des règles (importées ou non)

css-getcomputedstyle.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Accès à propriété CSS avec getComputedStyle</title>
7   <link rel="stylesheet" href="css-feuille.css">
8   <style>
9     div {
10       color: orange;
11     }
12   </style>
13 </head>
14
15 <body>
16   <div style="font-style:italic">green border, orange, italic</div>
17   <script>
18     let div = document.querySelector('div');
19     alert(getComputedStyle(div).border); //0.55px solid rgb(0, 128, 0)
20     alert(getComputedStyle(div).color); //rgb(255, 165, 0)
21     alert(getComputedStyle(div).fontStyle); //italic
22   </script>
23 </body>
24
```

Ecriture de propriété CSS en JS

Avec `element.style.propriété`

- CamelCase pour propriété CSS comportant un tiret.
- Ne pas utiliser pour lire la propriété CSS calculée car n'intègre pas les règles importées !

Ecriture de propriété CSS en JS

css-style-vs-feuille.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Accès à propriété CSS via attribut style d'éléments HTML</title>
7   <link rel="stylesheet" href="css-feuille.css">
8   <style>
9     div {
10       color: orange;
11     }
12   </style>
13 </head>
14
15 <body>
16   <div style="font-style:italic">green border, orange, italic</div>
17   <script>
18     let div = document.querySelector('div');
19     alert(div.style.border); //rien !
20     alert(div.style.color); //rien !
21     alert(div.style.fontStyle); //italic
22     div.style.color = "blue";
23     alert(div.style.color); //blue
24   </script>
25 </body>
26
27 </html>
```

Gestion d'évènements

Les évènements

Sont

- Déclenchés par l'utilisateur (clic souris, frappe au clavier, etc.) ou générés par une API représentant la progression d'une tâche asynchrone (minuteur, réponse HTTP, fin d'une vidéo, etc).
- Communiqués au code source lorsqu'ils surviennent.

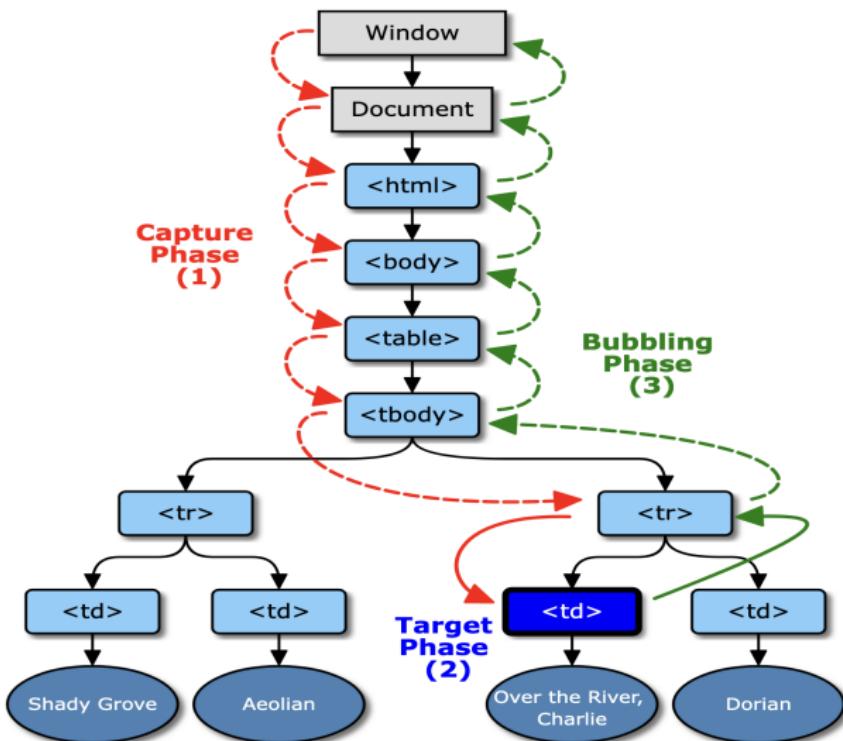
Plusieurs types d'évènements

- Standard (normalisés dans le DOM), non-standard, ou spécifique à un type de navigateur.
- Animation, batterie, appel, CSS, base de données, document, glisser-déposer, élément, focus, formulaire, frame, entrée (clavier, pad, souris), média (audio, vidéo), menu, réseau, notification, popup, impression, ressource, script, capteurs (compas, lampe, gyroscope, etc), session, carte, SMS, SVG, table, texte, touché, mise à jour, champ, vue, websocket, fenêtre, ...

Principes de la gestion d'évènements sur le DOM

- Un évènement a pour cible initiale un élément HTML.
- Un évènement se propage à tous les éléments situés sur la branche du DOM reliant la racine `window` à la cible en trois phases :
 - 1 Capture (*capture*) : de la racine à la cible.
 - 2 Cible (*target*) : sur l'élément cible.
 - 3 Bouillonnement (*bubbling*) : de la cible à la racine.
- Un écouteur (*listener*) intercepte un type d'évènements sur un élément dans une phase choisie en exécutant une fonction de rappel.
- L'enregistrement et le désenregistrement d'un écouteur se programme : objet cible, évènements écoutés, phase d'interception, fonction de rappel.

Phases de propagation



Les interfaces d'évènements

Tout évènement est un objet implémentant l'interface `Event` ou une sous-interface (p. ex. `MouseEvent` hérite de `UIEvent` qui hérite de `Event`)

`Event` fournit une information contextuelle aux écouteurs :

- `type` : son type (p. ex. `click`).
- `target` : son objet cible initial.
- `currentTarget` : l'objet courant sur lequel il est propagé.
- `bubbles` : s'il est en phase de bouillonnement ou de capture.
- `timestamp` : son horodatage relatif à l'`epoch`, etc.

Les sous-interfaces fournissent des informations supplémentaires

Par ex., `MouseEvent` a des propriétés relatives à la position de la souris, pression et relâchement du bouton, la molette, etc.

Les principaux évènements du DOM

Nom de l'événement	Action pour le déclencher
<code>click</code>	Cliquer (appuyer puis relâcher) sur l'élément
<code>dblclick</code>	Double-cliquer sur l'élément
<code>mouseover</code>	Faire entrer le curseur sur l'élément
<code>mouseout</code>	Faire sortir le curseur de l'élément
<code>mousedown</code>	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
<code>mouseup</code>	Relâcher le bouton gauche de la souris sur l'élément
<code>mousemove</code>	Faire déplacer le curseur sur l'élément
<code>keydown</code>	Appuyer (sans relâcher) sur une touche de clavier sur l'élément
<code>keyup</code>	Relâcher une touche de clavier sur l'élément
<code>keypress</code>	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
<code>focus</code>	« Cibler » l'élément
<code>blur</code>	Annuler le « ciblage » de l'élément
<code>change</code>	Changer la valeur d'un élément spécifique aux formulaires (<code>input</code> , <code>checkbox</code> , etc.)
<code>input</code>	Taper un caractère dans un champ de texte (son support n'est pas complet sur tous les navigateurs)
<code>select</code>	Sélectionner le contenu d'un champ de texte (<code>input</code> , <code>textarea</code> , etc.)

`submit`

Envoyer le formulaire

`reset`

Réinitialiser le formulaire

Enregistrement d'écouteurs (DOM-2)

En invoquant `addEventListener(type, listener[, ...])` sur tout objet implémentant `EventTarget`

- `type` : le type d'évènements à écouter (`string`).
- `listener` : la fonction à exécuter (ou objet implémentant `EventListener`) dont l'unique argument est l'objet évènement.

2 variantes pour l'argument optionnel :

- un booléen - faux par défaut - qui donne la priorité d'écoute à l'objet cible courant (= `this`) sur ses descendants dans le DOM.
- un objet à propriétés booléennes - fausses par défaut - :
 - `capture` : priorité d'écoute à l'objet cible courant (= `this`) sur ses descendants dans le DOM.
 - `once` : unique notification de l'écouteur suivie de sa suppression automatique.
 - `passive` : interdiction pour l'écouteur de bloquer l'action par défaut de l'évènement.

Enregistrement d'écouteurs

evenement-addeventlistener.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Enregistrement d'écouteurs</title>
6 </head>
7 <body>
8     <span id="clickme">Cliquez-moi !</span>
9     <script>
10        var span = document.querySelector('#clickme');
11        span.addEventListener('click', function() {
12            alert("Vous avez cliqué !");
13        }, false);
14        document.addEventListener('click', function() {
15            alert("Je confirme : vous avez cliqué !");
16        }, false);
17    </script>
18 </body>
19 </html>
```

Accès à l'objet cible initial

evenement-addeventlistener-2.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Accès à cible initiale</title>
6 </head>
7 <body>
8     <span id="clickme">Cliquez-moi !</span>
9     <script>
10        var span = document.querySelector('#clickme');
11        span.addEventListener('click', function(e) {
12            e.target.innerHTML += '<br>Vous avez cliqué sur un ${e.target.nodeName}
(1)';
13            console.log(e.target === this); //true
14        }, false);
15        document.addEventListener('click', function(e) {
16            e.target.innerHTML += '<br>Vous avez cliqué sur un ${e.target.nodeName}
(2)';
17            console.log(e.target === this); //false
18        }, false);
19    </script>
20 </body>
21 </html>
```

Accès à l'objet cible courant

Ne pas confondre cible initiale et cible courante

evenement-addeventlistener-3.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Accès à cibles initiale et courante</title>
6 </head>
7 <body>
8     <span id="clickme">Cliquez-moi !</span>
9     <script>
10        var body = document.querySelector('body');
11        body.addEventListener('click', function(e) {
12            console.log("clic sur " + e.target.nodeName); //sur SPAN (1)
13            console.log("propagation sur " + e.currentTarget.nodeName); //sur BODY (2)
14            console.log(e.currentTarget === e.target); //false
15            console.log(e.currentTarget === this);
16        });
17    </script>
18 </body>
19 </html>
```

Capture vs. bouillonnement

evenement-addeventlistener-4.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Capture et bouillonnement</title>
6 </head>
7 <body>
8     <div>
9         <span id="clickme">Cliquez-moi !</span>
10    </div>
11    <script>
12        var body = document.querySelector('body');
13        var div = body.querySelector('div')
14        var span = div.querySelector('#clickme');
15        body.addEventListener('click', function(e) {
16            console.log("L'évènement traverse " + e.currentTarget.nodeName);
17        }, true); //interception en phase de capture
18        div.addEventListener('click', function(e) {
19            console.log("L'évènement traverse " + e.currentTarget.nodeName);
20        }, false); //interception en phase de bouillonnement
21        span.addEventListener('click', function(e) {
22            console.log("L'évènement traverse le " + e.currentTarget.nodeName);
23        }, true); //valeur indifférente ici : interception en phase cible
24    </script>
25 </body>
26 </html>
```

Délégation d'évènements basée sur le bouillonnement

Créer un écouteur sur un parent plutôt que sur ses enfants

evenement-delegation.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8">
3 <title>Délégation d'évènements</title>
4 </head>
5 <body>
6   <ul id="parent-list">
7     <li id="post-1">Item 1</li>
8     <li id="post-2">Item 2</li>
9     <li id="post-3">Item 3</li>
10   </ul>
11   <script>
12     document.querySelector("#parent-list").addEventListener(
13       "click",
14       function(e) {
15         // !! balise en majuscules
16         if (e.target && e.target.nodeName == "LI") {
17           // e.target est l'item cliqué de la liste
18           alert("Item " + e.target.id.replace("post-", "") +
19             " a été cliqué!");
20         }
21       });
22   </script>
23 </body></html>
```

Gestionnaires d'évènements souris (1)

evenement-souris.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Evènements souris : mousemove</title>
6 </head>
7 <body>
8     <div id="position"></div>
9     <script>
10        var position = document.getElementById('position');
11        document.addEventListener('mousemove', function(e) {
12            position.innerHTML = ' X = ' + e.clientX + 'px Y = ' + e.clientY + 'px';
13        }, false); // écouteur sur le document, pas le DIV!
14    </script>
15 </body>
16 </html>
```

Gestionnaires d'évènements souris (2)

evenement-souris-1.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Evènements souris : mouseover, mouseout</title>
6 </head>
7 <body>
8     <div id="D" style="height:100px;width:100px;background-color:lightcoral;"></div>
9     <div id="R"></div>
10    <script>
11        var D = document.getElementById('D'),
12            R = document.getElementById('R');
13        D.addEventListener("mouseover", function(e) {
14            R.innerHTML += "Le curseur vient d'entrer dans " + e.currentTarget.id + "<br"
15        }, false);
16        D.addEventListener("mouseout", function(e) {
17            R.innerHTML += "Le curseur vient de sortir de " + e.currentTarget.id + "<br"
18        }, false);
19    </script>
20 </body>
21 </html>
```

Gestionnaires d'évènements clavier

evenement-clavier.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Évènements clavier : keypress, keyup, keydown</title>
6 </head>
7 <body>
8     <p><input id="field" type="text" /></p>
9     <table style="margin-top: 20px; border: 1px solid black;">
10         <tr><td>keydown</td><td></td></tr>
11         <tr><td>keypress</td><td></td></tr>
12         <tr><td>keyup</td><td></td></tr>
13     </table>
14     <script>
15         var cells = document.querySelectorAll("td:last-child");
16         document.addEventListener('keydown', e => cells[0].innerHTML = e.key, false);
17         document.addEventListener('keypress', e => cells[1].innerHTML = e.key, false);
18         document.addEventListener('keyup', e => cells[2].innerHTML = e.key, false);
19     </script>
20 </body>
21 </html>
```

Désenregistrement d'écouteurs (DOM-2)

Avec `removeEventListener(type, listener[, opti])` en repassant exactement les paramètres passés à `addEventListener()`

Suppose de passer une fonction nommée ou variable fonctionnelle comme `listener` à `addEventListener()`

evenement-removeeventlistener.html

```
1 <!DOCTYPE html>
2 <html><head><meta charset="utf-8" />
3 <title>Désenregistrer un gestionnaire d'évènements</title>
4 </head>
5 <body>
6   <span id="clickme">Cliquez-moi !</span>
7   <script>
8     var element = document.querySelector('#clickme');
9     var fa1 = function() {alert("Vous m'avez cliqué !");}
10    var fa2 = function() {alert("Je confirme : Vous m'avez bien cliqué !");}
11    element.addEventListener('click', fa1, false);
12    element.addEventListener('click', fa2, false);
13    element.removeEventListener('click', fa1, false);
14  </script>
```

Déactivation du comportement par défaut

Avec `Event.preventDefault()` qui empêche l'action par défaut associée à l'objet Event

evenement-preventdefault.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Inhiber le comportement par défaut</title>
6 </head>
7 <body>
8     <p>Please click on the checkbox control.</p>
9     <form>Checkbox:
10        <input id="c" type="checkbox" />
11    </form>
12    <div id="d"></div>
13    <script>
14        document.querySelector("#c").addEventListener("click", function(event) {
15            document.querySelector("#d").innerHTML += "no visible check!<br>";
16            event.preventDefault();
17        }, false);
18    </script>
19 </body>
20 </html>
```

Déactivation du comportement par défaut

evenement-preventdefault-formulaire.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Inhiber le comportement par défaut</title>
6 </head>
7 <body>
8     <form method="get" action="script.php">
9         <input name="i1" type="text" />
10        <input name="i2" type="text" />
11        <button type="submit">Go</button>
12    </form>
13    <script>
14        document.querySelector('form').onsubmit = function(e) {
15            if (Array.from(this.querySelectorAll('input')).some(i => i.value == "")) {
16                e.preventDefault();
17                alert('You need to fill in both names!');
18            }
19        } //== document.querySelector('form').addEventListener("submit",function(e)...,false);
20    </script>
21 </body>
22 </html>
```

Gestion des formulaires

La propriété value des champs

formulaire-value.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Formulaires : value</title>
7 </head>
8
9 <body>
10    <input class="fields" type="text" size="60" value="Vous n'avez pas le focus !" />
11    <textarea class="fields" rows="2" cols="55">Vous n'avez pas le focus !</textarea>
12    <script>
13        document.querySelectorAll('.fields').forEach((f) => {
14            f.addEventListener('focus', (e) => e.target.value = "Vous avez le focus !",
false);
15            f.addEventListener('blur', (e) => e.target.value = "Vous n'avez pas le focus
!", false);
16        });
17    </script>
18 </body>
19
20 </html>
```

Les propriétés booléennes des champs

disabled et readOnly

formulaire-disabled-readonly.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Formulaires : disabled, readOnly</title>
7 </head>
8
9 <body>
10    <input type="text" size="60" value="disabled => texte non sélectionnable" />
11    <br />
12    <input type="text" size="60" value="readOnly => texte sélectionnable" />
13    <script>
14        document.querySelectorAll("input[type='text']")[0].disabled = true;
15        document.querySelectorAll("input[type='text']")[1].readOnly = true;
16    </script>
17 </body>
18
19 </html>
```

La propriété checked des cases et boutons radio

formulaire-checked.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Formulaires : checked</title>
7 </head>
8
9 <body>
10    <label><input type="checkbox" name="cc[]" value="A" /> Case A</label><br />
11    <label><input type="checkbox" name="cc[]" value="B" /> Case B</label><br />
12    <label><input type="checkbox" name="cc[]" value="C" /> Case C</label><br />
13    <label><input type="radio" name="r" value="1" /> Bouton 1</label><br />
14    <label><input type="radio" name="r" value="2" /> Bouton 2</label><br />
15    <input type="button" value="Afficher les boutons cochés et les cases décochées" />
16 <script>
17     let C = Array.from(document.querySelectorAll("input[type='checkbox']"));
18     let R = Array.from(document.querySelectorAll("input[type='radio']"));
19
20     function check() {
21         C.filter(c => !c.checked).forEach(c => alert("Case décochée : " +
c.value));
22         R.filter(r => r.checked).forEach(r => alert("Bouton coché : " + r.value));
23     }
24     document.querySelector("input[type='button']").addEventListener('click', check);
25 </script>
26 </body>
27
28 </html>
```

Les propriétés selectedIndex et options des listes déroulantes

formulaire-selectedindex-options.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Formulaires : selectedIndex, options</title>
7 </head>
8
9 <body>
10    <select id="list">
11        <option value="aucune">Sélectionnez une couleur</option>
12        <option value="bleu">Bleu</option>
13        <option value="vert">Vert</option>
14    </select>
15    <script>
16        document.querySelector('#list').addEventListener('change', () =>
17            alert(list.options[list.selectedIndex].value), false);
18    </script>
19 </body>
20
21 </html>
```

Les méthodes submit() et reset() de l'élément form

formulaire-form.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Formulaires : submit(), reset()</title>
7 </head>
8
9 <body>
10    <form method="get" action="script.php">
11        <input type="text" name="t" value="Tapez" /> <br />
12        <button type="button">OK</button> <br />
13        <input type="submit" value="OK !" />
14        <input type="reset" value="RAZ !" />
15    </form>
16    <script>
17        let form = document.querySelector("form");
18        form.addEventListener('submit', e => {
19            if (!confirm('Soumettre ?')) {
20                e.preventDefault(); // empêche la soumission
21            }
22        }, false);
23        form.addEventListener('reset', e =>
24            alert('Vous avez réinitialisé le formulaire !'), false);
25        document.querySelector("button").addEventListener('click', () => form.submit(),
false);
```

Les méthodes de gestion du focus et de la sélection

focus(), blur() et select()

formulaire-focus.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Formulaires : focus(), blur()</title>
7 </head>
8
9 <body>
10    <form>
11        <input id="text" type="text" value="Entrez un texte" /> <br />
12        <button type="button">Donner le focus</button><br />
13        <button type="button">Retirer le focus</button><br />
14        <button type="button">Sélectionner le texte saisi</button>
15    </form>
16    <script>
17        let text = document.querySelector('#text');
18        document.querySelectorAll("button") [0].addEventListener('click', e =>
19            text.focus(), false);
20        document.querySelectorAll("button") [1].addEventListener('click', e =>
21            text.blur(), false);
22        document.querySelectorAll("button") [2].addEventListener('click', e =>
23            text.select(), false);
24    </script>
25 </body>
26
27 </html>
```

CM PHP : Introduction

Le cours PHP

Maîtriser les bases du langage PHP **version 8** :

- Variables, constantes, types
 - Instructions de contrôle
 - Chaînes de caractères
 - Tableaux
 - Fonctions
 - Formulaires
-
- Dates
 - Cookies, sessions
 - Programmation objet
 - Fichiers
 - XML
 - Bases de données

Le langage PHP

Origines

- PHP (acronyme pour Php Hypertext Processor) est un langage de scripts interprété.
- Crée en 1994 par Rasmus Lerdorf.
- Utilisé pour générer des pages Web dynamiques.

Un langage impératif, orienté objet, fonctionnel

- Structure proche du langage C.
- Typage dynamique.
- Tableaux associatifs, couche objet réflexive, ressources ...

Site et documentation officiels

Historique

Quelques versions choisies

- 4.1 : variables superglobales.
- 4.3 : CLI (Common Line Interface) en supplément de CGI (Common Gateway Interface).
- 5.0 : modèle objet.
- 5.3 : espaces de noms, résolution statique à la volée, fermetures, fonctions anonymes.
- 5.4 : traits.

Historique

PHP 7.0 (2015) - 7.1 (2016) - 7.2 (2017) - 7.3 (2018) - 7.4 (fév. 2020)

- Nouveau moteur : Zend engine.
- Prise en charge cohérente du 64 bits.
- Gain de performances en temps CPU et consommation mémoire.
- Amélioration de la hiérarchie des exceptions.
- De nombreuses erreurs fatales converties en exceptions.
- Un générateur de nombres aléatoires sécurisé.
- Suppression des interfaces SAPI et des extensions obsolètes.
- Typage des déclarations de fonction : retours et scalaires.
- Opérateurs null coalescent ?? et de comparaison <=>
- Classes anonymes, assertions à coût nul ...

Historique

PHP 8.0 (nov. 2020)

PHP 8.1 (nov. 2021)

Structure des fichiers HTML5

Une page dynamique PHP est un document HTML envoyé au client par le serveur.

Structure de document HTML 5 (pagehtml.html)

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Titre de la page</title>
6   </head>
7   <body>
8     <h2>Bienvenue sur le site PHP 7 </h2>
9   </body>
10 </html>
```

On pourrait aussi utiliser le suffixe .htm ou .php.

Première page PHP

```
1 <!DOCTYPE html>
2 <html lang="fr">
3     <head>
4         <meta charset="UTF-8" />
5         <title>Une page générée en PHP</title>
6     </head>
7     <body>
8         <?php
9             echo "<p>Aujourd'hui le </p>";
10            $d = date('d / M / Y H:m:s');
11        ?>
12        <hr/>
13        <?= $d ?>
14    </body>
15 </html>
```

- Les marqueurs `<?php` et `?>` délimitent un script PHP.
- `<?= $x ?>` équivaut au script `<?php echo $x; ?>`
- On peut entremêler scripts PHP et code HTML librement.

Cycle de vie d'une page PHP

Trois étapes :

- ① Envoi d'une requête HTTP par le client (eg. navigateur) du type
`http://www.server.com/codephp.php`
- ② Interprétation par le serveur du code PHP contenu dans la page demandée (ici `codephp.php`)
 - l'interpréteur renvoie le code HTML après évaluation du code PHP rencontré
- ③ Envoi par le serveur d'un fichier dont le contenu n'est que du HTML (+ CSS + JavaScript)
 - voir l'onglet `Code source` de la page sous Firefox

Inclusion de fichiers externes

Séparer le HTML du PHP pour plus de modularité/réutilisabilité

Pour pouvoir utiliser dans un script `a.php` des variables/fonctions/... stockées dans un script `b.php`, on utilise l'une des instructions suivantes :

- `include ("b.php")` : importe le contenu de `b.php` dans `a.php` sans générer d'erreur si `b.php` n'existe pas.
- `require ("b.php")` : idem mais génère une erreur fatale et met fin au script `a.php` en cas d'absence de `b.php`.
- `include_once` et `require_once` se comportent comme `include` et `require` respectivement mais importent une seule fois le fichier demandé.

`principal.php`

Extension de fichiers PHP externes

On peut utiliser pour extension de fichiers PHP :

- .inc : le navigateur affichera le contenu intégral du fichier.
- .inc.php : si le fichier ne contient que des affectations de variables (eg, paramètres sensibles), le serveur ne renverra rien au navigateur.

test.inc

```
1 $password = "WILL show in browser";
```

test.inc.php

```
1 $password = "WON'T show in browser";
```

Ajout de commentaires

Plusieurs types de commentaires :

- // sur une seule ligne
- # sur une seule ligne
- /* sur plusieurs lignes */
- /**
 - * sur plusieurs lignes
 - * pour génération automatique
 - * de documentation (PHPDoc)*/

Organisation de PHP

Organisation modulaire

- Module standard : accès aux types, instructions et fonctions élémentaires.
- Modules additionnels : ajout de fonctionnalités particulières (eg. accès et gestion de diverses bases de données).

Pour connaître la liste des modules disponibles :

- pour PHP en ligne de commande (CLI) :
`$ php -m`
- pour PHP déployé sur votre serveur, chargez le script :
`<?php phpinfo(); ?>`

Installation d'un serveur local

Éléments d'un serveur local :

- Serveur Apache.
- Interpréteur PHP.
- Système de gestion de bases de données ([MySQL](#), [MariaDB](#), [SQLite](#))
- Utilitaires [PHPMyAdmin](#) pour créer et gérer bases et tables de données MySQL/MariaDB, [SQLiteManager](#) pour SQLite, ...

Installation clé-en-main de plateformes *AMP

- [XAMPP](#) : multi-plateformes avec MariaDB
- [LAMP](#) pour Linux.
- [WAMP](#) pour Windows.
- [MAMP](#) pour Mac.

CM PHP : Variables et typage

Les variables

Une variable prend l'un des types suivants et peut en changer en cours d'exécution :

- entiers
- flottants
- chaînes de caractères
- booléens
- tableaux
- objets
- ressources
- nul

Identifiants de variables

Un identifiant de variable commence par le caractère \$ et est suivi du nom de la variable

- Le nom de la variable commence par une lettre ou le tiret-bas _
- Les caractères suivants sont des lettres, des chiffres ou _

Identifiants corrects

\$maVar

\$_maVar

\$MaVar2

\$_123

Identifiants incorrects

maVar

\$1maVar

\$+maVar

\$maVar*

Les identifiants de variables sont sensibles à la casse.

Déclaration et initialisation des variables

On déclare les variables où l'on veut dans un script.

Déclarer une variable sans l'initialiser est permis mais sans intérêt.

- Une variable non initialisée est de type NULL.

non-initialisation.php

```
1 $var;  
2 echo $var; // PHP Notice: Undefined variable: var in /Users/davidlesaint/...  
3 echo gettype($var); // NULL
```

Affectation de variable par valeur et par référence

L'affectation de variable consiste à attribuer une valeur à une variable

- Elle détermine le type de la variable (*typage dynamique*).
- Elle s'effectue par valeur ou par référence.

Affectation par valeur

- Syntaxe : `$var=exp` où `exp` dénote une expression PHP (littéral, variable, appel de fonction ...)
- Sémantique : `$var` prend la valeur de `exp`.

Affectation par référence avec & (esperluette)

- Syntaxe : `$var1=&$var2` où `$var2` dénote une variable.
- Sémantique : `$var1` est un alias de `$var2`.

Affectation de variable par valeur et par référence

On peut réaffecter une même variable au cours d'un script

- Si `$var1=$var2`, toute réaffectation de `$var2` n'aura aucune incidence sur `$var`.
- Si `$var1=&$var2`, toute réaffectation par valeur de l'une des variables sera répercutée sur l'autre mais les 2 variables seront dissociées si l'une est réaffectée par référence.
- Destruction de référence avec `unset ($var1)`.

L'affectation par valeur d'une variable objet/ressource équivaut à une affectation par référence

Il est possible de cloner (dupliquer) un objet.

Affectation de variable par valeur et par référence

affectation.php

```
1 echo '$x="X"; $y="Y"<br>';
2 $x = "X";
3 echo "> \${x==$x<br>}"; // X
4 $y = "Y";
5 echo "> \${y==$y<br>}"; // Y
6 echo '$y=&$x<br>';
7 $y = &$x;
8 echo "> \${x==$x<br>}"; // X
9 echo "> \${y==$y<br>}"; // X
10 echo '$x="X1"<br>';
11 $x = "X1";
12 echo "> \${y==$y<br>}"; // X1
13 echo '$y="Y"<br>';
14 $y = "Y";
15 echo "> \${x==$x<br>}"; // Y
16 $z = "Z";
17 echo '$z="Z"; $y=&$z<br>';
18 $y = &$z;
19 echo "> \${y==$y<br>}"; // Z
20 echo "> \${x==$x<br>}"; // Y
21 echo 'unset($y);<br>';
22 unset( $y );
23 echo "> \${y==$y<br>}"; // (null -> "")
24 echo "> \${z==$z<br>}"; // Z
```

Les variables dynamiques

Variables dont on détermine le nom en cours de script !

- Syntaxe : `$$var` où `$var` est une variable dont la valeur est une chaîne de caractères correspondant à un nom de variable bien formé.
- Accès :
 - en dehors d'une chaîne (eg, affectation) : `$$var=exp;`
 - dans une chaîne : `${$var}`

variables-dynamiques.php

```
1 $php="PHP";
2 $$php="Open Source";
3 echo $$php,"<br/>"; echo $PHP,"<br/>";
4 echo "$php est ${$php}", "<br/>";
5 $PHP="un langage";
6 echo "$php est ${$php}", "<br/>";
```

Les variables superglobales

Variables prédéfinies de type tableau, accessibles dans tout script, et contenant des informations sur le serveur et des données échangées : données de formulaires, fichiers, cookies, sessions ...

Identifiant	Description
<code>\$_GET</code>	Contient nom et valeur des champs de formulaires envoyés par HTTP GET. Les noms (attribut <code>name</code>) des champs du formulaire sont les clés du tableau.
<code>\$_POST</code>	Contient nom et valeur des champs de formulaires envoyés par HTTP POST. Les noms (attribut <code>name</code>) des champs du formulaire sont les clés du tableau.
<code>\$_COOKIE</code>	Contient nom et valeur des cookies enregistrés sur le poste client. Les noms des cookies sont les clés du tableau.
<code>\$_FILES</code>	Contient les noms des fichiers téléchargés à partir du poste client.
<code>\$_REQUEST</code>	Contient l'ensemble des superglobales <code>\$_GET</code> , <code>\$_POST</code> , <code>\$_COOKIE</code> et <code>\$_FILES</code> .
<code>\$_SERVER</code>	Contient les informations liées au serveur : contenu des en-têtes HTTP, nom du script en cours d'exécution, ...
<code>\$GLOBALS</code>	Contient nom et valeur des variables globales du script. Les noms des variables sont les clés du tableau. Exemple : <code>\$GLOBALS["var"]</code> récupère la valeur de <code>\$var</code> .
<code>\$_ENV</code>	Contient nom et valeur des variables d'environnement.
<code>\$_SESSION</code>	Contient l'ensemble des noms des variables de session et leur valeurs.

Superglobale `$_SERVER` : quelques éléments

Identifiant	Description
<code>\$_SERVER["DOCUMENT_ROOT"]</code>	La racine web sous laquelle le script est exécuté (eg. <code>/var/www</code>).
<code>\$_SERVER["PHP_SELF"]</code>	Chemin du script en cours d'exécution par rapport à la racine web.
<code>\$_SERVER["REQUEST_METHOD"]</code>	Méthode de requête HTTP utilisée pour accéder à la page (GET, ...).
<code>\$_SERVER["QUERY_STRING"]</code>	Chaîne de requête utilisée, si elle existe, pour accéder au script.
<code>\$_SERVER["HTTP_ACCEPT"]</code>	Contenu de l'en-tête <code>Accept</code> de la requête courante.
<code>\$_SERVER["HTTP_ACCEPT_CHARSET"]</code>	Contenu de l'en-tête <code>Accept-Charset</code> de la requête courante (eg. <code>'iso-8859-1,*;utf-8'</code>).
<code>\$_SERVER["HTTP_ACCEPT_ENCODING"]</code>	Contenu de l'en-tête <code>Accept-Encoding</code> de la requête courante (eg. <code>'gzip'</code>).
<code>\$_SERVER["HTTP_ACCEPT_LANGUAGE"]</code>	Contenu de l'en-tête <code>Accept-Language</code> de la requête courante (eg. <code>'fr'</code>).
<code>\$_SERVER["HTTP_USER_AGENT"]</code>	Contenu de l'en-tête <code>User-Agent</code> de la requête courante.
<code>\$_SERVER["HTTPS"]</code>	Valeur vide si le script n'a pas été appelé par HTTPS.

Les constantes personnalisées

Sont sensibles à la casse et se définissent

- Avec `const nom = valeur;`
- Ou par appel à `define(string nom, mixed valeur)`
- `defined(string nom)` teste si la constante de nom nom existe.

constantes.php

```
1 // Définition sensible a la casse
2 const PI = 3.1415926535;
3 // Utilisation
4 echo "La constante PI vaut ",PI,"<br />";
5 // Vérification de l'existence
6 if (defined( "PI")) echo "La constante PI est déjà définie","<br
/>";
7 // Vérification de l'existence et utilisation
8 if(define("site","http://www.funhtml.com")) {
9   echo "<a href=",site,">Lien vers mon site</a>";
10 }
```

Les constantes prédéfinies

Les constantes prédéfinies sont nombreuses !

```
<?php print_r(get_defined_constants()) ; ?>
```

Quelques exemples :

Nom	Description
PHP_VERSION	Version de PHP installée sur le serveur.
PHP_OS	Nom du système d'exploitation du serveur.
DEFAULT_INCLUDE_PATH	Chemin d'accès aux fichiers par défaut.
__FILE__	Nom du fichier en cours d'exécution.
__LINE__	Numéro de ligne du fichier en cours d'exécution.

Les types de données

Les types prédéfinis

Type abstrait	Type	Description
Scalaires	integer	entiers en base 2, 8, 10 ou 16
	float OU double	nombres décimaux
	string	chaînes de caractères
	boolean	les booléens TRUE et FALSE
Composés	array	tableaux
	object	objets
Spéciaux	resource	(références à) ressources externes
	null	type nul

Les pseudo-types de données

Mots-clés utilisés pour typer et documenter les prototypes de fonctions et méthodes

- Type d'arguments ou de valeur-retour attendus.
- Absence d'arguments ou de valeur-retour.
- Nombre d'arguments variables ...

Pseudo-type	Description
mixed	le paramètre peut accepter plusieurs types
number	le paramètre est de type integer ou float
callable	le paramètre est une fonction de rappel (alias callback)
array object	le paramètre est de type array ou object
\$...	nombre indéfini d'arguments
void	pas d'arguments ou de valeur-retour

Détermination du type d'une variable

`get_type(mixed $var):string`

Retourne le type de `$var` sous forme de string.

Tests de type (fonctions booléennes) :

- `is_scalar($var)` : teste si `$var` est un scalaire.
- `is_numeric($var)` : teste si `$var` est un nombre (integer ou float).
- `is_int($var)`, `is_float($var)`, `is_string($var)`, `is_bool($var)`.
- `is_array($var)`, `is_object($var)`.
- `is_resource($var)`, `is_null($var)`.

`var_dump($var)` : affiche type et valeur de `$var`.

Conversion de type (alias transtypage, coercition)

- Par affectation combinée à une coercition

```
$var2 = (type_désiré) $var1;
```

- En utilisant la fonction `settype()`

```
boolean settype($var, type_désiré)
```

coercition.php

```
1 $x="3.14 radians";
2 echo "\$x is string ".$x."  
"; // $x is string 3.14 radians
3 settype($x, "double");
4 echo "\$x is double ".$x."  
"; // $x is double 3.14
5 $y = (integer) $x;
6 echo "\$y is integer ".$y."  
"; // $y is integer 3
7 settype($y, "boolean");
8 echo "\$y is boolean ".$y."  
"; // $y is boolean 1
```

Utile en particulier pour traiter les données de formulaire qui sont par défaut de type `string`

Contrôler l'état d'une variable

- `isset ($var)` renvoie FALSE ssi \$var n'existe pas, ou n'est pas initialisée, ou vaut NULL.
- `empty ($var)` renvoie TRUE ssi \$var n'existe pas, ou n'est pas initialisée, ou vaut NULL, FALSE, 0, "", "0", ou [].

controle-etat.php

```
1 $unini; $null=NULL;
2 echo (int) isset ($undef) . "<br/>"; //0
3 echo (int) isset ($unini) . "<br/>"; //0
4 echo (int) isset ($null) . "<br/>"; //0
5 echo (int) !empty (NULL) . "<br/>"; //0
6 echo (int) !empty (FALSE) . "<br/>"; //0
7 echo (int) !empty (0) . "<br/>"; //0
8 echo (int) !empty (0.0) . "<br/>"; //0
9 echo (int) !empty ("") . "<br/>"; //0
10 echo (int) !empty ('0') . "<br/>"; //0
11 echo (int) !empty ([]). "<br/>"; //0
```

Utile pour tester si un champ de formulaire a bien été saisi

Contrôler l'état d'une variable : opérateur ??

Idiome classique (patron-is-set.php)

```
1 if (isset($_GET['user'])) {  
2     $nom = $_GET['user'];  
3 } else {  
4     $nom = 'nobody';  
5 }  
6 //équivalent à  
7 $nom = isset($_GET['user']) ? $_GET['user'] : 'nobody';
```

Opérateur ?? "null coalescing" (operateur-coalescing.php)

```
1 $nom = $_GET['user'] ?? 'nobody';  
2 //équivalent à  
3 $nom = isset($_GET['user']) ? $_GET['user'] : 'nobody';
```

Les entiers (type integer)

Codage

- Généralement sur 32 bits (varie selon les plateformes).
- Valeur comprise dans $\{-2^{31}, \dots, 2^{31} - 1\}$.
- Une variable entière est automatiquement convertie en double si sa valeur est en dehors de l'intervalle.

Représentations littérales

- Décimale : `$x=1789; $y=-476;`
- Binaire : `$x=0b1101111101; $y=-0b111011100;`
- Octale : `$x=03375; $y=-0734;`
- Hexadécimale : `$x=0x6FD; $y=-0x1DC;`

Les entiers sont affichés en base 10 via echo.

Les flottants (type double)

Le type double

Représente les nombres décimaux avec une précision de 14 chiffres.

- Voir librairie BCMath pour permettre des calculs plus précis.

Représentations littérales

- Notation décimale avec . : \$x=101.23;
- Notation scientifique avec E ou e : \$x=1.0123E2;

Les flottants sont affichés sous forme décimale si le nombre a moins de 15 chiffres, sous forme exponentielle sinon.

Les opérateurs numériques

S'appliquent aux opérandes de type numérique

Opérateur	Description
+	addition
-	soustraction
*	multiplication
**	exponentiation/puissance (associatif à droite)
/	division
%	modulo (s'applique aux flottants par restriction aux parties entières)
--	pré-déCREMENTATION (<code>--\$x</code>) ou post-déCREMENTATION (<code>\$x--</code>)
++	pré-INCRÉMENTATION (<code>++\$x</code>) ou post-INCRÉMENTATION (<code>\$x++</code>)

Les opérateurs d'affectation combinée

Réalisent une opération entre deux opérandes et affectent le résultat à l'opérande de gauche.

Opérateur	Description
<code>+=</code>	<code>\$x += \$y</code> équivaut à <code>\$x = \$x + \$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est un nombre.
<code>-=</code>	<code>\$x -= \$y</code> équivaut à <code>\$x = \$x - \$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est un nombre.
<code>*=</code>	<code>\$x *= \$y</code> équivaut à <code>\$x = \$x * \$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est un nombre.
<code>**=</code>	<code>\$x **= \$y</code> équivaut à <code>\$x = \$x**\$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est un nombre.
<code>/=</code>	<code>\$x /= \$y</code> équivaut à <code>\$x = \$x / \$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est un nombre différent de 0.
<code>%=</code>	<code>\$x %= \$y</code> équivaut à <code>\$x = \$x % \$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est un nombre différent de 0.
<code>.=</code>	<code>\$x .= \$y</code> équivaut à <code>\$x = \$x . \$y</code> . <code>\$y</code> peut être une expression complexe dont la valeur est une chaîne de caractères.

Les fonctions mathématiques

De nombreuses fonctions sont disponibles par défaut :

- **Conversions** : integer hexdec(string) ...
- **Arrondis** : double ceil(double) ...
- **Trigonométrie** : double cos(double) ...
- **Logarithmes** : double log(double X, double B) ...
- **Génération de nombres aléatoires** : integer rand(integer min, integer max) ...
- ...

Les booléens (type boolean)

Le type boolean

- Contient uniquement les valeurs TRUE et FALSE.
- Est à la base des instructions conditionnelles.

Pour l'affichage, PHP assimile TRUE à "1" et FALSE à "".

En contexte booléen, évaluation d'expression

- à valeur booléenne
 - `$x<10` vaut TRUE ssi `$x` est de valeur inférieure à 10.
- à valeur non booléenne :
 - `$x` vaut TRUE ssi `$x` est initialisée à une valeur *non nulle*.

Règles d'évaluation booléenne d'expressions

Expressions évaluées à FALSE	Une expression logique fausse utilisant un ou plusieurs opérateurs de comparaison
Le mot-clé FALSE	
La valeur entière 0	
La valeur décimale 0.0	
Les chaînes "" et "0"	
Une variable de type null	
Une variable non initialisée	
Un tableau vide	
Un objet sans propriétés ni méthodes	
Expressions évaluées à TRUE	Toute autre possibilité dont : <ul style="list-style-type: none">- Les entiers strictement positifs ou négatifs- La chaîne "FALSE"- Les variables de type resource

Règles d'évaluation booléenne d'expressions

evaluation-booleenne.php

```
1 // Expressions fausses
2 if (!FALSE)      var_dump (FALSE) ;      // bool(false)
3 if (!0)            var_dump (0) ;          // int(0)
4 if (!0.0)          var_dump (0.0) ;        // float(0)
5 if (! "")           var_dump ("") ;         // string(0) ""
6 if (! "0")          var_dump ("0") ;        // string(1) "0"
7 if (!null)        var_dump (null) ;        // NULL
8 if (!isset ($x)) echo "1" ;                // 1
9 if (![])           var_dump ([]);          // array(0) {}

10 // Expressions vraies
11 if (-1)            var_dump (-1) ;        // int(-1)
12 if ("FALSE")     var_dump ("FALSE") ;    // string(5) "FALSE"
13 $f = fopen (__FILE__, "r");
14 if ($f)            var_dump ($f) ;        // resource(5) of type (stream)
```

Les opérateurs booléens

Se divisent en

- Opérateurs de comparaison (binaires).
- Opérateurs logiques de composition (unaires/binaires).

Opérateurs de comparaison

Opérateur	Description
<code>==</code>	égalité
<code>!= ou <></code>	inégalité
<code><</code>	strictement inférieur
<code><=</code>	inférieur ou égal
<code>></code>	strictement supérieur
<code>>=</code>	supérieur ou égal
<code><=></code>	<code>\$x<=>\$y</code> vaut -1 ssi <code>\$x<\$y</code> , 0 ssi <code>\$x==\$y</code> , 1 ssi <code>\$x>\$y</code> ("spaceship operator" introduit en PHP 7)
<code>====</code>	égalité des valeurs et des types
<code>! ==</code>	inégalité des valeurs ou des types

Comparaison et coercition implicite

La comparaison via `==` d'arguments de types différents transtype l'un des arguments en appliquant les règles de transtypage PHP

operateur-comparaison.php

```
1 var_dump(FALSE==0); // TRUE
2 var_dump(FALSE==0.0); // TRUE
3 var_dump(FALSE==""); // TRUE
4 var_dump(FALSE=="0"); // TRUE
5 var_dump(FALSE==NULL); // TRUE
6 var_dump(FALSE==[]); // TRUE
7 var_dump(0.0==0); // TRUE - conversion (double) 0
8 var_dump(0==""); // TRUE - conversion (integer) ""
9 var_dump(41=="41ok12"); // TRUE - conversion (integer) "41ok12"
10 var_dump(41==" 41ok12"); // TRUE - conversion (integer) " 41ok12"
11 var_dump(0=="a41ok12"); // TRUE - conversion (integer) "a41ok12"
12 var_dump(NULL==""); // TRUE
13 var_dump(NULL==[]); // TRUE
14 var_dump("")=="0"); // FALSE
15 var_dump("")==[]); // FALSE
16 var_dump(0==[]); // FALSE
```

Comparaison stricte

Pour éviter les désagréments, privilégier les opérateurs === et !==.

opérateur-comparaison-stricte.php

```
1 var_dump(TRUE === 1); // FALSE
2 var_dump("") === NULL; // FALSE
3 var_dump(1e2 === "100"); // FALSE
```

Les opérateurs booléens

Opérateurs logiques

Opérateur	Description
OR	disjonction
	équivaut à OR mais prioritaire sur =
XOR	disjonction exclusive
AND	conjonction
&&	équivaut à AND mais prioritaire sur =
!	négation

Priorité des opérateurs en PHP

Les chaînes de caractères

Suites de caractères alphanumériques encadrées par

- des apostrophes : ' PHP7 et MySQL'
- des guillemets : "PHP7 et MySQL"

Traitement d'une variable apparaissant dans une chaîne

- Son identifiant est affiché si la chaîne est encadrée par des apostrophes.
 - <?php \$x="A"; echo '\$x'; ?> affiche \$x.
- Sa valeur est affichée si la chaîne est encadrée par des guillemets.
 - <?php \$x="A"; echo "\$x"; ?> affiche A.

Affichage mixte à l'aide de séquences d'échappement

```
<?php $x="A"; echo "\$x vaut $x"; ?> affiche $x vaut A.
```

Les tableaux

Un tableau stocke des *éléments* (valeurs) indépendants

- Les éléments peuvent prendre n'importe quel type.
- Les éléments peuvent être de types différents.

On peut créer implicitement des tableaux multi-dimensionnels avec des tableaux de tableaux.

Deux variantes de tableaux :

- Tableau *indicé* : les éléments sont repérés par des indices numériques.
- Tableau *associatif* : les éléments sont repérés par des clés de type chaîne.

Les types spéciaux

Le type `resource`

- Une valeur de type `resource` est une référence vers une ressource externe : fichier ouvert, connexion à base de données (BDD), image, ...
- Libérées automatiquement par le ramasse-miettes SAUF cas des connexions persistantes à BDD.

Le type `null` (ou `NULL`)

Attribué à une variable sans contenu ou qui a été explicitement initialisée à la valeur `NULL`.

- `""` et `"0"` ont le type `string`.
- `0` a le type `integer`.

CM PHP : Chaînes de caractères

Les chaînes de caractères

Suites de caractères encadrées par des

- Apostrophes : ' PHP 8 et MySQL'
- Guillemets : "PHP 8 et MySQL"

Importance des chaînes de caractères

- Constituent l'essentiel du contenu des pages Web.
- Sont manipulées pour créer des pages à partir de fichiers ou de BDD.
- Sont le type des données envoyées par formulaire, des cookies, des données de sessions ...

Affichage des chaînes de caractères

Avec `echo` ou la fonction `print()`

Traitement des variables apparaissant dans une chaîne

- Affichage littéral de son identifiant si la chaîne est encadrée par des apostrophes.
 - `<?php $x="A"; echo '$x'; ?>` affiche \$x.
- Interpolation (affichage de sa valeur) si la chaîne est encadrée par des guillemets.
 - `<?php $x="A"; echo "$x"; ?>` affiche A.

Affichage mixte à l'aide de séquences d'échappement

```
<?php $x="A"; echo "\$x vaut $x"; ?> affiche $x vaut A.
```

Séquences d'échappement

Pour afficher

- Des caractères protégés du langage (eg, \$).
- Des caractères de contrôle (eg. un retour à la ligne).
- Des caractères à partir de leur code ASCII octal ou hexadécimal.

Séquence	Signification
\'	affiche une apostrophe
\"	affiche un guillemet droit
\\$	affiche le symbole \$
\\"	affiche une barre oblique inversée \ (alias backslash)
\n	nouvelle ligne (code ASCII 0x0A)
\r	retour chariot (code ASCII 0x00)
\t	tabulation (code ASCII 0x09)
\[0-7] {1,3}	affiche le caractère dont le code ASCII en octal correspond à la séquence de caractères (séquence de 1 à 3 chiffres pris entre 0 et 7). echo "\101"; affiche A
\x[0-9 A-F a-f] {1,2}	affiche le caractère dont le code ASCII en hexadécimal correspond à la séquence de caractères (séquence de 1 à 2 caractères). echo "\x4A"; affiche J

Concaténation de chaînes

Avec l'opérateur point (.)

```
1 $str1 = "AA";
2 $str2 = "BB";
3 $str3 = $str1.$str2."<br/>";
4 echo $str3; // affiche AABB<br/>
```

Avec la virgule (,) pour echo

```
1 $str1 = "AA";
2 $str2 = "BB";
3 echo $str1,$str2,"<br/>"; // affiche AABB<br/>
```

Syntaxes alternatives

La syntaxe Heredoc permet de définir de longues chaînes (eg, fragment HMTL) et fonctionne comme les guillemets

```
1 $p = 2;  
2 $str=<<<IDF  
3 hello 'ma' "variable" \${p} ${p}  
4 IDF;  
5 // !!! AUCUN caractère avant et après "IDF;"  
6 echo $str;
```

La syntaxe Nowdoc permet de définir de longues chaînes et fonctionne comme les apostrophes

```
1 $p = 2;  
2 $str=<<<'IDF'  
3 hello variable \${p} ${p}  
4 IDF;  
5 // !!! AUCUN caractère avant et après "IDF;"  
6 echo $str;
```

Affichage formatté

A la C

```
void printf(string format, string $ch1, ...string $chN)
```

- Affiche le contenu de \$ch1,...\$chN
- Selon le format spécifié dans la chaîne format.

Autres fonctions (\$tab est un tableau de chaînes) :

- string sprintf(string format, string \$ch1, ...string \$chN).
- string vprintf(string format, array \$tab).
- void vsprintf(string format, array \$tab).

Chaîne de formatage format composée :

- D'un texte personnalisé.
- Et de directives d'affichage constituées de caractères spéciaux indiquant comment les variables string en paramètre sont incorporées et affichées.

Directives d'affichage

Composées, dans l'ordre,

Du caractère % suivi par :

- Un caractère de remplissage (une espace par défaut ou utiliser 'x' pour caractère de remplissage x).
- Le caractère – pour un alignement du 'mot' à gauche (à droite par défaut).
- Un nombre indiquant le nombre de caractères pour la chaîne formattée.
- Un point suivi d'un entier pour fixer le nombre de décimales à afficher.
- Une lettre spécifiant le type de valeur à afficher :
 - %b pour affichage en binaire, %d en base 10, %f en flottant, %o en octal, %x ou %X en hexadécimal.
 - %c pour affichage par code ASCII.
 - %s pour affichage littéral.

Exemples

sprintf.php

```
1 echo "<h3>Votre facture </h3>";
2 echo sprintf("<b>%' _28s %' _22s %' _22s <br /></b>", "Prix H.T.", "T.V.A.", "Prix
T.T.C.");
3 $ht[1] = 27.55 ;
4 $ht[2] = 3450.40 ;
5 $ht[3] = 320.75 ;
6 $total=0;
7 for ($i=1;$i<4;$i++) {
8   echo "Article $i",sprintf ("%' _20.2f %' _22.2f %' _20.2f <br
/>", $ht[$i], $ht[$i]*0.196, $ht[$i]*1.196);
9   $total+=$ht[$i];
10 }
11 echo str_repeat("*",71), "<br />";
12 echo "TOTAL",sprintf ("%' _20.2f %' _22.2f %' _20.2f <br
/>", $total, $total*0.196, $total*1.196);
```

Ordre de passage modifiable avec %n\\$c

```
1 $ch1 = "Monsieur" ;
2 $ch2 = "Madame" ;
3 echo sprintf ('%2$s, %1$s, ...', $ch1, $ch2) ;
```

Longueur, Accès, Codage ASCII

`strlen(string) :int`

Donne le nombre de caractères de la chaîne.

`empty(mixed) :boolean`

Teste si une chaîne est vide.

Accès aux caractères avec `[]`

Les chaînes de caractères sont assimilables à des tableaux indicés (le premier caractère est d'indice 0).

`ord(string l) :int et chr(int c) :string`

- `ord` retourne le code ASCII du caractère `l`.
- `chr` retourne le caractère de code ASCII `c`.

Longueur, Accès, Codage ASCII

```
1 for ($i=1 ; $i<6 ; $i++) {  
2     $nb=rand(65,90);  
3     $code .=chr($nb);  
4 }  
5 echo "Votre mot de passe est : ",$code;
```

Modification de la casse

Minuscules / Majuscules / Capitalisation

- `strtolower(string)` : string : tout en minuscules.
- `strtoupper(string)` : string : tout en majuscules.
- `ucfirst(string)` : string : 1ère lettre en majuscule.
- `ucwords(string)` : string : 1ère lettre de chaque mot en majuscule.

Suppression des caractères invisibles

- `trim(string $ch [, charlist])` : string supprime tout espace, tabulation, retour chariot et caractère nul apparaissant au début et en fin de \$ch ou bien tout caractère dans charlist
- `ltrim(...)` supprime en début de chaîne.
- `rtrim(...)` supprime en fin de chaîne.

Entités HTML et caractères spéciaux

`htmlspecialchars(string $ch [, ...]):string`

- Remplace les caractères & " ' < > par leurs entités HTML.
- Utile pour créer des noeuds texte HTML ou XML contenant des caractères spéciaux HTML/XML.

`htmlspecialchars_decode(string $ch [int $flags]):string`

- Fonction inverse.

`htmlentities(string $ch [, ...]):string`

- Remplace tous les caractères éligibles (Unicode>128) par leurs entités HTML.

`html_entity_decode(string $ch [, ...]):string`

- Fonction inverse.

Entités HTML et caractères spéciaux

`addslashes(string $ch) :string`

- Echappe (ajoute \ devant) toute occurrence de ' " \ et du caractère NUL dans \$ch.
- Utile pour enregistrer des données envoyées par formulaire en BDD (sans requêtes préparées).

`stripslashes(string $ch) :string`

- Fonction inverse.

`quotemeta(string $ch) :string`

- Echappe les caractères réservés des regex . \ + * ? [] () \$ ^

Recherche de sous-chaînes

`substr_count(string $ch1, string $ch2):int`

- Renvoie le nombre d'occurrences de `$ch2` dans `$ch1`.

`strstr(string $ch1, string $ch2):string`

- Renvoie `FALSE` si `$ch2` n'apparaît pas dans `$ch1` ou sinon tous les caractères allant de la 1ère occurrence de `$ch2` dans `$ch1` jusqu'à la fin de `$ch1`.

`substr(string $ch, int i [, int N]):string`

- Renvoie la sous-chaîne de `$ch` commençant à la position `i` et de longueur maximale `N`.

Recherche de sous-chaînes

```
str_replace(string $ch1, string $ch2, string $ch [,  
string $n]):string
```

- Remplace les occurrences de \$ch1 par \$ch2 dans \$ch. \$n passée par référence est le nombre de remplacements effectués.

exemple4-5.php

```
1 $ch = "Perette et le pot au lait. C'est pas de pot!" ;  
2 $ssch = substr($ch, 8, 9) ;  
3 echo $ssch, "<br />" ;  
4 $ssch = substr($ch, 8);  
5 echo $ssch , "<br />";  
6 $ch2="pot";  
7 $nb=substr_count($ch,$ch2);  
8 echo "Le mot $ch2 est présent $nb fois dans $ch <br />";  
9 $ch3=str_replace('pot','bol',$ch);  
10 echo $ch3, "<br />" ;
```

Recherche de position ou d'existence d'un mot

```
strpos(string $ch1, string $ch2 [, int p]):int
```

- Donne la position de la chaîne \$ch2 dans \$ch1 en commençant au début de la chaîne ou à la position p. Renvoie FALSE si le motif n'est pas trouvé.

strpos.php

```
1 $chn="go do you go to the togo ?";
2 $pattern="go";
3 $pos=0;
4 while (true) {
5     $pos=strpos($chn,$pattern,$pos);
6     /* ATTENTION : l'usage du == au lieu du === ne donnerait pas le
7      * fonctionnement "attendu" car la position 0 serait automatiquement
8      * convertie à FALSE et le test réussirait */
9     if ($pos === false) break;
10    echo "found pattern at $pos\n";
11    ++$pos;
12 }
```

Capture de sous-chaînes dans des variables

```
sscanf(string $ch, string "format" [,  
$var1,$var2,...]):array|int|null
```

Extrait de \$ch et stocke dans \$var1, \$var2 ... les éléments correspondant au format spécifié dans format à l'aide de spécificateurs (comme pour printf). Retourne le nombre d'éléments.

exemple4-7.php

```
1 $personne = "1685-1750 Jean-Sébastien Bach";  
2 $format=%d-%d %s %s";  
3 $nb = sscanf($personne, $format, $ne, $mort, $prenom, $nom);  
4 echo "$prenom $nom né en $ne, mort en $mort <br />";  
5 echo "Nous lisons $nb informations";
```

Comparaison de chaînes

`==` compare une chaîne à un nombre en la convertissant en nombre
(ses premiers caractères numériques sont retenus)

- Conversion identique pour les opérations arithmétiques (`+`, `-`, `*`, `/`, `%`) entre chaîne et nombre.

`<`, `>`, `<=` et `>=` comparent les chaînes selon le code ASCII

Alternatives :

- `strcmp(string $ch1, string $ch2):int` : sensible à la casse.
- `strcasecmp(string $ch1, string $ch2):int` : insensible à la casse.

strcmp.php

```
1 $ch1 = "Blanc"; $ch2 = "Bleu"; $ch3 = "blanc";
2 echostrcasecmp($ch1,$ch3). "<br/>"; // affiche 0
3 echo strcmp($ch1,$ch2). "<br/>"; // affiche -4
```

Les expressions régulières

Expression régulière (alias motif, masque, regex, modèle)

Chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles.

- Adresses e-mail.
- Numéros de téléphone.
- Code INSEE ...

Encadrement des motifs avec / et /

/motif/ pour la regex motif.

Caractères spéciaux (alias méta-caractères)

\ + * ? [] () \$ ^

Motifs élémentaires

Recherche de chaînes précises

- `/angers/` : `angers` apparaît dans la chaîne analysée.
- `/\.fr/` : `.fr` apparaît.
- `/a|b/` : `a` ou `b` apparaît.

Recherche d'un ou plusieurs caractères

- `/[axz]/` : l'un des caractères `a`, `x`, `z`.
- `/[b-p]/` : l'une des minuscules entre `b` et `p`.
- `/[A-Z]/` : une majuscule.
- `/[0-9]/` : un chiffre.

Recherche de métacaractères

A échapper avec l'antislash.

Classes de caractères prédéfinies

Classe	Recherche
<code>[[:alnum:]]</code>	Tous les caractères alphanumériques : [a-zA-Z0-9].
<code>[[:alpha:]]</code>	Tous les caractères alphabétiques : [a-zA-Z].
<code>[[:blank:]]</code>	Tous les caractères blancs : espaces, tabulations ...
<code>[[:ctrl:]]</code>	Tous les caractères de contrôle.
<code>[[:digit:]]</code>	Tous les chiffres : [0-9]
<code>[[:print:]]</code>	Tous les caractères imprimables sauf caractères de contrôle.
<code>[[:punct:]]</code>	Tous les caractères de ponctuation.
<code>[[:space:]]</code>	Tous les caractères d'espace : espaces, tabulations, sauts de ligne, ...
<code>[[:upper:]]</code>	Tous les caractères en majuscules : [A-Z].
<code>[[:xdigit:]]</code>	Tous les caractères en hexadécimal.

Restriction de caractères

Avec ^ entre crochets

- / [^axz] / : un caractère différent de a, x, z.
- / [^A-Z] / : un caractère qui n'est pas une majuscule.

Début et fin de chaîne :

Avec ^ hors crochets

- / ^axz / : toute chaîne démarrant par axz.

Avec \$

- / axz\$/ : toute chaîne finissant par axz.

Motifs généraux

N'importe quel caractère

Avec . (non échappé)

- /a.z/ : toute chaîne contenant a suivi d'un caractère suivi de z.

Répétition de caractères

Avec ? (0 ou 1 fois), + (au moins une fois), * (0 ou plus)

- /ab?/ : présence d'un a non suivi d'un b ou suivi d'un seul b.
- /ab+/ : présence d'un a suivi d'une suite d'au moins un b.
- /ab*/ : présence d'un a suivi ou non d'une suite de b.

Motifs généraux

Sous-motifs

Regroupement avec ()

- / (ab) + / : toute chaîne contenant une série de ab.

Contraintes de cardinalité

Avec {n} (n répétitions)

- / (ab) {5} / : série de 5 ab.

Avec m, n (entre m et n répétitions) :

- / (ab) {3, 5} / : série de 3 à 5 ab.

Avec m, (au moins m répétitions) :

- / (ab) {3, } / : série d'au moins 3 ab.

Les fonctions de recherche

```
preg_match(string $motif, string $ch [, array  
$tab]):int|false
```

- Recherche une sous-chaîne de \$ch satisfaisant la regex \$motif. Retourne 1 le cas échéant, 0 sinon ou FALSE si une erreur survient.
- \$tab est un tableau indicé contenant \$ch comme premier élément puis toutes les sous-chaînes satisfaisant à \$motif comme éléments suivants.

preg-match.php

```
1 $ch1="30-19-1970";  
2 $ch2="4-01-18";  
3 $motif="/([0-9]{1,2})-([0-9]{2})-([0-9]{2,4})/i";  
4 $result=array();  
5 preg_match($motif,$ch1,$result);  
6 print_r($result); //Array([0]=> 30 [1]=> 19 [2]=> 1970)  
7 preg_match($motif,$ch2,$result);  
8 print_r($result); //Array([0]=> 4 [1]=> 01 [2]=> 18)
```

Les fonctions de recherche

```
preg_replace(string $motif, string $rep, string  
$ch) :string|array|null
```

- Remplace toute occurrence du motif \$motif dans \$ch par \$rep et retourne la chaîne résultat.

preg-replace.php

```
1 $chn="30-09-1970";  
2 $motif="/(\d+)-(\d+)-(\d{2,4})/i";  
3 $remplace="$3/$2/$1";  
4 $str=preg_replace($motif,$remplace,$chn);  
5 echo "chaîne de départ : $chn\n";  
6 echo "chaîne résultat : $str\n";
```

Autres fonctions pour les expression régulières

- preg_split
- preg_grep
- preg_filter
- preg_match
- preg_match_all

CM PHP : Tableaux

Les tableaux

Un tableau stocke des *éléments* (valeurs) indépendants

- Les éléments peuvent prendre n'importe quel type.
- Les éléments peuvent être de types différents.

Deux types de tableaux :

- Tableau *indicé* : les éléments sont repérés par des indices numériques.
- Tableau *associatif* : les éléments sont repérés par des *clés* (chaîne ou variable de type chaîne).

Les tableaux indicés

Création d'éléments avec []

- `$tab[] = exp;` ajoute l'élément `exp` en fin du tableau `$tab`, c.a.d. à l'indice qui suit l'indice maximum ou 0 si `$tab` est vide.
- `$tab[n] = exp;` initialise/remplace l'élément d'indice `n` du tableau `$tab` à/avec l'expression `exp`.
- `print_r` formate l'affichage de tableaux.

tableau-indice-creation1.php

```
1 $tab[0] = "UFR"; // équivalent ICI à $tab[] = "UFR";
2 $tab[1] = "Sciences"; // équivalent ICI à $tab[] = "Sciences";
3 $tab[49] = "Angers";
4 $tab[] = "janvier"; // équivalent ICI à $tab[50] = "janvier";
5 $i = 3;
6 $tab[$i] = "Université"; // équivalent ICI à $tab[3] = "Université";
7 $tab[] = "2017"; // équivalent ICI à $tab[51] = "2017";
8 echo count($tab), "<br/>"; // affiche 6
9 print_r($tab);
```

Les tableaux associatifs

Création d'éléments avec []

Les clés (littéral ou variable `string`) remplacent les indices.

- Les clés ne comportent pas d'espace, sont encadrées par apostrophes/guillemets, et sont sensibles à la casse.
- Encadrer tout élément de tableau utilisé dans une chaîne par des accolades
 - `"{$tab['alpha']} "` et non pas `"$tab['alpha']"`.

tableau-associatif-creation1.php

```
1 $tab ["zero"]="UFR";
2 $tab [1]="Sciences";
3 $tab ["forty-nine"]="Angers";
4 $tab []="janvier"; // équivalent ICI à $tab[2]="janvier";
5 $tab ["3"]="Université";
6 $tab [51]="2017";
7 echo count ($tab), "<br/>";
8 print_r($tab);
```

Interpolation d'expressions tableaux

tableau-interpolation.php

```
1 <?php
2 // création des éléments du tableau
3 $tab["php"] = "php.net";
4 $tab["mysql"] = "mysql.com";
5 $tab["html"] = "w3.org";
6 // création des liens
7 echo "<h2> Mes liens préférés </h2>";
8 echo "<ul><li><a href=\" http://www.$tab['php']\" title=\"Le site php.net\">&nbsp;
PHP </a> </li>";
9 echo "<li><a href=\" http://www.$tab['mysql']\" title=\"Le site mysql.com\">&nbsp;
MySQL </a> </li>";
10 echo "<li><a href=\" http://www.$tab['html']\" title=\"Le site du W3C\">&nbsp; HTML
</a> </li></ul>";
```

Les tableaux

Fonctions natives

- Crédit de tableaux.
- Lecture des éléments.
- Ajout et retrait d'éléments.
- Tri des éléments.
- Sélection d'éléments.
- Fonctions sur tableau.

Création de tableaux avec array() ou []

```
$tab=[v0, ..., vN];  
$tab=array(v0, ..., vN);
```

Crée un tableau indicé à $N+1$ éléments :

- Le 1er élément de valeur v_0 a pour indice 0.
- Accès au $i+1$ -ième élément : $\$tab[i]$.

```
$tab=[ "k1"=>v1, ..., "kN"=>vN];  
$tab=array( "k1"=>v1, ..., "kN"=>vN);
```

Crée un tableau associatif à N éléments :

- Chaque élément est une paire clé " k " - valeur v .
- Pas de notion d'ordre entre les éléments.
- Accès par clé aux éléments : $\$tab["k"]$.

Tableaux multidimensionnels par tableaux de tableaux

```
$tab=[ [...], ..., [...]];
$tab=array(array(...), ..., array(...));
$tab=["k1"=>[ ...], ..., "kN"=>[ ...]];
$tab=array("k1"=>array(...), ..., "kN"=>array(...));
```

Accès : \$tab[a][b] où a et b sont des indices ou clés selon le cas.

array-multidimensional.php

```
1 $tab=[ ["A-0", "A-1", "A-2"],
2           ["B-0", "B-1", "B-2"],
3           ["C-0", "C-1", "C-2"],
4           ["D-0", "D-1", "D-2"]];
5 echo "<h3>Tableau multidimensionnel</h3><table><tbody>";
6 for ($i=0; $i<count($tab); $i++) {
7     echo "<tr>";
8     for ($j=0; $j<count($tab[$i]); $j++)
9         echo "<td><h3> .. ", $tab[$i][$j], " .. </h3></td>";
10    echo "</tr>";
11 }
12 echo " </tbody> </table> ";
13 //-----($tab)
```

Suites de nombres/lettres

`range(int m, int M):array`

- Crée un tableau indicé contenant les entiers de m à M .

`range(char c, char C):array`

- Crée un tableau indicé contenant les caractères de c à C selon le code ASCII.

`array-range.php`

```
1 // Suite de nombres de 1 à 10
2 $tabnombre= range(1,10);
3 print_r($tabnombre);
4 echo "<hr>";
5 // Suite de lettres de a à z avec une boucle
6 for ($i=97;$i<=122;$i++) {
7     $stabalpha[$i-96]=chr($i);
8 }
9 print_r($stabalpha);
10 echo "<hr>";
11 //Suite des caractères de Y à b avec range()
12 $stabalpha2 = range("Y", "b");
13 print_r($stabalpha2);
```

Transformations de chaînes en tableaux

`explode(string sep, string $ch [, int N]):array`

- Décompose la chaîne `$ch` en tableau de mots (sous-chaînes) selon le séparateur `sep` fourni. `N` est le nombre maximum de mots recherchés.

`implode(string sep, array $tab):string`

- Fonction inverse.

Transformations de chaînes en tableaux

Exemple4-8.php

```
1 //Passage chaîne -> tableau
2 $ch1="L'avenir est à PHP7 et MySQL";
3 $tab1=explode(" ",$ch1);
4 echo $ch1, "<br />";
5 print_r($tab1);
6 echo "<hr />";
7 $ch2="C:\wampserver\www\php7\chaines\string2.php";
8 $tab2=explode("\\",$ch2);
9 echo $ch2, "<br />";
10 print_r($tab2);
11 echo "<hr />";
12 //Passage tableau -> chaîne
13 $tab3[0]="Bonjour";
14 $tab3[1]="monsieur";
15 $tab3[2]="Rasmus";
16 $tab3[3]="Merci!";
17 $ch3=implode(" ",$tab3);
18 echo $ch3, "<br />";
```

Décompte des éléments

`count(array $tab) ou int sizeof(array $tab):int`

- Renvoie le nombre d'éléments.

Pour tableau "multi-dimensionnel"

```
1 // Comptage du nombre d'éléments
2 $tab=array ("Bonjour", "Web", array ("1-0", "1-1", "1-2"),
3             1970, 2013, array ("3-0", "3-1", "3-2", "3-3"));
4 echo "Le tableau \$tab a ", count ($tab), " éléments <br />";
5 //ou encore: echo "Le tableau \$tab a ",sizeof($tab)," éléments <br />";
6 // Comptage du nombre de valeurs
7 $nb_val=0;
8 for ($i=0; $i<count ($tab) ; $i++) {
9     if (gettype ($tab [$i] ) == "array") {
10         $nb_val+=count ($tab [$i] );
11     } else {
12         $nb_val++;
13     }
14 }
15 echo "Le tableau \$tab a ", $nb_val, " valeurs <br />";
```

Décompte des valeurs

`array_count_values(array $tab):array`

- Renvoie un tableau associatif ayant pour clés les valeurs de \$tab et pour valeur le nombre d'occurrences de chaque valeur dans \$tab.
- Ne s'applique qu'aux tableaux de nombres et/ou chaînes.

`array-count-values.php`

```
1 $tab= ["PHP", "JavaScript", "PHP", "ASP", "PHP", "ASP"];  
2 $result=array_count_values($tab);  
3 echo "Le tableau \$tab contient ",count($tab), " éléments <br>";  
4 echo "Le tableau \$tab contient ",count($result), " valeurs  
différentes <br>";  
5 print_r($result);
```

Lecture des éléments avec boucles for et while

Avec boucle for

```
1 $montab=array ("Paris", "London", "Brüssel");
2 for ($i=0; $i<count ($montab) ;$i++) {
3   echo "L'élément $i est $montab[$i]<br />";
4 }
```

Avec boucle while

```
1 $montab=array ("Paris", "London", "Brüssel");
2 $i=0;
3 while (isset ($montab[$i])) {
4   echo "L'élément $i est $montab[$i]<br />";
5   $i++;
6 }
```

Lecture des éléments avec `list()`

L'expression `list ($x1, ..., $xN) = $tab`

- Ne s'applique qu'aux tableaux indicés `$tab`.
- Affecte la i -ième valeur de `$tab` à `$xi`.
- Ne modifie pas le pointeur interne de `$tab`.

Filtrage des valeurs

`list ($x1,, $x4) = $tab` récupère dans `$x1` et `$x4` les éléments d'indice 0 et 3.

Lecture des éléments avec `foreach()`

```
foreach($tab as $val) {bloc}
```

- Pour tableaux indicés.
- La variable `$val` contiendra successivement chacune des valeurs du tableau `$tab`.

```
foreach($tab as $key=>$val) {bloc}
```

- Pour tableaux associatifs.
- La paire de variables (`$key, $val`) correspondra successivement à chaque élément (clé, valeur) du tableau `$tab`.

Exemple avec foreach

array-foreach.php

```
1 //*****
2 //Lecture de tableau indicé sans récupération des indices
3 //*****
4 $stab=array ("Paris", "London", "Brüssel");
5 echo "<H3>Lecture des valeurs des éléments </H3>";
6 foreach ($stab as $ville) {
7   echo "<b>$ville</b> <br>";
8 }
9 echo "<hr>";
10 //*****
11 //Lecture de tableau indicé avec récupération des indices
12 //*****
13 echo "<h3>lecture des indices et des valeurs des éléments </h3>";
14 foreach ($stab as $indice=>$ville) {
15   echo "L'élément d'indice <b>$indice</b> a la valeur <b>$ville</b><br>";
16 }
17 echo "<hr>";
18 //*****
19 //Lecture de tableau associatif avec récupération des clés
20 //*****
21 $stab2=array ("France"=>"Paris", "Great Britain"=>"London", "België"=>"Brüssel");
22 echo "<h3>lecture des clés et des valeurs des éléments</h3>";
23 foreach ($stab2 as $cle=>$ville) {
24   echo "L'élément de clé <b>$cle</b> a la valeur <b>$ville</b> <br>";
25 }
26 echo "<hr>";
```

Ajout et retrait d'éléments

`array_push($tab, v1, ..., vN) :int`

- Ajoute `v1, ..., vN` en fin de `$tab` et en retourne la taille.
- Indexation de `v1, ..., vN` à partir de `count($tab)`.

`array_pop($tab) :mixed`

- Supprime et retourne le dernier élément de `$tab` s'il existe, `NULL` sinon.

`unset(mixed $tab [, mixed $...]) :void`

Utilisé avec un élément de tableau comme argument :

- Supprime l'élément.
- Sans conséquence sur les indices/clés des éléments restants.

Ajout et retrait d'éléments

`array_unshift ($tab, v1, ..., vN) : int`

- Ajoute `v1, ..., vN` en début de `$tab` et en retourne la taille.
- Ré-indexation à partir de 0 avec décalage-droite pour éléments indicés, clés inchangées pour les autres.

`array_shift ($tab) : mixed`

- Supprime et retourne le premier élément de `$tab` s'il existe, `NULL` sinon.
- Ré-indexation à partir de 0 avec décalage-gauche pour éléments indicés, clés inchangées pour les autres.

`array_unique ($tab) : array`

Supprime les valeurs en doublons et ne conserve que la "dernière" occurrence.

- Sans conséquence sur les indices/clés des éléments restants.

Exemple

array-pop-shift.php

```
1 $tab= array(800,1492, 1515, 1789); print_r($tab);
2 echo "<hr />";
3 // Ajout au début du tableau
4 $poitiers=732;
5 $nb=array_unshift($tab,500,$poitiers);
6 echo "Le tableau \${$tab} a maintenant $nb éléments <br>"; print_r($tab);
7 echo "<hr />";
8 // Ajout à la fin du tableau
9 $armi=1918;
10 $newnb=array_push($tab,1870,1914,$armi);
11 echo "Le tableau \${$tab} a maintenant $newnb éléments <br>"; print_r($tab);
12 echo "<hr />";
13 // Suppression du dernier élément
14 $suppr= array_pop($tab);
15 echo "Le tableau \${$tab} a perdu l'élément $suppr <br>"; print_r($tab);
16 echo "<hr />";
17 // Suppression du premier élément
18 $suppr= array_shift($tab);
19 echo "Le tableau \${$tab} a perdu l'élément $suppr <br>"; print_r($tab);
20 echo "<hr />";
21 // Suppression de l'élément d'indice 4
22 unset($tab[4]);
23 echo "L'élément d'indice 4 a été supprimé <br>"; print_r($tab);
```

Extraction d'éléments avec array_slice()

```
array_slice(array $tab, int i, int n):array
```

Retourne un sous-tableau de \$tab différent selon le signe et la valeur de i et n :

- ① `array_slice($tab, 2, 3)` : les 3 premiers éléments à partir de celui d'indice 2.
- ② `array_slice($tab, 2, -3)` : tous les éléments à partir de celui d'indice 2 sauf les 3 derniers.
- ③ `array_slice($tab, -2, 3)` : les 3 éléments précédant l'avant-dernier.
- ④ `array_slice($tab, -3, -2)` : les éléments d'indices virtuels négatifs entre -3 compris et -2 non compris.

exemple5-14.php

Tri d'éléments

Différentes méthodes

- Tri sur tableaux indicés ou sur tableaux associatifs.
- Tri sur les valeurs ou sur les clés.
- Critères de tri prédéfinis (ordre alphabétique, ordre ASCII ...).
- Critères de tri personnalisés.
- Préservation ou non des indices/clés.

Tri sur tableaux indicés

Selon l'ordre ASCII

- `sort(array &$tab) :bool` : tri des valeurs en ordre croissant.
Perte des correspondances indices-valeurs.
- `rsort(array &$tab) :bool` : tri des valeurs en ordre décroissant. Perte des correspondances indices-valeurs.
- `array_reverse(array $tab) :array` : inverse l'ordre des valeurs. Perte des correspondances indices-valeurs.

array-sort-rsort-reverse.php

Tableau indicé d'origine

```
Array ( [0] => Blanc2 [1] => Jaune [2] => rouge [3] => Vert [4] => Bleu [5] => Noir [20] => Blanc10 )
```

Tri en ordre ASCII sans sauvegarde des indices

```
Array ( [0] => Blanc10 [1] => Blanc2 [2] => Bleu [3] => Jaune [4] => Noir [5] => Vert [6] => rouge )
```

Tri en ordre ASCII inverse sans sauvegarde des indices

```
Array ( [0] => rouge [1] => Vert [2] => Noir [3] => Jaune [4] => Bleu [5] => Blanc2 [6] => Blanc10 )
```

Inversion de l'ordre des éléments

```
Array ( [0] => Blanc10 [1] => Noir [2] => Bleu [3] => Vert [4] => rouge [5] => Jaune [6] => Blanc2 )
```

Tri sur tableaux indicés ou associatifs

Selon l'ordre naturel

- `natsort (array &$tab) :bool` : tri des valeurs selon l'ordre "naturel" ($a_2 < a_{10}$). Préservation des correspondances indices/clés-valeurs.
- `natecatesort (array &$tab) :bool` : comme `natsort` mais insensible à la case.

Boucle `foreach` indispensable pour itérer selon le nouvel ordre.

array-natsort.php

Tableau indicé d'origine

```
Array ( [0] => Blanc2 [1] => Jaune [2] => rouge [3] => Vert [4] => Bleu [5] => Blanc10 [6] => 1ZZ [7] => 2AA )
```

Tri en ordre naturel avec sauvegarde des indices

```
Array ( [6] => 1ZZ [7] => 2AA [0] => Blanc2 [5] => Blanc10 [4] => Bleu [1] => Jaune [3] => Vert [2] => rouge )
```

Tri en ordre naturel insensible à la casse avec sauvegarde des indices

```
Array ( [6] => 1ZZ [7] => 2AA [0] => Blanc2 [5] => Blanc10 [4] => Bleu [1] => Jaune [2] => rouge [3] => Vert )
```

Tri des valeurs sur tableaux associatifs

Avec préservation des correspondances clés-valeurs

- `asort (array &$tab) :bool` : tri ASCII des valeurs en ordre croissant.
- `arsort (array &$tab) :bool` : tri ASCII des valeurs en ordre décroissant.

array-asort-arsort.php

Tableau associatif d'origine

Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des valeurs avec sauvegarde des clés

Array ([w10] => Blanc10 [w2] => Blanc2 [blu] => Bleu [y] => Jaune [bla] => Noir [g] => Vert [r] => rouge)

Tri en ordre ASCII inverse avec sauvegarde des clés

Array ([r] => rouge [g] => Vert [bla] => Noir [y] => Jaune [blu] => Bleu [w2] => Blanc2 [w10] => Blanc10)

Tri en ordre naturel avec sauvegarde des clés

Array ([w2] => Blanc2 [w10] => Blanc10 [blu] => Bleu [y] => Jaune [bla] => Noir [g] => Vert [r] => rouge)

Tri en ordre naturel insensible à la casse avec sauvegarde des clés

Array ([w2] => Blanc2 [w10] => Blanc10 [blu] => Bleu [y] => Jaune [bla] => Noir [r] => rouge [g] => Vert)

Tri des clés sur tableaux associatifs

Avec préservation des correspondances clés-valeurs

- `ksort (array &$tab) :bool` : tri ASCII des clés en ordre croissant.
- `krsort (array &$tab) :bool` : tri ASCII des clés en ordre décroissant.

array-ksort-krsort.php

Tableau associatif d'origine

Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [bla] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des clés

Array ([bla] => Noir [bla] => Bleu [g] => Vert [r] => rouge [w10] => Blanc10 [w2] => Blanc2 [y] => Jaune)

Tri en ordre ASCII inverse des clés

Array ([y] => Jaune [w2] => Blanc2 [w10] => Blanc10 [r] => rouge [g] => Vert [bla] => Bleu [bla] => Noir)

Tri selon la longueur des clés

Array ([w10] => Blanc10 [bla] => Bleu [bla] => Noir [w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert)

Manipulation du pointeur interne de tableaux

- mixed current(array \$tab)
 - Retourne (la valeur de) l'élément courant de \$tab.
- int|string|null key(array \$tab)
 - Retourne la clé de l'élément courant de \$tab.
- mixed end(array &\$tab)
 - Retourne la valeur du dernier élément de \$tab (ou FALSE si \$tab est vide) et positionne le pointeur en fin de \$tab.
- mixed next(array &\$tab)
 - Avance le pointeur interne de \$tab et retourne l'élément courant ou FALSE.
- mixed prev(array &\$tab)
 - Recule le pointeur interne de \$tab et retourne l'élément courant ou FALSE.
- mixed reset(array &\$tab)
 - Repositionne le pointeur interne de \$tab sur le premier élément.

CM PHP : Structures de contrôle

Les instructions de contrôle

Instructions de contrôle des scripts

Syntaxe et sémantique proches du C/C++.

Trois types

- Instructions conditionnelles :

- if et if...else
- ? et ??
- switch...case

- Instructions de boucle :

- for
- while et do...while
- foreach
- break, continue , goto

- Gestion des erreurs :

- error_reporting
- try...catch...finally

L'instruction if

```
if(exp) instruction;
```

instruction est exécutée si exp est évaluée à TRUE.

```
if(exp) {bloc}
```

Le bloc d'instructions est exécuté si exp est évaluée à TRUE.

Différentes types d'expressions possibles :

- Booléenne ($\$x > 2$) ou non ($\y avec $\$y = "abc"$).
- Atomique ($\$x > 2$) ou composite ($\$x > 2 \text{ || } \y).

Confer cours 2

- Tableau des règles d'évaluation booléenne d'expression.
- Liste des opérateurs logiques de comparaison et composition.

L'instruction if...else

```
if(exp) ins1; else ins2;
```

Exécute ins1 si exp est évaluée à TRUE ou ins2 sinon.

```
if(exp) {bloc1} else {bloc2}
```

Exécute bloc1 si exp est évaluée à TRUE ou bloc2 sinon.

control-if-else.php

```
1 $prix=55;
2 if ($prix>100)
3 {
4 echo "<b>Pour un montant d'achat de $prix &#8364;, la remise est de 10 % </b><br>";
5 echo "Le prix net est de ",$prix*0.90;
6 }
7 else
8 {
9 echo "<b>Pour un montant d'achat de $prix &#8364;, la remise est de 5 %</b><br>";
10 echo "<h3>Le prix net est de ",$prix*0.95," &#8364;</h3>";
11 }
```

Les if imbriqués avec if...elseif...else

control-if-elseif.php

```
1 // ****if...elseif...else*****
2 $cat = "PC";
3 $prix = 900;
4 if ($cat == "PC") {
5     if ($prix >= 1000) {
6         echo "<b>Pour l'achat d'un PC d'un montant de $prix &#8364;, la remise est de 15
%</b><br>";
7         echo "<h3> Le prix net est de : ", $prix * 0.85, "&#8364; </h3>";
8     } else {
9         echo "<b>Pour l'achat d'un PC d'un montant de $prix &#8364;, la remise est de 10
%</b><br>";
10        echo "<h3> Le prix net est de : ", $prix * 0.90, "&#8364; </h3>";
11    }
12 } elseif ($cat == "Livres") {
13     echo "<b>Pour l'achat de livres la remise est de 5 %</b><br />";
14     echo "<h3> Le prix net est de : ", $prix * 0.95, "&#8364; </h3>";
15 } else {
16     echo "<b>Pour les autres achats la remise est de 2 %</b><br>";
17     echo "<h3> Le prix net est de : ", $prix * 0.98, "&#8364; </h3>";
18 }
```

L'opérateur ternaire ?

exp ? val1 : val2

Renvoie val1 si exp est évaluée à TRUE ou val2 sinon.

Usage : \$var = exp ? val1 : val2;

Equivaut à : if(exp) {\$var=val1;} else {\$var=val2;}

control-operator-if-else.php

```
1 $ch = "Bonjour ";
2 $sexe="M";
3 $ch .= ($sexe=="F") ? "Madame" : "Monsieur";
4 echo "<h2>$ch</h2>";
5 $nb = 3;
6 $pmu ="Il faut trouver ".$nb;
7 $mot = ($nb==1) ? " cheval" : " chevaux";
8 echo "<h3> $pmu $mot </h3>";
```

L'opérateur binaire *null coalescent* ??

`$var ?? exp`

Renvoie `$var` si `$var` initialisée à non NULL ou `exp` sinon.

Usage : `$x = $y ?? exp;`

Equivaut à : `$x = isset($y) ? $y : exp;`

control-operator-coalescing.php

```
1 // 3 instructions équivalentes
2 $x = $_GET['user'] ?? 'aie';
3 $x = isset($_GET['user']) ? $_GET['user'] : 'aie';
4 if(isset($_GET['user'])) $x = $_GET['user']; else $x = 'aie';
```

Chaînage possible

```
1 $x = $_GET['user'] ?? $_POST['user'] ?? 'aie';
```

L'instruction switch...case

Alternative à if...elseif...else

Pour simplifier l'écriture de branchements conditionnels imbriqués.

Syntaxe

```
switch(exp) {  
    case val1:  
        bloc1;  
        break;  
    ...  
    case valN:  
        blocN;  
        break;  
    default:  
        blocD;  
        break;  
}
```

Sémantique

- Si **exp vaut val1**, le bloc1 est exécuté et l'exécution passe à la fin du bloc switch.
- Sinon, la procédure est réitérée sur les valeurs val2 ...valN jusqu'à concordance.
- Si aucune concordance n'est trouvée, le blocD est exécuté.

L'instruction switch...case

control-switch-case.php

```
1 $dept = 75;
2 switch ($dept) {
3     // Premier cas
4     case 75:
5         case "Capitale":
6             echo "Paris";
7             break;
8     // Deuxième cas
9     case 78:
10        echo "Hauts de Seine";
11        break;
12     // Troisième cas
13     case 93:
14         case "Stade de France":
15             echo "Seine Saint Denis";
16             break;
17     // Cas par défaut
18     default:
19         echo "Département inconnu en Ile de France";
20         break;
21 }
```

La boucle for

Syntaxe

for(exp1;exp2;exp3) {instruction;}

ou for(exp1;exp2;exp3) {bloc}

Sémantique

- ① exp1 est évaluée.
- ② exp2 est évaluée en contexte booléen : si elle vaut TRUE, l'instruction (ou le bloc d'instructions) est exécutée, sinon on sort du bloc for.
- ③ exp3 est exécutée et la boucle reprend à l'étape (2) jusqu'à ce que exp2 soit évaluée à FALSE.

control-for.php

```
1 for ($i = 1; $i < 7; $i++) {  
2     echo "<h$i> $i :Titre de niveau $i </h$i>";  
3 }
```

Boucle à plusieurs variables

Sous-expressions séparées par des virgules

- Multiples initialisations (eg, plusieurs compteurs).
- Multiples conditions.
- Multiples in-/dé-crémentations.

control-for1.php

```
1 for ($i=1, $j=9; $i<10, $j>0; $i++, $j-) {  
2 // $i varie de 1 à 9 et $j de 9 à 1  
3     $css="style=\"border-style:double; border-width:3;\"";  
4     echo "<span ".$css.">$i + $j=10</span>";  
5 }
```

Boucles imbriquées

control-for-nested.php

```
1 echo "<h2>Révisez votre table de multiplication!</ h2>";
2 //Début du tableau HTML
3 echo "<table border=\"2\" style=\"background-color:yellow\">
<th>&nbsp;X &nbsp;</th>";
4 //Création de la première ligne
5 for ($i=1 ; $i<10 ; $i++)
6   echo "<th>&nbsp; $i &nbsp;</th>";
7 //Création du corps de la table
8 //Boucles de création du contenu de la table
9 for ($i=1 ; $i<10 ; $i++) {
10   //Création de la première colonne
11   echo "<tr><th>&nbsp; $i &nbsp;</th>";
12   //Remplissage de la table
13   for ($j=1 ; $j<10 ; $j++) {
14     echo "<td style=\"background-color:red;color:white\">
&nbsp; &nbsp; <b>". $i*$j . "&nbsp; &nbsp; </td>";
15   }
16   echo "</b></tr>";
17 }
18 echo "</table>"
```

La boucle while

Alternative à for

Quand on ne connaît pas a priori le nombre limite d'itérations.

Exemple : afficher les résultats d'une requête sur une BDD.

`while(exp) {instruction;} ou while(exp) {bloc}`

Tant que `exp` est évaluée à TRUE, exécute l'instruction (ou le bloc d'instructions).

control-while.php

```
1 $n=1;
2 while ($n%7 !=0)
3 {
4   $n = rand(1,100);
5   echo $n, "  /";
6 }
```

La boucle do...while

```
do {bloc} while(exp);
```

- Similaire à while mais bloc est au moins exécutée une fois avant d'évaluer exp.
- Les variables évaluées dans exp peuvent être initialisées dans bloc.

control-do-while.php

```
1 do
2 {
3     $n = rand(1,100);
4     echo $n, " &nbsp; / ";
5 }
6 while ($n%7 !=0);
```

La boucle `foreach`

Pour itérer sur les éléments d'un tableau

- Plus efficace que `for`.
- Adaptée aux tableaux associatifs : elle ne nécessite pas d'en connaître la taille ou les clés.

Deux syntaxes selon que l'on souhaite récupérer

- Les valeurs uniquement.
- Les valeurs et les clés/indices.

foreach pour récupérer les valeurs d'un tableau

```
foreach($tab as $val){bloc}
```

- La variable `$val` contiendra chacune des valeurs du tableau `$tab`.
 - La variable `$tab` peut être remplacée par une expression de type `array`.
-
- Veiller à ne pas utiliser un nom de variable existant pour `$val` (écrasement).
 - Les variables utilisées dans la boucle ne sont pas locales et gardent leur dernière valeur en sortie de boucle.

foreach pour récupérer les valeurs d'un tableau

control-foreach.php

```
1 //Création du tableau de 9 éléments
2 for ($i=0 ; $i<=8 ; $i++) { $tab[$i] = pow(2,$i); }
3 $val = "Une valeur";
4 echo $val, "<br />";
5 //Lecture des valeurs du tableau
6 echo "Les puissances de 2 sont :";
7 foreach ($tab as $val) { echo $val." : "; }
8 foreach ($tab as $val) { echo $val." : "; } // on repart bien à "zéro"
```

foreach pour récupérer indices et valeurs d'un tableau

```
foreach($tab as $key=>$val) {bloc}
```

La paire de variables (\$key,\$val) correspondra successivement à chaque élément (indice,valeur) du tableau \$tab.

control-foreach1.php

```
1 //Création du tableau
2 for ($i=0 ; $i<=8 ; $i++) {
3     $tab[$i] = pow(2, $i);
4 }
5 //Lecture des indices et des valeurs
6 foreach ($tab as $ind=>$val) {
7     echo " 2 puissance $ind vaut $val <br />";
8 }
9 echo "Dernier indice ", $ind, " , dernière valeur ", $val;
```

foreach pour récupérer clés et valeurs d'un tableau

control-foreach2.php

```
1 //Création d'un tableau associatif
2 for ($i=0 ; $i<=8 ; $i++) {
3     $tabass ["client".$i] = rand(100,1000);
4 }
5 //Lecture des clés et des valeurs
6 foreach ($tabass as $cle=>$val) {
7     echo " Le client de login <b>$cle</b> a le code
<b>$val</b><br />";
8 }
```

foreach pour itérer sur les propriétés d'un objet

Assimilées aux éléments d'un tableau associatif

- Nom de propriété = clé.
- Valeur de propriété = valeur.

Sortie anticipée de boucle avec break

break

Arrête une boucle `for`, `foreach` ou `while` avant son terme

- N'arrête pas le script contrairement à `exit`.
- Arrête uniquement la boucle qui le contient dans le cas de boucles imbriquées.
- `break n;` arrête les `n` boucles les plus internes l'imbriquant.

Sortie anticipée de boucle avec break

control-break.php

```
1 //Création d'un tableau de noms
2 $tab [1]="Basile"; $tab [2]="Conan";
3 $tab [3]="Albert"; $tab [4]="Vincent";
4 //Boucle de lecture du tableau
5 for ($i=1;$i<count ($tab) ;$i++) {
6   if ($tab [$i] [0]=="A") {
7     echo "Le premier nom commençant par A est le numéro $i:
", $tab [$i];
8     break;
9   }
10 }
```

- $\$tab[\$i][n]$ ($n \geq 0$) correspond à la $n+1$ -ième lettre de la chaîne $\$tab[\$i]$.
- $count (\$tab)$ évitera une boucle infinie éventuelle si aucun mot de $\$tab$ ne démarre par A.

Avancement de boucle avec continue

continue

Arrête l'itération en cours, pas la boucle.

control-continue.php

```
1 //Interruption d'une boucle for
2 for ($i=0;$i<20;$i++)
3 {
4     if ($i%5==0) { continue; }
5     echo $i, "<br />";
6 }
7 //Interruption d'une boucle foreach
8 $tab[1]="Ain";
9 $tab[2]="Allier";
10 $tab[27]="Eure";
11 $tab[28]="Eure-et-Loir";
12 $tab[29]="Finistère";
13 $tab[33]="Gironde";
14 foreach ($tab as $cle=>$valeur)
15 {
16     if ($tab[$cle][0]!="E") { continue; }
17     echo "code $cle : département ",$tab[$cle], "<br />";
18 }
```

Avancement de boucle avec continue

continue n;

Arrête les n-1 boucles les plus internes l'imbriquant et l'itération courante de la n-ième.

control-continue1.php

```
1 for ($i=0 ; $i<10 ; $i++)
2 {
3     for ($j=0 ; $j<10 ; $j++)
4     {
5         for ($k=0 ; $k<10 ; $k++)
6         {
7             if (( $i+$j+$k ) %3==0) continue 3;
8             echo "$i : $j : $k <br /> ";
9         }
10    }
11 }
```

Gestion des erreurs

Filtrage des messages d'erreur renvoyés au poste client

- Nomenclature (partielle) des erreurs en PHP :

Constante	Valeur	Niveau d'affichage
E_ERROR	1	Erreur fatale (eg. appel de fonction inexistante) : le script s'arrête.
E_WARNING	2	Avertissement (eg. division par 0) : le script se poursuit.
E_PARSE	4	Erreur de syntaxe : le script s'arrête.
E_NOTICE	8	Avis de problème simple.
...		
E_ALL	2^{15}	Toute erreur.

- Le type d'erreurs relayées est prédéfini et configurable dans le fichier de configuration `php.ini`
- On peut l'ajuster pour chaque script en y insérant `error_reporting(n)` ; au début ce qui n'affichera que les erreurs dont les valeurs correspondent aux bits positionnés dans `n`.

Gestion des erreurs

control-error-reporting.php

```
1 error_reporting(E_WARNING); // E_ALL, E_WARNING ...
2 $x=2;$y=0;
3 echo $x/$y;
4 fopen("nofile.txt", "r");
5 jenexistepas();
```

Suppression des messages d'erreur par fonction

- @f() : faire précéder l'appel de la fonction f par @.

Exceptions

Mécanismes d'interception et traitement d'erreurs

Gestion d'exception avec `try...catch...finally` et utilisant la classe prédefinie `Exception`.

Syntaxe

```
try {  
    if(test){  
        throw new Exception(m,c);  
    } else {  
        blocI  
    }  
    catch(Exception $e)  
    {  
        blocG  
    }  
    finally  
    {  
        bloc  
    }  
}
```

Sémantique

`try{...}` délimite le bloc dans lequel peut survenir l'erreur :

- `test` est la condition nécessaire à l'erreur.
- `throw` lance un objet `Exception` créé par `new` avec message d'erreur `m` et code d'erreur `c`.
- `blocI` est le bloc exécuté si l'erreur est évitée.

`catch(...){...}` intercepte n'importe quel objet `e` de classe `Exception` lancé dans `try{...}` :

- `blocG` est le bloc exécuté pour récupérer et afficher des informations sur `e` (`m, c, ...`).

`finally{...}` délimite un bloc systématiquement exécuté.

La classe Exception

Son constructeur prend 2 arguments qui initialise 2 des propriétés de chaque objet créé :

- message : le message d'erreur sous forme de chaîne de caractères.
- code : le code d'erreur sous forme d'entier.

Méthodes de la classe Exception

Méthode	Définition
<code>__construct(m, c)</code>	Constructeur de l'objet (appelé implicitement avec <code>new Exception(m, c)</code>) prenant les deux paramètres <code>m</code> et <code>c</code> .
<code>getCode()</code>	Retourne la valeur de la propriété <code>message</code> .
<code>getMessage()</code>	Retourne la valeur de la propriété <code>code</code> .
<code>getFile()</code>	Retourne la valeur de la propriété <code>file</code> contenant nom et chemin d'accès du fichier dans lequel s'est produit l'erreur.
<code>getLine()</code>	Retourne la valeur de la propriété <code>line</code> correspondant au numéro de ligne à laquelle a été créée l'exception.
<code>__toString()</code>	Retourne une chaîne contenant toutes les informations sur l'exception.

Exemple

Alternative à l'absence de gestion d'erreur ou à la suppression d'avertissement via `error_reporting`

```
1 $a=100;$b=0;
2 try{
3     if ($b==0) {
4         throw new Exception("Division par 0", 7);
5     } else {
6         echo "Résultat de : $a / $b = ",$a/$b;
7     }
8 } catch (Exception $e) {
9     echo "<hr>Message d'erreur : ",$e->getMessage();
10    echo "<hr>Code d'erreur : ",$e->getCode();
11    echo "<hr>Nom du fichier :",$e->getFile();
12    echo "<hr>Numéro de ligne :",$e->getLine();
13    echo "<hr>__toString : ",$e->__toString();
14 } finally {
15     echo "<hr>L'exception a été traitée, le
script continue.";
16 }
17 echo "<hr>Vraie Fin";
```

Exemple amélioré avec boîte d'alerte

control-throw-catch1.php

```
1 $a=10;
2 $b=0;
3 try {
4     if ($b==0) {
5         throw new Exception("Division par 0", 7);
6     } else {
7         echo "Résultat de : $a / $b = ",$a/$b;
8     }
9 } catch (Exception $e) {
10     $c = $e->getCode();
11     $m = $e->getMessage();
12     echo "<script>alert(' Erreur numéro ",$c, " \\\n ",$m, " '";
);</script>";
13 } finally {
14     echo "Tout est sous contrôle<br/>";
15 }
16 echo "FIN";
```

Exceptions personnalisées

Par création de classe

Héritant et spécialisant la classe `Exception` avec nouvelles propriétés et méthodes.

control-exception.php

```
1 class MonException extends Exception {
2     public function alerte() {
3         $this->message = "<script> alert(' Erreur numéro ". $this->getCode() . " \n
". $this->getMessage() . " ' )</script> ";
4         return $this->getMessage();
5     }
6 }
7 // Utilisation de la classe
8 $a=100; $b=3;
9 try {
10    if($b == 0) {throw new MonException("Division par 0",7);}
11    elseif($a%$b != 0) {throw new MonException("Quotient entier impossible",55);}
12    else echo "Résultat de : $a / $b = ",$a/$b;
13 } catch(MonException $except) {
14    echo $except->alerte();
15 } finally {
16    echo "Le script continue sans problème<br/>";
17 }
18 echo "FIN";
```

CM PHP : Fonctions

Les fonctions natives de PHP

Vérifier la disponibilité de modules/fonctions (eg. chez votre hébergeur)

Liste des modules installés

- `get_loaded_extensions () :array`

Liste des fonctions disponibles dans un module d'extension

- `get_extensions_funcs ("nom_module") :array`

Tester l'existence d'une fonction

- `function_exists ("nom_fonction") :bool`

function-availability.php

```
1 //Tableau contenant le nom des extensions
2 $tabext = get_loaded_extensions (); natcasesort ($tabext) ;
3 foreach ($tabext as $cle=>$valeur) {
4     echo "<h3>Extension $valeur </h3> ";
5 //Tableau contenant le nom des fonctions
6 $fonct = get_extension_funcs ($valeur) ; sort ($fonct) ;
7 for ($i=0 ; $i<count ($fonct) ; $i++) {
8     echo $fonct [$i] , " " ;
9     echo "<hr />" ;
10 }
```

Définir ses fonctions

- Avec ou sans paramètres.
- Avec un nombre fixe ou variable de paramètres.
- Avec paramètres par défaut ou non.
- Avec ou sans valeur de retour.
- Avec passage par référence ou non.
- Avec retour par référence ou non.
- Avec variables statiques ou non.
- Avec itération sur valeurs de retour (générateur) ou non.
- Pour rappel (callback) ou non.
- Nommée ou anonyme, dynamique ou non.
- Avec capture de l'environnement lexical (fermeture) ou non.
- Typée fortement ou faiblement.

Définition de fonction

Déclaration=Définition en PHP

- Pas de déclaration séparément de la définition.
- Peut être définie n'importe où dans un fichier.

En-tête

- Mot-clé `function`.
- Nommage : mêmes règles que pour les variables.
- Paramètres : liste de variables.

Appel

- Peut être appelée avant sa définition dans le script.
- Peut être appelée sans arguments même si elle en attend (avertissement émis) !

Fonctions sans valeur retour

function-no-return-value1.php

```
1 // Fonction sans argument
2 function ladate() {
3 echo "<table><tr><td> ";
4 echo date("\l\le d/m/Y \i\l \\e\s\\t H:i:s");
5 echo "</td></tr></table><hr />";
6 }
7 // Fonction avec un argument
8 function ladate2($a) {
9 echo "<table><tr><td>";
10 echo "$a ",date("\l\le d/m/Y \i\l \\e\s\\t H:i:s");
11 echo "</td></tr></table><hr />";
12 }
13 // Appels des fonctions
14 echo ladate();
15 echo ladate2("Bonjour");
16 echo ladate2("Salut");
17 //echo ladate2();// erreur fatale
```

Fonctions sans valeur retour

function-no-return-value2.php

```
1 // Définition de la fonction
2 function tabuni ($tab, $bord, $lib1, $lib2)
3 {
4 echo "<table border=\"$bord\" width=\"100%\"><tbody><tr><th>$lib1</th> <th>$lib2
5 </th></tr>";
6 foreach ($tab as $cle=>$valeur)
7 {
8     echo "<tr><td>$cle</td> <td>$valeur </td></tr>";
9 }
10 echo "</tbody> </table><br />";
11 // Définition des tableaux
12 $tab1 = array ("France"=>"Paris", "Allemagne"=>"Berlin", "Espagne"=>"Madrid");
13 $tab2 = array ("Poisson"=>"Requin", "Cétacé"=>"Dauphin", "Oiseau"=>"Aigle");
14 // Appels de la fonction
15 tabuni ($tab1, 1, "Pays", "Capitale");
16 tabuni ($tab2, 6, "Genre", "Espèce");
```

Typage des fonctions/méthodes

Pour déclarer paramètres et valeur de retour

- int, float, string, bool, array.
- callable : fonctions de rappel.
- Nom de classe ou interface.
 - self : Réfère à la classe définissant la méthode.

Valeur de retour inutile ou aucun paramètre en entrée : void.

Typage faible vs. typage fort (alias typage strict)

Choix du typage fort en incluant en début de fichier la directive

```
declare(strict_types=1);
```

- Exception `TypeError` levée en cas d'erreur de type à l'appel de fonctions.

Transtypage automatique en cas de typage faible.

Typage faible et typage fort

function-typing-weak.php

```
1 function sum(int $a, int $b) : int
2 { return $a + $b; }
3 echo sum(1, 2);
4 // conversion automatique de float vers int
5 echo '\n'.sum(1.5, 2.5);
```

function-typing-strict.php

```
1 declare(strict_types=1);
2 function sum(int $a, int $b) : int { return $a + $b; }
3 try {
4     echo sum(1.5, 2.5);
5 } catch (TypeError $e) {
6     echo 'Erreur : ' . $e->getMessage();
7 }
```

Retourner plusieurs valeurs avec un tableau

function-return-array.php

```
1 function modarg ($reel, $imag)
2 {
3     $mod = sqrt ($reel * $reel + $imag * $imag); // ou
$mod=hypot($reel,$imag);
4     $arg = atan2 ($imag, $reel);
5     return array (
6         "module" => $mod,
7         "argument" => $arg
8     );
9 }
10 // Appels de la fonction
11 $a = 5;
12 $b = 8;
13 $complex = modarg ($a, $b);
14 echo "<b>Nombre complexe $a + $b i:</b><br />
15     module = ". $complex ["module"], "<br />
16     argument = " . $complex ["argument"], " radians<br />";
```

Paramètres par défaut

Passage de paramètres par défaut

Les paramètres “les plus à droite” peuvent avoir des valeurs par défaut.

function-default.php

```
1 function f ($a, $b=2) { return $a*$b; }
2 echo f(10);
```

Fonctions variadiques : informations sur les paramètres

Nombre d'arguments passés à une fonction

- `func_num_args() :int`

Accès à paramètre individuel par position (0 pour le 1er)

- `func_get_arg(int) :mixed`

Tableau indicé des paramètres

- `func_get_args() :array`

`function-variadic1.php`

L'opérateur ...

Pour définir des fonctions variadiques

Syntaxe : `f ($x, ...$tab);`

- `$tab` traité comme un tableau comportant un nombre quelconque de paramètres supplémentaires.
- Lecture classique de `$tab` avec boucle.

Différents appels possibles

- Une seule valeur liée à `$x` : `f (1)`
- Une liste de paramètres, le premier lié à `$x`, les autres à `$tab` :
`f (1, 3, 4)`
- Un tableau lié à `$tab` : `f (... [1, 3, 4])`
- Une valeur et un tableau liés respectivement à `$x` et `$tab` :
`f (1, ... [3, 4])`

L'opérateur ...

function-variadic2.php

```
1 function prod($a,...$tab) {
2     foreach ($tab as $nb) { $a *=$nb; }
3     echo " Le produit vaut ";
4     return $a;
5 }
6 echo "Produit 1",prod(2*3*4*5), "<br/>";//120
7 echo "Produit 2",prod(2,3,4,5), "<br/>";//120
8 $tab1= range(2,5);
9 echo "Produit 3",prod(...$tab1), "<br/>";//120
10 $tab2 = array(3,4,5);
11 echo "Produit 4",prod(2,...$tab2), "<br/>";//120
```

Les générateurs

Fonction retournant plusieurs valeurs à la demande

- Alternative efficace à la création et au renvoi de tableau.
- Crée et renvoie un itérateur (objet de classe Generator).
- Le code appelant peut itérer sur les valeurs générées avec `foreach`.
- Syntaxe avec mot-clé `yield`:
 - `yield $value;` pour générer des valeurs sans clés.
 - `yield $key=>$value;` pour générer des valeurs avec clés.

function-generator1.php

```
1 function suite($min, $max, $pas) {  
2     for ($i=$min; $i<=$max; $i+=$pas) { yield $i; }  
3 }  
4 echo "Suite : ";  
5 foreach (suite(0, 50, 10) as $nb) { echo $nb, " /// "; }
```

Les générateurs

Avec instruction `return`

Valeur de retour accessible par invocation de la méthode
`getReturn()` sur le générateur.

function-generator2.php

```
1 function suite($min,$max,$pas) {  
2     $total=0;  
3     for ($i=$min;$i<=$max;$i+=$pas) {  
4         yield $i;  
5         $total+=$i;  
6     }  
7     return $total;  
8 }  
9 echo "Suite : ";  
10 $gener = suite(0,45,5);  
11 foreach ($gener as $nb) {  
12     echo $nb, " ";  
13 }  
14 echo "<br/> Total = ",$gener->getReturn();
```

Délégation de générateurs

Appel de générateur à générateur avec `yield from`

function-generator3.php

```
1 function nom($prenom, $nom) {
2     yield $prenom;
3     yield $nom;
4     yield from adresse();
5     yield from code();
6 }
7 function adresse() {
8     yield "Paris"; yield "France";
9 }
10 function code() {
11     yield "75006";
12 }
13 foreach (nom("Jan", "Geelsen") as $coord) {
14     echo $coord, "\n";
15 }
```

Portée des variables

Déterminée selon le contexte

Portée globale pour toute variable déclarée en dehors d'une fonction ou d'une classe (portée limitée au script et aux scripts inclus).

function-variable-global-scope-include.php

```
1 echo $a;
```

function-variable-global-scope.php

```
1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4 <meta charset="utf-8" />
5 </head>
6 <body>
7 $a = 1;
8 include 'variable-scope-include.php';
9 <div></div>
10 echo $a;
11 </body>
12 </html>
```

Portée des variables

Au sein des fonctions

Portée locale pour toute variable définie dans une fonction ou une classe SAUF si redéclarée avec `global` ou utilisée avec `$GLOBALS[]`.

function-variable-local-scope.php

```
1 $a = $b = $c = 1;
2 function test() {
3     global $b;
4     echo $b;
5     echo $GLOBALS['c'];
6     echo $a; // undefined variable $a
7 }
8 test();
```

Variables statiques

Variable statique

- Variable locale déclarée avec `static` dans une fonction.
- Garde sa valeur en sortie d'appel à la fonction pendant la durée du script.

function-variable-static.php

```
1 function test() {  
2     static $a = 0;  
3     echo $a;  
4     $a++;  
5 }  
6 test(); //0  
7 test(); //1 ...
```

Passage par référence

Référence en PHP

- Semblable à la notion de lien sur fichier en LINUX.
- Différent de la notion de pointeur en C (pas d'arithmétique sur références ...).

Trois usages

- Affectation par référence : opérateur `=&`
- Passage par référence : fonction `f(&$a)`
- Retour par référence : fonction `&f();` `$a= &f();`

Affectation par référence - Destruction de référence

function-reference1.php

```
1 // affectation par référence
2 $b = 10;
3 $a = & $b;
4 echo "\$a=". $a; //10
5 $a += 1;
6 echo "\n\$b=". $b; //11
```

function-reference2.php

```
1 // destruction de référence
2 $b = "b";
3 $c = "c";
4 $a = & $b;
5 echo "\n\$a=". $a; // b
6 $a = & $c;
7 echo "\n\$a=". $a; // c
8 echo "\n\$b=". $b; // b
9 unset ($a);
10 echo "\n\$a=". $a; // Undefined
11 echo "\n\$c=". $c; // c
```

Fonctions : passage et retour par référence

function-reference3.php

```
1 // passage par référence
2 function foo (&$var) { $var++; }
3 function bar ($var) { $var++; }
4 foo ($a);
5 echo "\n\$a=". $a;
6 bar ($a);
7 echo "\n\$a=". $a;
```

function-reference4.php

```
1 // retour par référence
2 function &collector () {
3     static $collection = array ();
4     print_r ($collection);
5     return $collection;
6 }
7 // !! préfixer l'appel avec &
8 $collect = &collector (); //Array()
9 $collect [] = 'foo';
10 collector (); //Array([0]=>foo)
```

Fonctions dynamiques

Nom de fonction déterminé par une variable `string`

Cf. variables dynamiques.

function-dynamic.php

```
1 $f1 = "function_exists";
2 $f2 = "date";
3 echo $f1 ($f1); // équivaut à function_exists("function_exists");
4 echo "\n";
5 echo $f2 ("d/M/Y"); // équivaut à date("d/M/Y");
6 // Bloc équivalent à ce qui précède
7 $tf = [ "function_exists", "date" ];
8 echo $tf[0] ($tf[0]);
9 echo "\n";
10 echo $tf[1] ("d/M/Y");
```

Fonctions de rappel

Fonction de rappel (alias callback)

- Fonction nommée ou anonyme, méthode d'objet ou méthode statique de classe.
- Passée comme argument à d'autres fonctions qui l'appelleront
 - Par la chaîne la dénommant.
 - Par une fonction anonyme (définition en ligne).
 - Par une variable (objet de classe interne `Closure`) qui la dénote : on parle alors de fermeture (alias closure).
- Pseudo-type `callable`.

Fonctions de rappel

function-callback-named.php

```
1 $tab=array(1,2,3);
2 // fonction de rappel nommée
3 function aucarre($n) { return $n*$n; }
4 $tab1=array_map("aucarre",$tab);
5 print_r($tab1);
```

function-callback-anonymous.php

```
1 $tab=array(1,2,3);
2 // fonction de rappel anonyme
3 $tab1=array_map(function ($n) { return $n*$n; },$tab);
4 print_r($tab1);
```

function-callback-closure.php

```
1 $tab=array(1,2,3);
2 // fermeture (objet Closure)
3 $aucarre = function ($n) { return $n*$n; };
4 echo $aucarre(10);
5 $tab1=array_map($aucarre,$tab);
6 print_r($tab1);
```

Application : tri personnalisé sur tableaux indicés

Sur la base d'une fonction binaire numérique à définir (le critère de tri) moncritere (mixed \$x, mixed \$y) : int doit renvoyer :

- -1 (ou nombre négatif) si \$x est “inférieur” à \$y.
- 0 si \$x est “égal/équivalent” à \$y.
- 1 (ou nombre positif) si \$x est “supérieur” à \$y.

Fonction utilisée comme argument de usort (array &\$tab, callable \$f) :bool

```
usort ($tab, "moncritere");
```

- Trie \$tab selon la fonction moncritere().
- Perte des correspondances indices-valeurs.

Exemple

array-usort.php

```
1 // Tableau à trier
2 $tab = ["Blanc", "Jaune", "rouge", "Vert", "Orange", "Noir", "Emeraude"];
3 echo "Tableau initial<br />"; print_r($tab);
4 // Un comparateur
5 function shorter($s1, $s2) {
6     return strlen($s1) <= > strlen($s2); // -1ssi <, 0ssi ==, 1ssi >
7     // ou bien return strlen($s1) - strlen($s2);
8 }
9 // Tri avec comparateur 'shorter'
10 usort($tab, "shorter");
11 echo "<br />Mots triés par longueur décroissante<br />";
12 print_r($tab);
13 // Alternative : fonction anonyme
14 usort($tab, function($s1, $s2) { return strlen($s1) <= > strlen($s2); });
15 // Exemple avec fermeture : comparateur des k+1-èmes caractères
16 function caractere($k) {
17     return function($s1, $s2) use ($k) { return $s1[$k] <= > $s2[$k]; };
18 }
19 usort($tab, caractere(2));
20 echo "<br />Mots triés selon l'ordre ASCII sur leurs 3èmes caractères<br />";
21 print_r($tab);
```

Tri sur tableaux associatifs

Tri des valeurs Avec préservation des correspondances clés-valeurs

- `uasort (array $tab, callable $f) :bool` : tri personnalisé des valeurs avec \$f.

Tri des clés avec préservation des correspondances clés-valeurs

- `uksort (array $tab, callable $f) :bool` : tri personnalisé des clés avec \$f.

array-ksort-krsort-uksort.php

Tableau associatif d'origine

Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des clés

Array ([bla] => Noir [blu] => Bleu [g] => Vert [r] => rouge [w10] => Blanc10 [w2] => Blanc2 [y] => Jaune)

Tri en ordre ASCII inverse des clés

Array ([y] => Jaune [w2] => Blanc2 [w10] => Blanc10 [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir)

Tri selon la longueur des clés

Array ([w10] => Blanc10 [bla] => Bleu [blu] => Noir [w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert)

Filtrage de tableaux

`array_filter($tab, callable $f):array`

Retourne le tableau d'éléments de `$tab` acceptés par la *call-back* `$f`

- `f(mixed $var):bool` est appelée avec chaque élément `$var` de `$tab` et doit retourner `TRUE` ssi elle accepte l'élément.
- Préserve les correspondances clés-valeurs pour les éléments retenus.

array-filter.php

```
1 //Définition du tableau
2 $villes=array("Paimpol", "Angers", "Pau", "Nantes", "Lille");
3 //Fonction de sélection
4 function init($ville) {
5     if ($ville[0]=="P" || $ville[0]=="p") return $ville;
6 }
7 //Utilisation de array_filter()
8 $select=array_filter($villes, "init"); print_r($select);
```

Transformation de tableaux

```
array_walk(array &$tab, callable $f [, mixed  
$userdata]):bool
```

Applique la callback `$f` à chaque élément de `$tab`.

- `f(mixed $v, mixed $c):bool` prend comme arguments la valeur et la clé de chaque élément. Accepte `$userdata` comme 3ème argument si défini.
- Ne peut modifier les valeurs de `$tab` qu'à la condition de spécifier `$v` comme référence :

```
f(mixed &$v, mixed $c):bool
```

array-walk.php

Fermeture et environnement lexical

Une fermeture est une fonction accompagnée de son environnement lexical

- On crée souvent une fermeture au sein d'une fonction/méthode qui la construit et la renvoie en lui donnant accès à certaines variables.
- L'environnement lexical d'une fermeture est l'ensemble de ces variables non-locales qui sont liées/héritées par valeur et/ou par référence.

En PHP, on doit expliciter les variables de l'environnement lexical et le type de liaison dans la déclaration de la fermeture

- Par valeur avec la syntaxe `use ($x)`
- Par référence avec la syntaxe `use (&$x)`

Fermetures

function-closure.php

```
1 $global = "<br/>";
2 function role(&$role) {
3     global $global;
4     $local = " !";
5     // liaison par valeur
6     return function ($nom) use ($global, $role, $local) {
7         echo "$nom est $role $local $global";
8         $role = "Mousquetaire<br/>"; // sans effet global
9     };
10 }
11 $role = "cardinal";
12 $cardinal = role($role);
13 $cardinal("Richelieu"); // Richelieu est cardinal
14 $cardinal("Mazarin"); // Mazarin est cardinal
15 $role = "mousquetaire";
16 $cardinal("Richelieu"); // Richelieu est [toujours] cardinal
17 function newrole(&$role) {
18     // liaison par référence
19     return function ($nom) use (&$role) {
20         echo "$nom est $role<br/>";
21         $role = "Cardinal"; // effet global
22     };
23 }
24 $musketeer = newrole($role);
25 $musketeer("D'Artagnan"); // D'Artagnan est mousquetaire
26 $musketeer("D'Artagnan"); // D'Artagnan est [devenu] cardinal
27 echo "Global - \$role=$role";
```

Objets et fermetures

Lier l'environnement lexical d'une fermeture à un objet

- Par appel à la méthode `call()` de la fermeture (objet Closure).

function-closure-call.php

```
1 class Nombre { public $v=10; }
2 // fermeture liée par référence à $text et à l'objet $this par Closure::call::bindTo
3 $clos1 = function ($a) use (&$text) { return $text.((($this->v *= 2) +
$a); };
4 $text = "Addition = ";
5 $nb = new Nombre();
6 // Appel à fermeture avec liaison temporaire à objet $nb
7 echo $clos1->call ($nb, 5); //Addition = 25 (2*10+5)
8 // $clos1(5); // erreur
9 $text = "Somme = "; $nb->v=20;
10 // Nouvel appel
11 echo $clos1->call ($nb, 7); //Somme = 47 (2*20+7)
12 // Liaison permanente
13 $clos2 = $clos1->bindTo (new Nombre);
14 echo $clos2 (6); //Somme = 26
15 // Interdit : $x=new Closure();
```

CM PHP : Formulaires

Rappel sur les formulaires HTML5

Les formulaires

- Base de l'interactivité des sites Web.
- Echanges avec serveur par le protocole HTTP en utilisant méthode GET ou, de préférence, POST.
 - Envoi de données.
 - Déclenchement d'une requête dans une BDD.
 - Création de page dynamique en réponse.

Composants de formulaires

- Saisie de texte et mots de passe.
- Choix par boutons radio, cases à cocher, listes de sélection.
- Envoi de fichiers ou données cachées.

Création de formulaire

Attributs de l'élément <form>

action="f.php"

- Désigne le fichier de traitement des données saisies.
- Chemin absolu ou relatif (de préférence), ou URL.
- action="<?= \$_SERVER ["PHP_SELF"] ?>" si le fichier de traitement contient le formulaire.
- Possibilité d'utiliser d'autres protocoles (mailto:a@b.c).

Création de formulaire

Attributs de l'élément <form>

method="get" ou method="post"

- Méthode HTTP d'envoi des données.
- HTTP GET par défaut.
 - Données en clair dans l'URI : le *query-string*
 - http://www.abc.fr/f.php?x=bleu&y=12
 - Ajouté à l'URL avec ?
 - Affectations de champs séparées par &
 - Caractères accentués, symboles de ponctuation ... encodés en hexadécimal.
 - Limitée à ~2000 caractères.
- HTTP POST recommandée.
 - Encapsule les données dans le corps de la requête.
 - Données de tout type.

Création de formulaire

Attributs de l'élément <form>

name="monformulaire"

- Utile essentiellement pour JavaScript.

enctype="type-encodage"

- "multipart/form-data" pour le transfert de fichiers.
- "application/x-www-form-urlencoded" par défaut.

```
<form method="post" action="f.php"  
enctype="application/x-www-form-urlencoded">  
...  
</form>
```

L'élément <input />

L'attribut `type` permet de distinguer différents types de champs

- Texte en clair et mot de passe.
- Email et numéro de téléphone.
- Nombre et date.
- Choix unique et choix multiple.
- Soumission et réinitialisation.
- Fichier et données cachées.

L'attribut `name` est obligatoire

- Identifie chaque champ côté serveur.
- Ne pas confondre avec l'attribut `id` utilisé pour CSS et manipulation JS via le DOM.

Champ texte mono-ligne en clair

Attributs de l'élément `<input type="text"/>`

`size="nombre"`

- Largeur de la zone affichée en #caractères.

`maxlength="nombre"`

- Nombre maximum de caractères à saisir.

`value="texte"`

- Texte affiché et transmis par défaut.

```
<input type="text" name="ville" size="30"  
maxlength="40" value="Angers"/>
```

Ergonomie : effacement au clic de la valeur par défaut via JS

```
<input ... onclick="this.value=''" />
```

Autres champs texte

Adresse e-mail : <input type="email"/>

N° de téléphone : <input type="tel">

Validation par regex avec l'attribut pattern.

Mots de passe : <input type="password"/>

Caractères saisis affichés par des *

Nombres : <input type="number"/>

Encadrement forcé avec attributs min et max.

Obligation de saisie avec required.

Dates : <input type="date"/>

Champ texte au format AAA-MM-JJ avec interface selon navigateur.

Encadrement avec attributs min et max.

Variantes : time (heure), month, week, datetime-local (date et heure), datetime (date, heure et fuseau horaire).

Boutons radio

Attributs de l'élément `<input type="radio"/>`

`checked="checked"`

- Bouton coché par défaut.

`value`

- Indispensable comme pour champs texte.

Utiliser la même valeur d'attribut `name` entre boutons pour imposer un choix exclusif

```
<label>Homme</label><input type="radio" name="genre" value="h"/> <label>Femme</label><input type="radio" name="genre" value="f"/>
```

Cases à cocher

Elément `<input type="checkbox" />`

Utiliser la même valeur d'attribut `name` concaténée avec `[]` pour permettre un choix multiple :

```
<input type="checkbox" name="lang[]" value="php"/>
<input type="checkbox" name="lang[]" value="javascript"/>
```

Boutons de soumission et RAZ

Elément `<input type="submit"/>`

Indispensable d'avoir au moins un bouton de soumission. Attribut value

- Le texte affiché du bouton.
- Permet d'effectuer différentes tâches de traitement selon la valeur du bouton pressé.

Elément `<input type="reset"/>`

Réinitialise les champs à leurs valeurs par défaut (\neq effacement).

Fichiers et données masquées

Elément `<input type="file">`

Même apparence qu'un champ texte et incorpore un sélecteur de fichier sur le poste client.

Attribut `accept`

- Définit le ou les types de fichiers acceptés.

Attribut `multiple`

- Possibilité de sélectionner et téléverser plusieurs fichiers.

```
<input type="file" name="image" accept=".jpeg, .jpg, .png" multiple>
```

Elément `<input type="hidden">`

Permet d'envoyer des données invisibles pour l'utilisateur (e.g. jeton de sécurité, identifiant de transaction, taille maximum de fichier téléversable).

Zones de texte multilignes

Attributs de l'élément <textarea>

cols

- Nombre de colonnes de la zone de texte.

rows

- Nombre de lignes de la zone de texte.

Liste déroulante

Elément <select>

Sélection simple ou multiple d'options définies chacune par un sous-élément <option>.

Attribut size

- Nombre d'options visibles simultanément.

Attribut multiple="multiple"

- Autorise la sélection multiple (avec touche Ctrl).

Attributs de l'élément <option>

value

- La valeur transmise si l'option est sélectionnée.

selected="selected"

- Option présélectionnée par défaut.

Récupération des données de formulaire

Différents cas de figure :

- Les valeurs uniques.
- Les valeurs multiples.
- Les fichiers.
- Les boutons d'envoi multiples.

Récupération de valeurs uniques

Contenues dans le tableau superglobal `$_POST` (ou `$_GET`)

form-single-value.php

Vérification avec `isset()` sur champ texte et boutons radio.

Tous deux `NULL` initialement au chargement de la page.

Si champ non saisi et boutons non cochés à la soumission :

- La valeur du champ est la chaîne vide : vérification plus fine avec `empty()`.
- La valeur des boutons radio reste `NULL`.

Récupération de valeurs multiples

Exemples

La valeur de l'attribut `name` doit se terminer par `[]` :

- Cases à cocher de même attribut `name`.
- Liste de sélection multiple avec attribut `multiple`.

Récupération des valeurs via `$_POST` qui est alors un tableau bi-dimensionnel.

[form-multiple-values.html](#), [form-multiple-values.php](#)

Récupération de fichiers

```
<input type="file" name="f" .../>
```

Le téléchargement renomme le fichier et le place dans un répertoire tampon sur le serveur.

Récupération via tableau associatif `$_FILES["f"]`.

Eléments de `$_FILES["f"]` :

- "name" : le nom d'origine.
- "type" : le type du fichier.
- "size" : la taille du fichier.
- "tmp_name" : le nom du fichier sur le serveur.
- "error" : code d'erreur en cas d'échec du téléchargement.

`move_uploaded_file(string src, string dest)` permet de renommer et déplacer le fichier.

form-file.php

Maintien de l'état du formulaire

Motivations

Ne pas avoir à faire ressaisir l'intégralité du formulaire en cas de saisie invalide.

- Si le script de traitement contient le formulaire, ce dernier sera réaffiché dans son état initial.

Approche 1 (form-state.php)

Intégrer dans un champ texte la dernière valeur saisie :

```
...<?php if(isset($_POST["k"])) echo $_POST["k"] ?>...
```

Intégrer dans chaque bouton coché l'attribut checked :

```
...<?php if(isset($_POST["k"]) && $_POST["k"]=="v") echo  
"checked =\"checked\\"" ?>...
```

Maintien de l'état du formulaire

Approche 2 (form-multiple-values.html, form-multiple-values.php)

- Un fichier HTML contient le formulaire dont l'action pointe sur un fichier PHP.
- Le fichier PHP redirige vers le fichier HTML par JavaScript.

```
echo  
"<script>alert(...);window.history.back();</script>";
```

Approche 3 (form-form.html / form-post-redirect.php / form-post.php)

- Un fichier HTML contient le formulaire dont l'action pointe sur un fichier PHP.
- Le fichier PHP redirige vers un second fichier PHP avec la méthode `header` : redirection temporaire HTTP avec transfert des valeurs de `$_POST` (code 307).

```
header("Location: $url", true, 307);
```

CM PHP : La couche Objet

Développement Web et programmation orientée objet

L'adoption de la programmation orientée objet (POO) est une tendance lourde en développement web et notamment en PHP

- Intérêts du paradigme pour les gros projets logiciels : modularisation, réutilisation, ...
- Les applications Web intègrent de multiples patrons de conception.
- Les cadriels PHP sont orientés objet.
- De nombreuses extensions PHP (ne) proposent (que) des interfaces de programmation d'application (API) objet.

La couche Objet en PHP

Proche de Java (refonte totale en PHP 5)

- Accessibilité publique, protégée ou privée des propriétés et méthodes.
- Propriétés et méthodes statiques.
- Chaînage d'invocation.
- Manipulation via handles et gestion mémoire automatique.
- Héritage simple avec polymorphisme d'inclusion (overriding) et liaison tardive.
- Méthodes et classes finales.
- Traits, classes abstraites, interfaces.
- Classes anonymes.
- Autres "aspects" : clonage, sérialisation, affichage ...

La couche Objet en PHP

Spécificités

- Pas de surcharge (overloading) de méthodes/fonctions (méthodes/fonctions de mêmes noms mais d'arguments différents).
- Méthodes magiques.
- API réflexion

Définition de classe

Recommandation

Définir chaque classe dans un fichier dédié à inclure par les scripts y faisant appel.

Création de classe

- Avec mot-clé `class`.
- Constantes, variables et méthodes déclarées avec la syntaxe usuelle.
- Valeurs par défaut autorisées pour les variables.
- Typage des méthodes autorisé.
- Spécificateur d'accès optionnel (`public` par défaut).

Création de classe

Cours.php

```
1 class Cours {
2     const ANNEE = 2021;
3     private static $types = ["CM" => "Cours magistral", "TP" => "Travaux pratiques"];
4     public static $nbCours = 0;
5     private $intitulé;
6     protected $type;
7     public $heures = 16;
8     public function __construct(string $i, string $t) {
9         $this->intitulé = $i;
10        $this->type = self::$types[$t];
11        ++ self::$nbCours;
12    }
13    public function __destruct() { echo "<script>alert(\".-- self::$nbCours.\")</script>"; }
14    public static function getTypesKeys() : array { return array_keys(self::$types); }
15    public function getIntitulé() : string { return $this->intitulé; }
16    protected function getType() : string {
17        global $EAD;
18        return $this->type . ($EAD ? " (EAD)" : "");
19    }
20    private function image() : string {
21        $image = base64_encode(file_get_contents("https://picsum.photos/200"));
22        return '';
23    }
24    public function info() : void {
25        echo "<p>Cours::ANNEE ( " . self::$nbCours . " cours) :
26                ($this->intitulé) - ($this->type) - ($this->heures) h</p>
27                ($this->image())<br/>";
28    }
29}
```

Construction et utilisation d'objets

- Instanciation avec mot-clé `new` suivi du nom de la classe (littéral ou variable `string`).
- Accès en lecture/écriture aux propriétés et invocation de méthodes avec mot-clé `$this` et notation fléchée `->` (sans symbole `$` pour les variables).

object-construct.php

```
1 require ("Cours.php");
2 $c1= new Cours("Constraint Programming","TP");
3 $c1->heures = 24;
4 $c1->info();
5 $EAD = true;
6 $c2= new Cours("Quantum Computing", "CM");
7 $c2->heures = 4;
8 $c2->info();
9 echo "<br/>Structure de l'objet \${$c2} est :<br/>";
10 var_dump($c2);
11 echo "<br/>Propriétés de l'objet \${$c2}<br/>";
12 array_walk($c2,function ($v,$p) { echo "Propriété \$p = \$v<br/>"; });
13 if($c2 instanceof Cours) {
14     echo "<br/>L'objet \${$c2} est du type Cours<br/>";
15 }
```

Accès aux variables dans les méthodes

- Avec `$this->` pour les variables propres.
- Littéralement pour les superglobales ou les globales déclarées avec `global`.

object-variable-access.php

```
1  public function __construct ($i, $t) {
2      $this->intitulé = $i;
3      $this->type = self::$types[$t];
4      ++ self::$nbCours;
5  }
6  public function getType () string {
7      global $EAD; // inclusion d'une globale dans la portée
8      return $this->type . ($EAD ? " (EAD)" : "");
9  }
```

Spécificateurs d'accessibilité

Trois niveaux d'accès aux propriétés

- `public` : accès universel (par défaut).
- `protected` : réservé aux instances de la classe et ses dérivées.
- `private` : réservé aux instances de la classe.

object-property-access.php

```
1 require ("Cours.php");
2 $c= new Cours("Cadriel Laravel", "TP");
3 echo COURS::ANNEE . "<br />"; // constante publique
4 //echo $c->intitulé; // Erreur fatale
5 //echo $c->type; // Erreur fatale
6 echo $c->heures; // propriété publique
```

Spécificateurs d'accessibilité

Trois niveaux d'accès aux méthodes

- `public` : accès universel (par défaut).
- `protected` : réservé aux instances de la classe et ses dérivées.
- `private` : réservé aux instances de la classe.

object-method-access.php

```
1 require ("Cours.php");
2 $c= new Cours("Cadriel Slim", "TP"); // appel au constructeur public
3 unset ($c); // appel au destructeur public
4 $c= new Cours("Cadriel CakePHP", "TP");
5 echo $c->getIntitulé(); // appel à méthode publique
6 //echo $c->getType(); // Erreur fatale
7 //echo $c->image(); // Erreur fatale
```

Propriétés et méthodes statiques

- Déclaration avec le mot-clé static.
- Pour une classe C de propriété p et méthode m :
 - Accès avec C:::\$p et C:::m() en dehors de la classe.
 - Accès avec self:::\$p et self:::m() au sein de la classe C.

object-static-access.php

```
1 require ("Cours.php");
2 //echo COURS::$types . "<br/>"; // Erreur fatale
3 echo COURS::$nbCours . " <br/>"; // propriété statique publique
4 $typeKeys = COURS::getTypesKeys (); // méthode statique publique
5 ($new Cours ("Cadriel Symfony", $typeKeys [0]))->info (); // méthode
publique
```

Méthodes magiques

Méthodes appelées automatiquement selon le contexte

- On peut implémenter, au choix, chaque méthode magique dans une classe.

Usages

- Initialiser un objet “correctement” à la construction.
- Libérer des ressources à la destruction.
- Contrôler la copie d’objet (eg. copie profonde).
- Contrôler l’accès à des membres privés (!)
- Interdire l’ajout de nouvelles propriétés.
- Contrôler sérialisation et affichage.
- ...

Méthodes magiques

Méthodes appelées automatiquement selon le contexte

<code>__construct</code>	Instantiation avec <code>new</code> .
<code>__destruct</code>	Destruction de la dernière référence (p. ex. avec <code>unset</code>).
<code>__clone</code>	Clonage avec <code>clone</code> .
<code>__get</code>	Accès en lecture à propriété inaccessible.
<code>__set</code>	Accès en écriture à propriété inaccessible.
<code>__isset</code>	Test de propriété inaccessible avec <code>isset()</code> ou <code>empty()</code> .
<code>__unset</code>	Suppression de propriété inaccessible avec <code>unset()</code> .
<code>__call</code>	Appel à méthode inaccessible.
<code>__callStatic</code>	Appel à méthode statique inaccessible.
<code>__sleep</code>	Appel avant sérialisation de l'objet avec <code>serialize</code> .
<code>__wakeup</code>	Appel après récupération d'un objet sérialisé avec <code>unserialize</code> .
<code>__toString</code>	Appel en contexte <code>string</code> (p. ex. avec <code>echo</code>).

Membre *inaccessible* = non déclaré ou non visible (`protected` ou `private`) dans le contexte

Construction d'objets

Construction d'instance de classe avec `new`

- Par appel automatique de la méthode `__construct(...)`
 - Si elle n'est pas définie, l'objet sera construit mais ses membres seront non initialisés.
 - Si on la définit, on ne fournit qu'une seule définition (pas de surcharge/overloading)
- NB. Certaines classes natives ont un constructeur privé (eg. Closure)

object-construct1.php

```
1 class A {  
2     private $p=1;  
3 }  
4 $a = new A();  
5 var_dump($a);  
6 //object(A)#1 (1) { ["p":"A":private]=> int(1) }
```

object-construct2.php

```
1 class B {  
2     public $p;  
3     function __construct(string $p) {  
4         $this->p = $p;  
5     }  
6 }  
7 $b = new B("e=mc2");  
8 echo $b->p; // ec=mc2
```

Objets, handles et références

Les objets sont gérés et manipulés via des “handles”

- Une variable de type objet est un handle pointant sur un objet (physique).
- Différents handles peuvent pointer sur un même objet et un handle peut être réaffecté.
- PHP gère l’allocation et la libération mémoire (garbage collection) en comptabilisant les handles restant sur un objet.
- Les objets sont affectés, passés aux fonctions/méthodes ou retournés par des fonctions/méthodes par handles (pas de clonage).
- Le passage par référence est toujours possible.

Objets, handles et références

object-assignment.php

```
1 class Alliage
2 {
3     public $nom;
4     function __construct(string $n) { $this->nom = $n; }
5     function __toString() { return $this->nom."<br/>"; }
6 }
7 $alliage1 = new Alliage("fonte");
8 $alliage2 = $alliage1; echo $alliage2; //fonte
9 function aciére(Alliage $a) : void { $a->nom = "acier"; }
10 aciére($alliage2); echo $alliage1; //acier
11 $alliage3 = new Alliage("fonte");
12 (function ($x,$y) { $tmp=$y;$y=$x;$x=$tmp; })($alliage1,$alliage3);
13 echo $alliage1, $alliage3; //pas d'échange
14 (function (&$x,&$y) { $tmp=$y;$y=$x;$x=$tmp; })($alliage1,$alliage3);
15 echo $alliage1, $alliage3; //échange
```

Destruction d'objets

Destruction d'instance de classe

- Par appel automatique de la méthode `__destruct()` en fin de script ou lors de la perte du dernier handle sur l'objet (eg. réaffectation de la variable correspondante).
- Possibilité de redéfinir `__destruct()`, eg. pour libérer des ressources (fermeture de fichier, déconnexion à un SGBD).

object-destruct.php

```
1 class Etudiant {  
2     private $nom;  
3     function __construct($nom) { $this->nom = $nom; }  
4     function __destruct() { echo "L'étudiant $this->nom n'existe plus!<br />"; }  
5 }  
6 $e1 = new Etudiant ("E1"); // handle $e1 sur objet physique E1  
7 $e1 = 1; // Appelle le destructeur "de" E1 via $e1  
8 $e2 = new Etudiant ("E2"); // handle $e2 sur objet physique E2  
9 $e3 = new Etudiant ("E3"); // handle $e3 sur objet physique E3  
10 $e3 = $e2; ; // Appelle le destructeur "de" E3 via $e3  
11 $e2 = 1; // !!! N'appelle pas le destructeur "de" E2 car reste le handle $e3  
12 $r3 =& $e3; // $r3 est une référence sur $e3  
13 $e3 = 1; // !!! Appelle le destructeur "de" E2 via $e3  
14 $e4 = new Etudiant ("E4");  
15 echo "Fin du script<br />".
```

Clonage d'objets

Copie superficielle d'objet (shallow copy) avec `clone`

- Par appel automatique de la méthode `__clone()`
- Les références internes de l'objet sont préservées.
- Possibilité de redéfinir `__clone()`, eg. pour copie profonde.

object-cloning.php

```
1 class Alliage {
2     public $nom;
3     function __construct(string $nom) { $this->nom=$nom; }
4     function __destruct() { echo "$this->nom a fondu<br/>"; }
5     function __clone() { $this->nom="clone de ".$this->nom; }
6 }
7 $bronze = new Alliage("Bronze");
8 $airain = $bronze;
9 $clone = clone $bronze;
10 $bronze->nom="alliage de cuivre et d'étain";
11 echo $bronze->nom , "<hr/>"; // alliage de cuivre et d'étain
12 echo $airain->nom , "<hr/>"; // alliage de cuivre et d'étain
13 echo $clone->nom , "<hr/>"; // clone de bronze
14 print_r($bronze); print_r($airain); print_r($clone); echo "<br/>";
```

Accesseurs et mutateurs magiques

Interception des (tentatives d') accès aux propriétés

- En lecture via `__get(...)`
- En écriture via `__set(...)`

object-get-set.php

```
1 class A {  
2     public $pub="propriété publique";  
3     private $priv;  
4     public function __construct($v) { $this->priv = $v; }  
5     public function __get($p) {  
6         return $this->$p;  
7     }  
8     public function __set($p, $v) {  
9         $this->$p = $v;  
10    }  
11    public function __unset($p) {  
12        unset($this->$p);  
13    }  
14 }  
15 $obj = new A('AZERTY');  
16 echo $obj->priv;//AZERTY  
17 $obj->priv="QWERTY"; echo $obj->priv;//QWERTY  
18 unset($obj->priv); echo $obj->priv;//PHP Notice: Undefined property: A::$priv  
19 echo $obj->pub; //__get non invoquée
```

Ajout dynamique de propriétés

Il est possible d'ajouter des propriétés après instantiation !

- A contrôler via `__set`

object-set.php

```
1 class Ville {  
2     private $nom;  
3  
4     function __construct ($nom) {  
5         $this->nom = $nom;  
6     }  
7     function __toString () {  
8         return $this->nom . '<br/>';  
9     }  
10    $stroie = new Ville ("Troie");  
11    var_dump ($stroie);  
12    $stroie->intrus = "Cheval de  
Troie";  
13    var_dump ($stroie);  
  
14 }  
15 $stroie = new Ville ("Troie");  
16 echo $stroie; // Troie  
17 $stroie->nom = "Ilios";  
18 echo $stroie; // Ilios  
19 $stroie->intrus = "Cheval de Troie";  
20 echo $stroie->intrus; // Notice: Undefined property:  
Ville::$intrus
```

object-member-addition.php

```
1 class Ville {  
2     private $nom;  
3     function __construct ($nom) {  
4         $this->nom = $nom;  
5     }  
6     function __toString () {  
7         return $this->nom . '<br/>';  
8     }  
9 }  
10 $stroie = new Ville ("Troie");  
11 var_dump ($stroie);  
12 $stroie->intrus = "Cheval de  
Troie";  
13 var_dump ($stroie);
```

Chaînage d'invocation de méthodes

Chaînage d'invocations de méthodes retournant des objets avec
\$x->m1 (. . .) ->m2 (. . .) -> . . .

object-method-chaining.php

```
1 class Varchar {
2     private $chaine;
3     function __construct ($a) {
4         $this->chaine= (string) $a;
5     }
6     function add(string $char) : Varchar {
7         $this->chaine .= $char;
8         return $this;
9     }
10    function getChaine() : string {
11        return $this->chaine;
12    }
13 }
14 $texte=new Varchar("Apache ");
15 echo $texte->getChaine(); //Apache
16 echo $texte->add("PHP 7 ")->getChaine(); //Apache PHP
17 echo $texte->add("MySQL ")->add("SQLite")->getChaine(); //Apache PHP MySQL SQLite
```

Héritage

Création de sous-classe avec mot-clé `extends`

Constructeur et destructeur parents appelés avec
`parent::__construct(...)` et `parent::__destruct()`

Référence aux membres de la classe parente en cas de redéfinition
(overriding) avec `parent::`

Héritage

Mere.php

```

1 class Mere {
2     private $pri;
3     protected $pro;
4     function __construct($pri,$pro) {
5         $this->pri = $pri;
6         $this->pro = $pro;
7     }
8     protected function get() {
9         return $this->pro
10        .'('
11        . $this->pri
12        .') ';
13    }
14 }
```

Fille.php

```

1 class Fille extends Mere {
2     private $pri;
3     public $pub;
4     function __construct($pri,$pro,$pub) {
5         parent::__construct($pri,$pro);
6         $this->pub = $pub;
7     }
8     public function getPro() {
9         return $this->pro;
10    }
11    public function get() {
12        return parent::get() . $this->pub;
13    }
14 }
```

object-inheritance.php

```

1 spl_autoload_register();
2 $f = new Fille("Byron","Ada","Lovelace");
3 echo $f->getPro(); //Ada
4 echo '<hr/>';
5 echo $f->get(); //Ada (Byron) Lovelace
```

Liaison tardive dynamique et statique (late binding)

Sélection de l'implémentation d'une méthode d'objet ou de classe

Liaison tardive statique avec mot-clé static.

object-late-static-binding.php

```
1 class Pere {
2     static public function info ($nom) {
3         static::affiche ($nom); // 'static' pour LSB
4     }
5     static public function affiche ($nom) {
6         echo "<h3>Je suis le père $nom </h3>";
7     }
8 }
9 class Fils extends Pere
10 {
11     static public function affiche ($nom) {
12         echo "<h3>Je suis le fils $nom </h3>";
13     }
14 }
15 Fils::info ('Luke');
```

Méthodes et classe finales, classes anonymes

Pour empêcher toute modification par héritage, préfixer définition de méthode/classe par le mot-clé `final`

Classe anonyme pour créer un objet “ponctuel” avec `new`
`class(...){...}`

object-anonymous-class.php

```
1 $client = new class {
2     private $nom;
3     public function setNom($nom) { $this->nom=$nom; }
4     public function getNom() { return $this->nom; }
5 };
6 var_dump($client);
7 $client->setNom("client");
8 echo $client->getNom();
```

Classe abstraite

Classe non-instanciable

- Se définit avec `abstract class nom-classe { ... }`
- Peut contenir des méthodes abstraites (sans implémentation).
- S'hérite avec possibilité de relâcher les spécificateurs d'accès.

object-abstract-class.php

```
1 abstract class Abstraite {
2     protected $hauteur;
3     abstract function surface();
4     protected function volume() { return $this->surface()*$this->hauteur; }
5 }
6 class Cylindre extends Abstraite {
7     private $rayon;
8     function __construct($r,$h) {
9         $this->hauteur = $h;
10        $this->rayon = $r;
11    }
12    public function surface() { return pi() * ($this->rayon)**2; }
13    public function volume() { return parent::volume($this->surface()); }
14 }
15 echo (new Cylindre(2,1))->volume();
```

Interface

Notion plus restrictive que les classes abstraites.

Une interface

- Ne contient aucune propriété.
- Ne définit que des méthodes abstraites publiques.
- Se définit avec `interface nom-interface { ... }`
- S'implémente avec `class nom-classe implements nom-interface { ... }`

Une classe peut implémenter plusieurs interfaces.

object-interface.php

Traits

Alternative flexible aux classes abstraites et pour remédier à l'impossibilité de l'héritage multiple.

Un trait

- Peut contenir propriétés et méthodes.
- Ne peut pas être instancié.
- Création avec `trait nom-trait { ... }`
- Peut être utilisé par n'importe quelle classe avec `use nom-trait;`
- Peut utiliser d'autres traits.

Une classe peut mixer plusieurs traits, tout en héritant d'une classe et en implémentant des interfaces . . .

Traits

object-trait.php

```
1 trait Marcher {
2     public $pattes;
3     function marche() { echo "Je marche sur ". $this->pattes." pattes<br/>"; }
4 }
5 trait Voler {
6     public $ailes;
7     function vole() { echo "Je vole avec ". $this->ailes." ailes <br/>"; }
8 }
9 trait Nager {
10    function nage() { echo "Moi je sais nager<br/>"; }
11 }
12 class Cheval { use Marcher,Nager; }
13 class Oiseau { use Marcher,Voler; }
14 class Pegase { use Marcher,Nager,Voler; }
15 // Un aigle
16 $aigle=new Oiseau(); $aigle->pattes=2; $aigle->ailes=2;
17 echo "<h3>L'aigle: </h3>"; $aigle->marche(); $aigle->vole();
18 // Un cheval
19 $dada=new Cheval(); $dada->pattes=4;
20 echo "<h3>Le cheval : </h3>";
21 $dada->marche(); $dada->nage();
22 // Pégase, le cheval ailé
23 $chevalaile=new Pegase(); $chevalaile->ailes=2; $chevalaile->pattes=4;
24 echo "<h3>Pégase : </h3>";
25 $chevalaile->marche(); $chevalaile->vole(); $chevalaile->nage();
```

Traits - Collision de noms

Collision de noms

Si classe enfant, trait et classe parente contiennent des méthodes homonymes, la 1ère a priorité sur la 2nde, elle-même prioritaire sur la 3ème.

Collision de noms

Si une classe utilise deux traits contenant des méthodes homonymes, résolution par

- Hiérarchisation avec `trait1::nom-commun insteadof trait2;`
- Renommage (aliasing) avec `nom-trait::nom-commun as nom-alias;`

Traits : Hiérarchisation et renommage

object-trait-conflict.php

```
1 trait UN {
2     public function small($text) { echo "<h6>trait UN : $text</h6>"; }
3     public function big($text) { echo "<h5>trait UN : $text</h5>"; }
4 }
5 trait DEUX {
6     public function small($text) { echo "<h2>trait DEUX : $text </h2>"; }
7     public function big($text) { echo "<h1>trait DEUX : $text </h1>"; }
8 }
9
10 class Texte {
11     use UN, DEUX {
12         DEUX::small insteadof UN;
13         UN::big insteadof DEUX;
14         DEUX::big as gros;
15         UN::small as petit;
16     }
17 }
18 $a=new Texte();
19 $a->small("Méthode small"); //trait DEUX : Méthode small
20 $a->big("Méthode big"); //trait UN : Méthode big
21 $a->gros("Méthode gros"); //trait DEUX : Méthode gros
22 $a->petit("Méthode petit"); //trait UN : Méthode petit
```

Test de type et d'héritage

Vérifier si une variable est l'instance d'une classe

- Avec mot-clé `instanceof`.

Vérifier si une variable est l'instance d'une sous-classe ou si une relation d'héritage existe entre classes

- Avec mot-clé `is_subclass_of`.

object-instanceof.php

```
1 class A {}
2 class B extends A {}
3 $p=new A();
4 $e=new B();
5 if ($p instanceof A)
6     echo var_dump($p), " is a A\n"; //object(A)#1 (0) {} is a A
7 if ($e instanceof A)
8     echo var_dump($e), " is a A\n"; //object(B)#2 (0) {} is a A
9 if(is_subclass_of($e,A))
10    echo var_dump($e), " instance of subclass of A\n";
11    //object(B)#2 (0) {} instance of subclass of A
12 if(is_subclass_of(B,A))
13    echo "B subclass of A\n"; //B subclass of A
```

CM PHP : Réflexion

Réflexion (Reflection)

Mécanisme pour examiner ou modifier programmatiquement (reverse engineering)

- Les classes et objets (propriétés et méthodes).
- Les interfaces.
- Les fonctions.
- Les extensions.
- Les commentaires.

PHP est un langage réflexif

Depuis PHP 5, on dispose d'une API (classes et interfaces) dédiée à la réflexion.

Réflexion : usages

- Le typage dynamique (duck typing).
- La programmation orientée aspect.
- La méta-programmation.
- Les frameworks Web : initialisation de modèles, création d'objets vues, injection de dépendances ...
- Les “mocking frameworks” (pour objets simulés), e.g., PHPUnit.
- Les frameworks d'analyse de code.

Réflexion procédurale

Examen du contenu d'un objet avec les fonctions suivantes

string get_class(o)	Renvoie le nom de la classe d'un objet.
string get_parent_class(o)	Renvoie le nom de la classe parente de l'objet.
array get_class_vars(o)	Renvoie les valeurs par défaut des attributs d'une classe.
array get_object_vars(o)	Renvoie les attributs non statiques de l'objet qui sont accessibles depuis le contexte courant.
array get_class_methods(o)	Renvoie les noms des méthodes d'une classe.

Autres fonctions

class_exists(), get_called_class(),
get_declared_interfaces(), method_exists() ...

get_object_vars

Ne donne accès qu'aux membres publics si invoquée en dehors de la classe.

get-object-vars.php

```
1 class Person {
2     public $pub;
3     protected $pro;
4     private $pri;
5     public function __construct ($pub, $pro, $pri) {
6         $this->pub=$pub;
7         $this->pro=$pro;
8         $this->pri=$pri;
9     }
10    public function __get ($name) {
11        $a=get_object_vars ($this); print_r ($a);
12        if (isset ($a[$name])) return $a[$name];
13        return null;
14    }
15    public function __toString () {
16        return $this->pub. " ". $this->pro. " ". $this->pri;
17    }
18 }
19 $p=new Person ("Snowden", "Edward", "Moscow");
20 echo $p. "\n"; // Snowden Edward Moscow via __toString
21 echo $p->pub. "\n"; // Snowden par accès direct
22 echo $p->pro. "\n"; // Array([pub]=> Snowden [pro]=> Edward [pri]=> Moscow) par __get
23 //Edward par __get
```

Réflexion avec l'API ReflectionClass

Donne accès aux propriétés et méthodes d'une classe cible

Permet d'invoquer accesseurs (getters) et mutateurs (setters) si l'on s'impose une convention de nommage, e.g.,

- CamelCase : setProp () pour la propriété \$prop.
- snake-case : set_prop () pour la propriété \$prop.

reflection-class.php

```
1 require ('employee.php');
2 $reflector=new ReflectionClass('Employee');
3 $e=new Employee("turing",100,"IQ_society");
4 $properties=$reflector->getProperties();
5 foreach ($properties as $property) {
6     $pname = $property->getName(); // objet ReflectionProperty
7     if ($property->isPublic())
8         echo $pname, " = ", $property->getValue($e), "\n";
9     else
10        if ($reflector->hasMethod("set".ucfirst($pname))) {
11            $method=$reflector->getMethod("set".ucfirst($pname)); // objet ReflectionMethod
12            $method->invoke($e,1000000);
13        }
14    }
15 //propriétés affichées et méthode setSalaire invoquée !
16 var_dump($e);
```

Particularité de la réflexion

Le transtypepage d'un objet en tableau permet d'obtenir toutes ses propriétés.

reflection-array.php

```
1 class A {  
2     public $pubA;  
3     protected $proA;  
4     private $priA;  
5 }  
6 class B extends A {  
7     public $pubB;  
8     protected $proB;  
9     private $priB;  
10 }  
11 $b = new B();  
12 $arr = (array) $b; //transtypepage explicite  
13 print_r($arr);  
14 //Array([pubB] => [*proB] => [BpriB] => [pubA] => [*proA] => [ApriA] => )
```

CM PHP : Sérialisation

Sérialisation/Désérialisation

Sérialisation (alias linéarisation)

- Transmettre des valeurs (scalaires, tableaux, objets) entre pages web en les représentant sous forme de chaînes de caractères (sérialisation) pour pouvoir ensuite les reconstruire (désérialisation).
- Choisir un format de sérialisation standard (eg. JSON).
- Les sérialisations sont des chaînes binaires qui peuvent être stockées en fichier ou base de données (champ de type BLOB).
- Mécanisme utilisé par PHP notamment pour la sauvegarde et restitution des données de session.

Fonction `serialize`

```
string serialize(mixed $value)
```

- S'applique à tout type de valeur sauf `resource`.
- Les références non-circulaires sont perdues.
- Sur les objets :
 - Linéarise les propriétés mais pas les méthodes.
 - Invoque la méthode magique `__sleep` au préalable si elle est définie ou la méthode `serialize` si la classe implémente l'interface `Serializable`.

Fonction unserialize

```
mixed unserialize(string $str[, array $options])
```

- Reconstruit une valeur à partir de sa linéarisation.
- Ne peut reconstruire intégralement un objet que si la définition de sa classe est dans la portée.
- Sur les objets, invoque la méthode magique `__wakeup` au préalable si elle est définie ou la méthode `unserialize` si la classe implémente l'interface `Serializable`.

Exemple de (dé)sérialisation d'objet

serialize-class.php

```
1 class A {
2     private $entier = 1234;
3     public $nom      = "nom";
4     public function __construct(int $i, string $n) {
5         $this->entier = $i;
6         $this->nom   = $n;
7     }
8     public function __toString() : string {
9         return $this->entier . " -- " . $this->nom;
10    }
11 }
```

Exemple de sérialisation d'objet

serialize.php

```
1 include ("serialize-class.php");
2 $a1 = new A(111, "name_a1");
3 $s = serialize($a1);
4 echo $s; //O:1:"A":2:{s:9:"Aentier";i:111;s:3:"nom";s:7:"name_a1";}
5 //enregistrer $s où unserialize.php peut le retrouver
6 file_put_contents (__DIR__ . '/serialize-store.txt', $s);
```

unserialize.php

```
1 // il faut importer la définition de la classe pour que
2 // unserialize() puisse reconstruire l'objet sérialisé
3 require "serialize-class.php";
4 $s = file_get_contents (__DIR__ . '/serialize-store.txt');
5 $a = unserialize($s);
6 var_dump ($a); // object(A)#1 (2) { ["entier":"A":private]=> int(111) ["nom"]=> string(7)
"name_a1"}
```

Interface Serializable

interface-serializable.php

```
1 interface Serializable {  
2     public function serialize();  
3     public function unserialize($serialized);  
4 }
```

Utilisation

- La fonction `serialize` appliquée à un objet appelle la méthode de même nom si la classe de l'objet implémente l'interface `Serializable`.
- La fonction `unserialize` devant retourner un objet appelle la méthode de même nom si la classe implémente l'interface `Serializable`.

Sérialisation et héritage

Classe parente (serialize1.php)

```
1 class Base implements Serializable {
2     private $base_var;
3     public function __construct() {
4         $this->base_var='hello';
5     }
6     public function serialize() {
7         return serialize($this->base_var);
8     }
9     public function unserialize($serialized) {
10        $this->base_var=unserialize($serialized);
11    }
12 }
```

Sérialisation et héritage

Sous-classe (serialize2.php)

```
1 class SubClass extends Base {
2     private $sub_var;
3     public function __construct() {
4         parent::__construct();
5         $this->sub_var='world';
6     }
7     public function serialize() {
8         $base=parent::serialize();
9         return serialize(array($this->sub_var,$base));
10    }
11    public function unserialize($serialized) {
12        $data=unserialize($serialized);
13        $this->sub_var=$data[0];
14        parent::unserialize($data[1]);
15    }
16 }
```

CM PHP : Cookies et sessions

Cookies et sessions

Rappel

HTTP est un protocole de transmission sans état (stateless).

- Aucun lien n'est établi par définition entre deux requêtes émanant d'un même poste client : eg. navigation par hyperlien, saisie de formulaire.

Motivations

Conserver des informations pour améliorer le service rendu par un site à un utilisateur.

- Personnaliser le contenu des pages grâce aux cookies.
- Partager des informations entre différentes pages grâce aux sessions.

Cookies

Fichiers écrits par un script sur le poste client

- Chaque cookie ne peut dépasser 4Ko.
 - Chaque site ne peut écrire plus de 20 cookies sur le poste client.
-
- Les cookies sont l'un des mécanismes possibles pour la mise en place de sessions.
 - La création et le stockage de cookies peuvent être désactivés par l'utilisateur sur son navigateur.

Ecriture des cookies

```
setcookie(string nom [, string valeur, int datefin,  
string chemin, string domaine, bool securite]:bool
```

nom	Le nom du cookie (même règles que pour les variables).
valeur	La valeur associée au cookie.
datefin	Date d'expiration sous la forme d'un timestamp Unix. Exemple : time() + 86400 pour 24h.
chemin	Chemin du dossier des scripts autorisés à accéder au cookie. Exemple : / pour l'intégralité des scripts du domaine.
domaine	Nom entier du domaine où l'accès au cookie est autorisé. Exemple : www.acme.com
securite	TRUE si transmission https requise, FALSE sinon (http par défaut).

Aucun contenu HTML (écrit en dur ou généré via echo ...) ne doit être produit avant appel à setcookie comme pour header.

Ecriture des cookies

cookies-creation.php

```
1 // Cookie valable uniquement pour la session
2 setcookie("cookie", "session");
3 // Cookie valable 15 secondes
4 setcookie("cookie15s", "15s", time() + 15);
5 // Cookie valable 1 heure
6 setcookie("cookie1h", "1h", time() + 3600);
7 // Cookie valable 1 journée
8 setcookie("cookie1j", "1j", time() + 86400);
9 // Cookie "sensible"
10 //setcookie("CB", "5612 1234 5678
1234",time() + 86400, "/client/paiement/", "www.acme.com", TRUE);
11 setcookie("CB", "5612 1234 5678
1234", time() + 86400, "/12/cm-php-sessions/paiement/", "localhost");
```

Ecriture des cookies

Tableau de cookies

```
setcookie("personne[prenom]", "John");
```

- Crée un tableau de cookies (sans guillemets pour les clés).

exemple12-2.php

```
1 $tabcook = array ("prenom"=>"Paul", "nom"=>"Char",
"ville"=>"Marseille");
2 foreach ($tabcook as $cle=>$valeur) {
3     setcookie("client2[$cle]", $valeur, time() +7200);
4 }
```

Effacement et suppression de cookies

Autres usages de setcookie

```
setcookie("nom");
```

- Efface la valeur précédente du cookie.

```
setcookie("nom", "v", time() - 3600);
```

- Supprime le cookie (réutiliser la valeur courante à l'appel).

cookies-effacement-destruction.php

```
1 setcookie("cookie15s", "30s", time() + 30); //Réaffectation
2 setcookie("cookie1j"); //Effacement
3 setcookie("cookie1h", "1h", time() - 1); //Destruction
4 print_r($_COOKIE);
```

Lecture des cookies

Avec superglobale `$_COOKIE`

En utilisant le nom du cookie comme clé.

- Toutes les pages du script ont accès immédiat aux cookies.
- La page créant le cookie n'y a accès qu'après rechargement.

`cookies-lecture.php`

```
1 $ck1 = $_COOKIE["cookie"] ?? "no cookie";
2 $ck2 = $_COOKIE["cookie15s"] ?? "no cookie15s";
3 $ck3 = $_COOKIE["cookie1h"] ?? "no cookie1h";
4 $ck4 = $_COOKIE["cookie1j"] ?? "no cookie1j";
5 $ck5 = $_COOKIE["CB"] ?? "no CB";
6 echo $ck1 . "<br/>";
7 echo $ck2 . "<br/>";
8 echo $ck3 . "<br/>";
9 echo $ck4 . "<br/>";
10 echo $ck5 . "<br/>";
11 print_r($_COOKIE);
```

Sessions

HTTP ne permet pas de conserver des informations provenant d'une page pour les utiliser dans une autre.

Support des sessions

- Permet de conserver et partager des informations entre pages pour un même visiteur.
- Aucun autre visiteur n'a accès à ces données.

Mécanisme des sessions en PHP

- Création ou réouverture d'une session par appel à `session_start` en début de script.
- Chaque session est nommée et a un identifiant unique (géré automatiquement par PHP ou contrôlable).
- Transmission du nom et identifiant de session d'une page à l'autre via cookie (automatique) ou par ajout (manuel) aux hyperliens des pages HTML renvoyées.
- A chaque session correspond la superglobale `$_SESSION` qui permet de partager des données entre scripts appelant `session_start`. La sauvegarde s'effectue automatiquement sur un dossier du serveur.
- PHP sérialise les données de session avant arrêt du script puis les désserialise à l'appel de `session_start` sur la base du nom et de l'identifiant de session transmis.

Sessions avec cookies

Possible uniquement lorsque l'écriture de cookies est autorisée par le client

Mise en place

- `session.use_cookies=on` dans `php.ini`

`session_start () :bool`

A appeler en début de script(s) pour activer les sessions.

- PHP se charge de l'écriture et de la lecture du cookie.

Sessions avec cookies

Exemple de partage d'une variable via session

session-avec-cookie.php

```
1 session_start();
2 echo "Nom de session : ", session_name(), "<br/>";
3 echo "Id de session : ", session_id(), "<br/>";
4 $_SESSION["variable"]='valeur';
```

session-avec-cookie1.php

```
1 session_start();
2 echo "Nom de session : ", session_name(), "<br/>";
3 echo "Id de session : ", session_id(), "<br/>";
4 echo '$_SESSION["variable"]', $_SESSION["variable"];
```

Sessions avec cookies

```
unset ($SESSION[ "ma-var" ]);
```

- Pour détruire une variable de session.

```
session_unset();
```

- Pour détruire les variables d'une session.

```
session_destroy();
```

- Pour détruire la session.

Sessions sans cookies

Lorsque l'écriture de cookies est impossible

```
session_start ()
```

A appeler en début de script(s) pour activer les sessions.

- Nom de session par défaut : PHPSESSID.
- Identifiant de session (26 caractères aléatoires) généré automatiquement, eg. 56epahncd9lh3mjap2eqorgf98.
- Nom et id de session stockés dans constante SID.

Recharge automatiquement "la" session si SID est ajoutée aux hyperliens.

Sessions sans cookies

Usage de l'identifiant de session

session-sans-cookie.php

```
1 session_start();
2 if ( empty( $_SESSION[ 'count' ] ) ) {
3     $_SESSION[ 'count' ] = 1;
4 }
5 else {
6     $_SESSION[ 'count' ]++;
7 }
8 echo "<p>Vous avez vu cette page {$_SESSION[ 'count' ]}
fois.</p>";
9 $query_string = session_name() . '=' . urlencode( session_id() ); // or SID;
10 echo "Pour continuer, <a
href=\"$session-sans-cookie.php?$query_string\">cliquez ici</a>.";
```

CM PHP : Fichiers

Notion de flux

Flux (stream)

- Une ressource (type `resource`) accessible linéairement en lecture ou en écriture : fichier, ressource Web, données de BDD,
...
- Identifié avec la syntaxe `scheme://target`
 - `file:///*` : accès aux fichiers locaux
 - `http:///*` : accès HTTP aux URL
 - `ftp:///*` : accès FTP aux URL
 - `zlib:///*` : compression de données
 - `phar:///*` : archive PHP
 - `ssh2:///*` : SSH 2
 - ...

Gestionnaire de protocole (wrapper)

Code PHP pour gérer certains protocoles/encodages liés à un flux.

- eg. le wrapper HTTP convertit une URL en requête HTTP.

Contexte de flux

Contexte de flux

- Un ensemble de paramètres et d'options spécifiques à un wrapper pour contrôler le comportement d'un flux.
- Se crée avec la fonction `resource stream_context_create([array $options [, $params]]).`
- Se passe ensuite en argument aux fonctions de création et d'accès aux flux : `fopen`, `file_get_contents`, `file_put_contents`, ...

Exemple : accès HTTP

Récupération d'une ressource Web par son URL (file_get_contents3.php)

```
1 // création des options du contexte de flux
2 $opts = array (
3     'http'=>array (
4         'method'=>"GET",
5         'header'=>"Accept-language: en\r\n".
6             "Cookie: foo=bar\r\n"
7     )
8 );
9 // création du contexte de flux
10 $context = stream_context_create ($opts);
11 // accès au flux "http://www.univ-angers.fr"
12 // avec envoi par le wrapper PHP pour HTTP
13 // d'une requête HTTP/1.1
14 // contenant les entêtes HTTP ci-dessus
15 $file = file_get_contents ('http://www.univ-angers.fr/');
16 echo $file;
```

Gestion des fichiers

Lecture et écriture de fichiers

A l'aide de

- Fonctions : fopen, fread, fwrite, fclose, file_get_contents, file_put_contents, ...
- Deux classes SPL : SPLFileInfo et SPLFileObject.

Différentes modalités d'accès

- Lecture seule, écriture seule, ou les deux.
- Ecrasement de fichier ou ajout de données par écriture.
- Lecture intégrale, ligne par ligne, caractère par caractère.
- Lecture automatique de données formattées.
- Positionnement de pointeur de lecture/écriture.
- Verrouillage d'accès.
- Copie, renommage, effacement, métadonnées.

Lecture de fichier avec file_get_contents

```
string file_get_contents(string $f [, ...])
```

Lit le contenu d'un fichier en totalité et le retourne dans une chaîne.

- `$f` : nom du fichier.
- Drapeaux :
 - recherche ou non du fichier dans le `$PATH`.
 - contexte de flux valide (`NULL` pour les fichiers).
 - positionnement du pointeur de lecture.
 - nombre d'octets à lire ou bien lecture intégrale.

Retourne `FALSE` en cas d'erreur.

Emet un `E_WARNING` si fichier introuvable.

Lecture de fichier avec file_get_contents

Lecture d'un fichier local (file_get_contents1.php)

```
1 $filename=__DIR__.'/html.txt';
2 $str=file_get_contents($filename);
3 if ($str===false) {
4     throw Exception('could not read');
5 } else {
6     echo $str;
7 }
```

Lecture d'une page HTML (file_get_contents2.php)

```
1 $file=file_get_contents("http://www.univ-angers.fr");
2 echo $file;
```

Ecriture de fichier avec file_put_contents

`file_put_contents(string $f, mixed $d [, ...])`

Écrit dans le fichier spécifié par `$f` la donnée `d` :

- `$d` : chaîne, tableau ou ressource de flux.
- Drapeaux :
 - ou-logique entre constantes : `FILE_USE_INCLUDE_PATH` (recherche dans le `$PATH`), `FILE_APPEND` (concaténation vs. écrasement), `LOCK_EX` (verrou exclusif).
 - contexte de flux (`NULL` pour les fichiers)

Crée le fichier s'il n'existe pas, sinon l'écrase ou y ajoute les données selon l'option `FILE_APPEND`.

Retourne le nombre d'octets écrits ou `FALSE` en cas d'erreur.

Ecriture de fichier avec file_put_contents

Ecriture dans un fichier local (file_put_contents1.php)

```
1 $fname=__DIR__.'/file_put_contents1.txt';
2 $tab=array('odyssée', 'de', 'l\'espace', 2001);
3 echo file_put_contents($fname, implode(' ', $tab)."\n");
```

Utilisation de fopen, fwrite, fclose

```
resource fopen(string $f, string $mode [, ...])
```

- Ouvre un fichier ou une URL spécifiée par \$f.
- Drapeaux :
 - \$mode : lecture seule "r", lecture et écriture "r+", écriture seule avec écrasement "w", lecture et écriture avec écrasement "w+", écriture seule par ajout "a", lecture et écriture par ajout "a+".
 - recherche dans le \$PATH.
 - contexte de flux valide (NULL pour les fichiers).

```
int fwrite(resource $h, string $s [,int $n])
```

- Ecrit le contenu de \$s (maximum \$n octets) dans le fichier identifié par \$h.

```
bool fclose(resource $h)
```

- Ferme le fichier identifié par \$h.

Utilisation de fopen, fwrite, fclose

Exemple de compression de fichier (fopen.php)

```
1 /* création d'un fichier compressé (binaire)
2 * Le fichier peut être décompressé
3 * et lu avec le wrapper compress.zlib stream
4 * ou en ligne de commande 'gzip -d foo-bar.txt.gz'
5 */
6 $fp = fopen("compress.zlib://foo-bar.txt.gz", "wb");
7 if ($fp) die("Unable to create file.");
8 fwrite($fp, "This is a test.\n");
9 fclose($fp);
```

Autres fonctions

Nom	Description
flock	dé-/verrouillage d'accès.
fgetc	lecture d'un caractère à la fois.
fgets	lecture d'une ligne à la fois (délimiteur \n).
fread	lecture de blocs de taille prédéfinie.
fseek	positionnement du pointeur.
rewind	positionner le pointeur en début de fichier.
ftell	position du pointeur.
fgetcsv	lecture d'une ligne à la fois et conversion sous forme de tableau de mots basée sur séparateur prédéfini.
readfile	lecture intégrale et affichage sur la sortie standard.

Autres fonctions

Nom	Description
copy	copie.
rename	renommage.
unlink	suppression.
file_exists	existence.
filesize	taille.
filetype	de type fichier ou répertoire.
is_file	vérification.
is_readable	accessibilité en lecture.
is_writable	accessibilité en écriture.
is_uploaded_file	test de téléchargement.
fileatime	dernier accès.
filemtime	dernière modification.
filectime	dernière modification des permissions.
realpath	chemin d'accès.
basename	extraction du nom à partir d'un chemin.

La classe SPLFileInfo

Classe SPLFileInfo

- Instanciée avec le nom d'un fichier.
- Ses méthodes donnent accès aux métadonnées système sur le fichier : date d'accès, extension, nom, groupe, noeud d'index (inode), fichier cible (lien), propriétaire et permissions, chemin, taille, type ...

splFileInfo.php

```
1 $file=new SPLFileInfo(__DIR__.'/html.txt');
2 echo $file->getExtension()."\n";
3 echo $file->getSize()."\n";
4 echo $file->isReadable()."\\n";
```

La classe SPLFileObject

Classe SPLFileObject

- Hérite de `SPLFileInfo` et implémente `Raversable`, `RecursiveIterator` et `SeekableIterator`.
- Instanciée avec le nom d'un fichier.
- Offre un large éventail de méthodes pour manipuler des fichiers.

Envoi d'une image en réponse HTTP (`splfileobject.php`)

```
1 // ouvre le fichier en mode lecture binaire
2 $file = new SplFileObject(__DIR__."/imagesvg.png", "rb");
3 // envoie les bonnes en-têtes
4 header("Content-Type: image/png");
5 header("Content-Length: " . $file->getSize());
6 // affiche sur le tampon de sortie et termine le script
7 $file->fpassthru();
8 exit;
```

Parcours de répertoires (SPL)

spl-directory.php

```
1 try {
2     foreach (new DirectoryIterator('.//') as $item) {
3         echo $item."\n";
4     }
5 } catch (Exception $e) {
6     echo "No files found!\n";
7 }
```

CM PHP : APIs XML

Manipuler des documents XML sous PHP

On peut utiliser

- L'API `DOMDocument`.
- L'extension `SimpleXML`.

L'API DOMDocument

Créer un document XML avec DOMDocument

On précise :

- La version.
- L'encodage des caractères.

dom-create.php

```
1 $dom = new DOMDocument('1.0', 'utf-8');  
2 echo $dom->saveXML();
```

Charger un document XML/HTML

Méthodes booléennes

- `load(string)` pour un fichier XML.
- `loadHTMLFile(string)` pour un fichier HTML.
- `loadXML(string)` à partir d'une chaîne.
- `loadHTML(string)` à partir d'une chaîne.

dom-load.php

```
1 error_reporting(E_ALL);
2 $dom=new DOMDocument();
3 if ($dom->load(__DIR__.'/bibliotheque.xml') ===false) {
4 die("error load");
5 } else {
6 echo $dom->saveXML();
7 }
```

Valider un document XML avec une DTD

dom-validate.php

```
1 error_reporting(E_ALL);
2 $dom = new DOMDocument();
3 $docs = array();
4 $docs[] = 'bibliotheque.xml';
5 $docs[] = 'bibliotheque-invalide.xml';
6 foreach ($docs as $doc) {
7     $dom->load($doc);
8     echo "Document ".$doc;
9     if ($dom->validate() === true)
10         echo " valide\n";
11     else
12         echo " invalide\n";
13 }
```

Sauvergarder un document XML/HTML

Retournent le nombre d'octets écrits en cas de succès et FALSE en cas d'erreur

- `save(string)` pour un fichier XML.
- `saveHTMLFile(string)` pour un fichier HTML.

Retournent une chaîne XML/HTML en cas de succès et FALSE en cas d'erreur

- `string saveXML(void).`
- `string saveHTML(void).`

Exemple de sauvegarde

dom-save.php

```
1 $dom=new DOMDocument();
2 $root=$dom->createElement('bibliotheque','some text');
3 $dom->appendChild($root);
4 if ($dom->save(__DIR__.'/dummy.xml')==false)
5 die("error save");
6 echo $dom->saveXML();
```

dummy.xml

```
1 <?xml version="1.0"?>
2 <bibliotheque>some text</bibliotheque>
```

Créer un document XML/HTML

Méthodes principales

- DOMElement createElement(string name, string value)
- DOMText createTextNode(string name)
- DOMAttr createAttribute(string name)
- appendChild(DOMNode node)

Exemple de création

dom-create-element.php

```
1 $dom=new DOMDocument('1.0', 'iso-8859-1');
2 $root_element=$dom->createElement('bibliotheque','');
3 $dom->appendChild($root_element);
4 $livre=$dom->createElement('livre','');
5 $root_element->appendChild($livre);
6 $titre=$dom->createElement('titre','Apprendre XML');
7 $livre->appendChild($titre);
8 $prix=$dom->createElement('prix','20');
9 $livre->appendChild($prix);
10 $monnaie=$dom->createAttribute('monnaie');
11 $montant=$dom->createTextNode('euro');
12 $monnaie->appendChild($montant);
13 $prix->appendChild($monnaie);
14 echo $dom->saveXML();
```

Parcourir un document XML/HTML

Méthodes principales

- DOMElement getElementById(string elementId)
- DOMNodeList getElementsByTagName(string name)

Fichier exemple

bibliotheque.xml

```
1 <?xml version="1.0" encoding="utf-8" standalone="no"?>
2 <!-- Déclaration externe de DTD -->
3 <!DOCTYPE bibliotheque SYSTEM "bibliotheque.dtd">
4 <!-- Référence à un fichier XSLT pour mise en forme -->
5 <?xmlstylesheet type="text/xsl" href="bibliotheque-liste.xsl" ?>
6 <bibliotheque>
7 <livre>
8   <titre>Apprendre XML en 10 mois</titre>
9   <liste_auteurs>
10    <auteur nom="Escargot" prenom="Jean" />
11    <auteur nom="Snail" prenom="John" />
12   </liste_auteurs>
13   <prix>20</prix>
14 </livre>
15 <livre>
16   <titre>Learn Java in 10 seconds</titre>
17   <liste_auteurs>
18    <auteur nom="Fast" prenom="Bob" />
19   </liste_auteurs>
20   <prix monnaie="dollars">25</prix>
21 </livre>
22 </bibliotheque>
```

Exemple de parcours

dom-traverse.php

```
1 function parcours_attributes ($node) {
2     if (!$node->hasAttributes()) return ;
3     foreach ($node->attributes as $key => $subnode)
4         echo " $key=". $subnode->nodeValue;
5 }
6 function parcours ($node,$nbr) {
7     if ($node->hasChildNodes())
8         foreach (range(0,$node->childNodes->length-1) as $index) {
9             $subnode=$node->childNodes->item($index);
10            if ($subnode->nodeType==XML_ELEMENT_NODE) {
11                foreach (range(0,4*$nbr) as $i) echo " ";
12                echo "+". $subnode->nodeName." = ".trim($subnode->nodeValue);
13                parcours_attributes($subnode); echo "\n";
14                parcours ($subnode,$nbr+1);
15            }
16        }
17    }
18 $dom=new DOMDocument();
19 $dom->load(__DIR__.'/bibliotheque.xml');
20 $liste=$dom->getElementsByTagName('livre');
21 foreach (range(0,$liste->length-1) as $ndx_livre) {
22     $livre=$liste->item($ndx_livre);
23     echo "livre $ndx_livre\n";
24     parcours ($livre,1);
25 }
```

Résultat du parcours

dom-traverse-out.txt

```
livre 0
    +titre = Apprendre XML en 10 mois
    +liste_auteurs =
        +auteur = nom=Escargot prenom=Jean
        +auteur = nom=Snail prenom=John
    +prix = 20
livre 1
    +titre = Learn Java in 10 seconds
    +liste_auteurs =
        +auteur = nom=Fast prenom=Bob
    +prix = 25 monnaie=dollars
```

L'extension SimpleXML

L'extension SimpleXML

Fournit

- Des fonctions simples de chargement de fichiers ou chaînes XML.
- Une classe `SimpleXMLElement` dont les instances permettent, via leurs propriétés et méthodes, d'inspecter ou modifier attributs et contenu des éléments (texte ou sous-éléments).
- Une classe `SimpleXMLIterator` pour itérer récursivement sur les éléments d'instances `SimpleXMLElement`.

Tout est objet : éléments, noeuds texte, attributs

Veuillez à caster noeuds texte et attributs au bon type (`string`, `int`, etc.) pour éviter les erreurs de traitement.

Convertir noeuds texte et attributs avec SimpleXML

simplexml-casting.php

```
1 $xml = simplexml_load_file( __DIR__ . "/bibliotheque.xml" );
2 echo gettype( $xml->livre->titre ); // object
3 echo "<br>";
4 echo $xml->livre->titre; // Apprendre XML en 10 mois
5 echo "<br><br>";
6 var_dump( $xml->livre->liste_auteurs->auteur[ 0 ] ); // object(SimpleXMLElement) ...
7 echo "<br><br>";
8 $nom = $xml->livre->liste_auteurs->auteur[ 0 ][ "nom" ];
9 echo gettype( $nom ); // !!! object
10 echo "<br>";
11 echo $nom; // Escargot
12 echo "<br><br>";
13 $prix2ndlivre = $xml->livre[ 1 ]->prix;
14 echo json_encode( $prix2ndlivre ); // {"@attributes":{"monnaie":"dollars"},"0":"25"}
15 echo "<br>";
16 echo json_encode( (int) $prix2ndlivre ); // 25
```

Exemple de parcours avec SimpleXML

simplexml.php

```
1 $xml = simplexml_load_file( __DIR__ . "/bibliotheque.xml" );
2 foreach ( $xml->livre as $livre ) {
3     echo "-----<br>";
4     echo "titre : " . $livre->titre . "<br>";
5     $liste_auteurs = $livre->liste_auteurs->auteur; // !! pointe sur le tableau
d'auteurs
6     foreach ( $liste_auteurs as $auteur )
7         echo "auteur : " . $auteur[ "prenom" ]
8             . " " . $auteur[ "nom" ] . "<br>";
9     echo "prix : " . $livre->prix . " "
10    . $livre->prix[ "monnaie" ] . "<br>";
11 }
```

Résultat du parcours avec SimpleXML (1/2)

simplexml-out1.txt

```
SimpleXMLElement Object
(
    [livre] => Array
        (
            [0] => SimpleXMLElement Object
                (
                    [titre] => Apprendre XML en 10 mois
                    [liste_auteurs] => SimpleXMLElement Object
                        (
                            [auteur] => Array
                                (
                                    [0] => SimpleXMLElement Object
                                        (
                                            [@attributes] => Array
                                                (
                                                    [nom] => Escargot
                                                    [prenom] => Jean
                                                )
                                        )
                                )
            )
        )
)
[1] => SimpleXMLElement Object
(
    [@attributes] => Array
        (
            [nom] => Snail
            [prenom] => John
        )
)
)
```

Résultat du parcours avec SimpleXML (2/2)

simplexml-out2.txt

```
-----  
titre : Apprendre XML en 10 mois  
auteur : Jean Escargot  
auteur : Brad Snail  
prix : 20  
-----  
titre : Learn Java in 10 seconds  
auteur : John Fast  
prix : 25 dollars
```

XPath

Interrogation d'un document XML avec XPath

XPath - XML Path Language

- XPath est un élément du standard XSLT.
- XPath 3.0 est une recommandation du [W3C](#) (2014).

Syntaxe

- De type "chemin" pour identifier et parcourir des noeuds de document XML.
- Nombreuses autres fonctions pour différents types de données.

Les expressions XPath sont directement utilisables en

- JavaScript.
- PHP avec la classe `DOMXPath` ou la méthode `xpath` de `SimpleXMLElement`.
- Java ...

Exemples de requêtes XPath

Voir Cheatsheet

titre	Tous les éléments titre.
/	L'élément racine.
//titre	Tous les noeuds titre sous le noeud courant.
.	Le noeud courant.
..	Le noeud parent.
@	Des attributs.
//liste_auteurs/auteur	Tous les éléments auteur à l'intérieur de liste_auteurs.
//prix[@monnaie]	Tous les éléments prix qui ont un attribut monnaie.
//prix[@monnaie='dollars']	Tous les éléments prix qui ont un attribut monnaie égal à dollars.

Utilisation intégrée dans SimpleXML

simplexml-xpath.php

```
1 $xml = simplexml_load_file( __DIR__ . "/bibliotheque.xml" );
2 $livres = $xml->xpath( "//livre" ); // tous les livres de <bibliotheque>
3 $livre2 = $xml->xpath( "//livre[1]" ); // tableau contenant le 2ème livre de <bibliotheque>
4 echo $livre2[ 0 ]->titre; // Apprendre XML en 10 mois
5 echo "<br>";
6 $auteur2_1 = $xml->xpath( "//livre[2]/liste_auteurs/auteur[1]" ); // tableau contenant le 1er auteur
du 2ème livre de <bibliotheque>
7 echo $auteur2_1[ 0 ][ "nom" ]; // Escargot
8 echo "<br>";
9 $bobs = $xml->xpath( "//auteur[starts-with(@prenom, 'J')]" );
10 array_walk(
11     $bobs,
12     function ($bob)
13     {
14         echo $bob[ "prenom" ] . "<br>";
15     }
16 ); // Jean <br> John
```

Utilisation de la classe DOMXPath avec l'API DOM

dom-xpath.php

```
1 $doc = new DOMDocument();
2 $doc->load("bibliotheque.xml");
3 $xpath = new DOMXPath($doc);
4 echo "traiter les titres\n";
5 $elements=$xpath->query("//titre");
6 if (!is_null($elements))
7     foreach ($elements as $element) {
8         echo "> ". $element->nodeName. " : ";
9         $nodes = $element->childNodes;
10        foreach ($nodes as $node) echo $node->nodeValue. "\n";
11    }
12 echo "\ntraiter les noms des auteurs\n";
13 $elements = $xpath->query("//auteur[@nom]");
14 if (!is_null($elements))
15     foreach ($elements as $element)
16         echo ">". $element->nodeName. " " . $element->getAttribute("nom")."\n";
17 echo "\ntraiter les auteurs et prénoms des auteurs\n";
18 $elements = $xpath->query("//auteur[@prenom]");
19 if (!is_null($elements))
20     foreach ($elements as $element)
21         echo ">". $element->nodeName. " " . $element->getAttribute("nom")
22             ." ". $element->getAttribute("prenom")."\n";
```

Résultat du traitement avec DOMXPath

dom-xpath-out.txt

traiter les titres

=> titre : Apprendre XML en 10 mois

=> titre : Learn Java in 10 seconds

traiter les noms des auteurs

=>auteur Escargot

=>auteur Snail

=>auteur Fast

traiter les noms et prenoms des auteurs

=>auteur Escargot Jean

=>auteur Snail Brad

=>auteur Fast John

Utilitaire xpath

```
xpath -e <query> <file>
```

Utilitaire PERL pour interroger un fichier XML par requête XPath.

Options :

- `-e <query>` : Requête à exécuter.
- `-q` : Mode silencieux.
- `-s <suffix>` : Placer le suffixe (par défaut retour chariot) à la fin de chaque ligne.
- `-p <prefix>` : Placer le préfixe (par défaut rien) au début de chaque ligne.

CM PHP : Bases de données

L'API PDO

PDO - PHP Data Objects (PHP 5+)

Fournit une interface d'abstraction à l'accès aux SGBD.

- Des pilotes (drivers) sont implémentés dans l'interface PDO pour différentes BDD.
- On utilise les mêmes fonctions pour exécuter des requêtes SQL quelle que soit la BDD utilisée.

Bases de données

Bases de données supportées

- CUBRID
- Sybase/MS SQL
- Firebird/Interbase
- IBM DB2
- IBM Informix
- MySQL 3, 4, 5
- Oracle
- ODBC v3
- PostgreSQL
- SQLite 2, 3
- Microsoft SQL Server / SQL Azure

Connexion

Etablie en instanciant la classe PDO

Le constructeur accepte différents paramètres :

- **Data Source Name** (DSN) : type de SGBD, hôte, port, BDD ...
- Nom d'utilisateur (optionnel).
- Mot de passe (optionnel).
- Tableau d'options : persistance/cache des connexions, compression, sécurité ...

Connexion à MySQL (pdo-connexion-open.php)

```
1 $sgbd = "mysql"; // choix de MySQL (fonctionnera aussi avec MariaDB !)
2 $host = "localhost";
3 $port = 3306; // port par défaut de MySQL (à adapter selon votre config et votre choix entre mysql/mariadb)
4 $charset = "UTF8";
5 $user = "etudiant"; // user id
6 $pass = "antietud"; // password
7 $pdo = new pdo("$sgbd:host=$host;port=$port;charset=$charset", $user, $pass, array(
8     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
9 ));
```

Gestion des erreurs

Trois modes de gestion d'erreurs (connexion, requêtes) avec

`PDO::setAttribute(PDO::ATTR_ERRMODE, $mode)`

- Mode “silencieux” (par défaut) : aucune exception émise mais un rapport d'erreurs accessible via `PDO::errorInfo()`.
- Mode “avertissement” : émission de `E_WARNING` sans interruption du script.
- Mode “exception” : déclenchement d'une exception par instantiation de la classe `PDOException`.

Class `PDOException`

- Fournit différentes méthodes de diagnostic d'erreur dont `getCode()` qui renvoie un code d'erreur standard `SQLSTATE`.

Gestion des erreurs

Rapport d'erreur (pdo-connexion-error.php)

```
1 $sgbd = "mysql"; // choix de MySQL (fonctionnera aussi avec MariaDB !)
2 $host = "localhost";
3 $port = 3306; // port par défaut de MySQL (à adapter selon votre config et votre choix entre mysql/mariadb)
4 $charset = "UTF8";
5 $user = "etudiant"; // user id
6 $pass = "antietud"; // password
7 $db = "l3info_cm_php_bdd";
8 try {
9     $pdo = new pdo("$sgbd:host=$host;port=$port;dbname=$db;charset=$charset", $user,
$pass, array(
10         PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
11     ));
12     $pdo->query("This is no SQL!");
13 } catch (PDOException $e) {
14     echo $e->getCode(); // 42000
15     echo $e->getMessage(); // SQLSTATE[42000]: Syntax error or access violation: 1064
You have an error in your SQL syntax ...
16 }
```

Déconnexion

Une connexion reste active tant que l'objet PDO existe

Pour fermer une connexion avant la fin du script, détruire l'objet et ses références (eg. instances de PDOStatement référençant l'objet).

Fermeture optionnelle (pdo-connexion-close.php)

```
1 $pdo = new PDO('mysql:host=localhost;dbname=l3info_cm_php_bdd');
2 $sth = $pdo->query('SELECT * FROM foo');
3 // détruire toute référence à l'objet PDO
4 $sth = null;
5 $pdo = null;
```

Mise en cache de connexions persistantes

Avec option PDO::ATTR_PERSISTENT => true à linstanciation de PDO.

Requêtes SQL avec PDO

Requête SQL avec PDO

De type `string`.

Ecrite en respectant la syntaxe SQL.

Dont on peut échapper les paramètres avec `PDO::quote()`

Exécutée avec `PDO::query()`

Cas des requêtes SELECT

`PDO::query()` renvoie une instance de `PDOStatement`.

On récupère les enregistrements (alias lignes) avec l'une des méthodes `fetch` de cette instance :

- `fetch(...)` : la ligne suivante
- `fetchAll(...)` : toutes les lignes sous forme de tableau
- `fetchColumn(...)` : une colonne de la ligne suivante
- `fetchObject(...)` : la ligne suivante sous forme d'objet

Création et connexion à une base de données

Création d'une BDD (pdo-create-db.php)

```
1 require 'pdo-connexion-open.php';
2 $db = "l3info_cm_php_bdd";
3 $str="CREATE DATABASE IF NOT EXISTS $db CHARACTER SET=utf8mb4 COLLATE utf8mb4_bin";
4 $pdo->query($str);
```

Connexion avec sélection de la BDD (pdo-connexion-db.php)

```
1 $sgbd = "mysql"; // choix de MySQL (fonctionnera aussi avec MariaDB !)
2 $host = "localhost";
3 $port = 3306; // port par défaut de MySQL (à adapter selon votre config et votre choix entre mysql/mariadb)
4 $charset = "UTF8";
5 $user = "etudiant"; // user id
6 $pass = "antietud"; // password
7 $db = "l3info_cm_php_bdd";
8 $pdo = new PDO("$sgbd:host=$host;port=$port;dbname=$db;charset=$charset", $user, $pass);
```

Création de table

Création d'une table (pdo-create-table.php)

```
1 require 'pdo-connexion-db.php';
2 $str = "CREATE TABLE `{$db}`.`Employee` 
3     (`id` INT NOT NULL AUTO_INCREMENT ,
4      `name` VARCHAR(40) NOT NULL ,
5      `salary` FLOAT NOT NULL ,
6      `age` INT(100) NOT NULL , PRIMARY KEY (`id`))
7     ENGINE=InnoDB
8     COLLATE utf8mb4_bin;";
9 try {
10     $pdo->query( $str );
11 }
12 catch ( PDOException $e ) {
13     echo $e->getMessage();
14 }
```

Insertion d'une ligne

pdo-record-create.php

```
1 require 'pdo-connexion-db.php';
2 $str="INSERT INTO Employee (name,salary,age) VALUES
('superman',10000,85)";
3 $pdo->query($str);
```

- Inutile d'insérer une valeur d'identifiant (ID).
- Les accents graves (alias backquote) encadrant le nom de table sont optionnels.
- Les apostrophes ou guillemets encadrant une valeur de type chaîne de caractères sont obligatoires.

Sélection de lignes

Récupérer les lignes tour à tour

Avec `PDOStatement::fetch([int $style...])`.

Plusieurs formats PHP de lignes sont supportés selon la valeur de `$style` utilisée

- `FETCH_NUM` : tableau indicé.
- `FETCH_ASSOC` : tableau associatif.
- `FETCH_BOTH` : tableau avec indices et clés (valeur par défaut).
- `FETCH_CLASS` : objet anonyme ou de classe prédefinie.
- `FETCH_INTO` : objet mis à jour.

Récupération de ligne sous forme de tableau indicé

Utilisation de `FETCH_NUM` (`pdo-record-read-num.php`)

```
1 require_once ('pdo-connexion-db.php') ;
2 $qry="SELECT * FROM Employee WHERE name='superman' ;
3 $stt=$pdo->query($qry) ;
4 while ($record=$stt->fetch(PDO::FETCH_NUM)) {
5     print_r($record) ;
6 }
```

Array

```
(  
    [0] => 1  
    [1] => superman  
    [2] => 10000  
    [3] => 85  
)
```

Récupération de ligne sous forme de tableau associatif

Utilisation de `FETCH_ASSOC` (`pdo-record-read-assoc.php`)

```
1 require_once('pdo-connexion-db.php');
2 $qry="SELECT * FROM Employee WHERE age<80";
3 $stt=$pdo->query($qry);
4 while ($record=$stt->fetch(PDO::FETCH_ASSOC)) {
5     print_r($record);
6 }
```

```
Array
(
    [id] => 2
    [name] => batman
    [salary] => 10.5
    [age] => 78
)
```

```
Array
(
    [id] => 3
    [name] => spiderman
    [salary] => 80
    [age] => 55
)
```

Récupération de ligne sous forme de tableau mixte

Utilisation de `FETCH_BOTH` (`pdo-record-read-both.php`)

```
1 require_once('pdo-connexion-db.php');
2 $qry="SELECT * FROM Employee WHERE name LIKE '%atm%'";
3 $stt=$pdo->query($qry);
4 while ($record=$stt->fetch(PDO::FETCH_BOTH)) {
5     print_r($record);
6 }
```

Array

```
(  
    [id] => 2  
    [0] => 2  
    [name] => batman  
    [1] => batman  
    [salary] => 10.5  
    [2] => 10.5  
    [age] => 78  
    [3] => 78  
)
```

Récupération de ligne sous forme d'objet

Utilisation de `FETCH_CLASS` sur une classe donnée `C`

Invoquer `setFetchMode(PDO::FETCH_CLASS, 'C')` sur l'objet `PDOStatement` avant appel à `fetch/fetchAll`.

- Crée une instance de `C` par ligne résultat en associant propriétés et colonnes par leurs noms.

Règles d'application

- S'il existe une propriété de même nom qu'une colonne, et quelle qu'en soit la visibilité, elle prend sa valeur.
- Sinon, si la méthode magique `__set` est définie, elle est appelée (avec avertissement).
- Sinon, une propriété publique sera créée de même nom et valeur que la colonne (avec avertissement).
- Le constructeur, qui doit être sans arguments, est appelé une fois les propriétés créées

Récupération de ligne sous forme d'objet

Classe de récupération (pdo-employee.php)

```
1 class Employee
2 {
3     public $name = "anonymous";
4     private $id;
5     private $salary;
6     private $age;
7     // OPTIONNEL public function __construct() {}
8     function setName( $name )
9     {
10         $this->name = $name;
11     }
12     function setId( $id )
13     {
14         $this->id = $id;
15     }
16     function setSalary( $salary )
17     {
18         $this->salary = $salary;
19     }
20     public function __set( $p, $v ) // pour PDO::FETCH_INTO sur propriétés inaccessibles
21     {
22         echo "calling __set for property $p<br>";
23         $this->$p = $v;
24     }
25     public function __toString()
26     {
27         return "employee: id=$this->id name=$this->name salary=$this->salary
age=$this->age\n";
28     }
}
```

Récupération de ligne sous forme d'objet

Avec `FETCH_CLASS` (`pdo-record-read-class.php`)

```
1 require_once( 'pdo-connexion-db.php' );
2 require_once( 'pdo-employee.php' );
3 $qry = "SELECT * FROM Employee";
4 $stt = $pdo->query( $qry );
5 $stt->setFetchMode( PDO::FETCH_CLASS, 'Employee' );
6 while ( $employee = $stt->fetch() ) {
7     var_dump( $employee );
8 }
```

Récupération de ligne sous forme d'objet

Utilisation de `FETCH_INTO` sur une instance existante o

Invoquer `setFetchMode (PDO::FETCH_INTO, $o)` sur l'objet
PDOStatement avant appel à `fetch/fetchAll`.

- Met à jour l'objet o à chaque ligne résultat en associant propriétés et colonnes par leurs noms.

Règles d'application

- Appelle la méthode magique `__set` sur les propriétés inaccessibles (protégées / privées).

Récupération de ligne sous forme d'objet

Avec `FETCH_INTO` (`pdo-record-read-class-into.php`)

```
1 require_once( 'pdo-connexion-db.php' );
2 require_once( 'pdo-employee.php' );
3 $qry = "SELECT * FROM Employee";
4 $stt = $pdo->query( $qry );
5 $emp = new Employee();
6 $stt->setFetchMode( PDO::FETCH_INTO, $emp );
7 while ( $stt->fetch() )
8     var_dump( $emp );
```

Récupération des lignes en une fois

Avec `PDOStatement::fetchAll([int $style...]).`

Avec `FETCH_NUM` (`pdo-record-fetchall-num.php`)

```
1 require_once('pdo-connexion-db.php');
2 $qry="SELECT * FROM Employee";
3 $stt=$pdo->query($qry);
4 $records=$stt->fetchAll(PDO::FETCH_NUM);
5 foreach ($records as $record) {
6     print_r($record);
7 }
```

Avec `FETCH_CLASS` (`pdo-record-fetchall-class.php`)

```
1 require_once('pdo-connexion-db.php');
2 require_once('pdo-employee.php');
3 $qry = "SELECT * FROM Employee";
4 $stt = $pdo->query($qry);
5 $emps = $stt->fetchAll(PDO::FETCH_CLASS, 'Employee');
6 foreach ($emps as $emp) {
7     var_dump($emp);
```

Modification et suppression de lignes

Exemple (pdo-record-update.php)

```
1 require_once('pdo-connexion-db.php');  
2 $qry="UPDATE Employee SET salary=200 WHERE name='superman'";  
3 $pdo->query($qry);
```

Exemple (pdo-record-delete.php)

```
1 require 'pdo-connexion-db.php';  
2 $qry="DELETE FROM Employee WHERE id=2";  
3 $pdo->query($qry);
```

Requêtes préparées (prepared statement)

Mécanisme privilégié pour faire de l'insertion “en masse”

Plus performant que les requêtes standard et plus sûr (échappement automatique des valeurs).

Principe

Création d'une unique requête paramétrée dont on substitue ensuite les paramètres par les valeurs réelles de chaque ligne à insérer.

Utilisées aussi pour les requêtes de sélection

Requêtes préparées

Deux étapes :

- ① Préparer une requête SQL incluant en lieu et place des valeurs attendues :
 - soit des paramètres nommés (:age)
 - soit des marqueurs interrogatifs (?)
- ② Pour chaque ligne à insérer, exécuter la requête
 - soit avec le tableau associatif de ses valeurs en utilisant les noms des paramètres comme clés
 - soit avec le tableau indiqué de ses valeurs qui sont substituées aux marqueurs interrogatifs par ordre d'apparition.

INSERT INTO Employee (name,salary,age) VALUES (:x, :y, :z)

INSERT INTO Employee (name,salary,age) VALUES (?, ?, ?)

SELECT * FROM Employee WHERE salary > :s AND age < :a

Insertion en cascade avec requête préparée

Deux étapes :

- ① Préparation en passant la requête (string) à la méthode `PDO::prepare()` qui renvoie une instance de `PDOStatement`.
- ② Exécution en passant le tableau de valeurs de la ligne à la méthode `execute()` de cette instance.

pdo-prepared-statement.php

```
1 require 'pdo-connexion-db.php';
2 $qry="INSERT INTO Employee (name,salary,age) VALUES
3 (:nom,:salaire,:age)";
4 $stt=$pdo->prepare($qry);
5 $data=array(
6     array (':nom'=>'batman',':salaire'=>10.5,':age'=>78),
7     array (':nom'=>'spiderman',':salaire'=>80,':age'=>55));
8 foreach ($data as $row) {
9     $stt->execute($row);
10 }
```

Sélection avec requête préparée

pdo-prepared-statement-select.php

```
1 require 'pdo-connexion-db.php';
2 $qry="SELECT * FROM `Employee` WHERE salary>:s AND age<:a";
3 $stt=$pdo->prepare($qry);
4 $parameters = [':s'=>50, ':a'=>80 ];
5 $stt->execute($parameters);
6 $rows = $stt->fetchAll(PDO::FETCH_ASSOC);
7 print_r($rows);
```

Transactions

Effectuent un ensemble d'opérations de manière groupée

Peuvent être annulées (rollback).

Le SGBD doit garantir les propriétés d'atomicité, cohérence, isolation et durabilité des transactions (**ACID**)

- A Une transaction s'exécute totalement ou pas du tout (eg. en cas de panne, défaillance).
- C Une transaction doit amener la BD dans un état conforme aux règles définies (eg. contraintes d'intégrité).
- I Toute transaction doit s'exécuter comme si elle était la seule (eg. deux transactions "simultanées" doivent aboutir au même état que celui obtenu en les exécutant séquentiellement et quelle que soit la séquence).
- D Les résultats d'une transaction confirmée sont enregistrés de façon permanente.

Transactions

Certains SGBD ne prennent pas les transactions en charge.

La requête SQL a été exécutée avec succès.

`show engines`

Profilage [Éditer en ligne] [Éditer] [Crée le code source PHP] [Actualiser]

+ Options

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking, and fore...	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary...	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it...)	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL

Transactions

Exemple (pdo-transaction.php)

```
1 // code omis préparant la requête $qry
2 $pdo->beginTransaction();
3 if ($pdo->exec($qry) === FALSE) {
4     $pdo->rollback();
5 }
6 $pdo->commit();
```

CM PHP : Standardisation

PHP Framework Interoperability Group (PHP-FIG)

Groupe de travail de type W3C/IETF visant à

- Normaliser le développement PHP.
- Harmoniser les pratiques.
- Faciliter la prise en main de cadriels.

Emet des normes (PHP Standard Recommendation - PSR)

- **PSR-1, PSR-12** : Style de codage.
- **PSR-3** : Interfaces d'enregistrement d'historique (logging).
- **PSR-4** : Auto-chargement et espaces de noms.
- **PSR-7** : Interfaces de représentation des messages HTTP.
- **PSR-11** : Conteneur d'injection de dépendances.
- ...

Conventions de codage (PSR-1,PSR-12)

Faciliter la prise en main de codes sources développés par d'autres programmeurs

- Organisation et contenu des fichiers.
- Déclaration d'espaces de noms.
- Nommage des variables, fonctions, classes,
- Indentation et dispositions des blocs (`if-else`, ...).
- Documentation

Conventions de codage (PSR-1,PSR-12)

Chaque fichier

- Est encodé en UTF-8 sans BOM (indicateur d'ordre des octets).
- SOIT déclare des symboles (fonctions, classes, ...), SOIT est à effet de bord (inclusion de fichiers, affichage, modification des réglages de `php.ini`, accès à ressources externes, ...)

Classes (interfaces, traits)

Une par fichier déclarée dans un espace de noms conforme à **PSR-4**

Règles de nommage :

- StudlyCaps pour les classes, interfaces et traits.
- MA_CONSTANTE pour les constantes de classes.
- camelCase() pour les méthodes de classes.
- Format libre pour les propriétés et le reste (eg. `ma_fonction`).

Documentation

A quoi sert la documentation du code ?

Expliquer au programmeur qui ne connaît pas le code ce que :

- Contiennent les variables.
- Font les classes/méthodes/fonctions : rôle, comportement, paramètres en entrée/sortie, valeur de retour.

Génération de documentation avec PhpDoc

Outil PhpDoc

- Pendant de Javadoc pour PHP
- Permet de générer une documentation HTML à partir de fichiers sources PHP.
- Par insertion de DocBlocks dans le code source.
- En voie de normalisation (drafts PSR-5, PSR-19)

Générer la documentation (phpdoc_example)

```
$ phpdoc -s -d sourcedir -t targetdir
```

Format des commentaires

Balises génériques

On peut introduire des mots-clés (*tags*) au sein des commentaires qui seront interprétés pour structurer la documentation HTML :

- @author : nom des auteurs.
- @copyright : information de copyright.
- @deprecated : élément obsolète.
- @license : URL / nom.
- @link : URL / description.
- @version : description.

Format des commentaires

Autres tags

- `@param` : type d'argument.
- `@return` : type de valeur retour.
- `@see` : référence.
- `@todo` : description.
- `@var` : type de variable de classe.

Commentaire des classes

comment-class.php

```
1 /**
2 * classe utilisée pour représenter une personne
3 * une personne est définie par son nom sous forme de
4 * chaîne de caractères
5 * @source 1
6 */
7 class Personne
8 {
9     /**
10     * nom de la personne
11     *
12     * @var string
13     */
14     protected $nom;
15 }
```

Commentaire des fonctions

comment-function.php

```
1 /**
2 * calcul de la moyenne
3 * on calcule la moyenne d'un ensemble de notes donné
4 *
5 * @param array $arr_notes tableau de notes entières
6 * @return float moyenne des notes
7 */
8 function moyenne($arr_notes)
9 {
10    $nbr = count($arr_notes);
11    $somme = 0;
12    // parcours séquentiel du tableau suivant la clé entière
13    for ($i = 0; $i < $nbr; ++ $i) {
14        $somme += $arr_notes[$i];
15    }
16    return $somme / $nbr;
17}
```

CM PHP : Gestion de paquets PHP

Espace de noms (alias namespaces - NS)

Pendant en programmation de la notion de répertoire

- Un NS regroupe des classes, fonctions et/ou constantes.
- On peut imbriquer des NS.
- Il existe un NS global - c'est le NS par défaut si on n'en définit pas.
- On peut faire référence à une classe/fonction/constante
 - de manière absolue : `new \someNS\Foo();`
 - relativement au NS courant : `new subNS\Foo();`
 - ou par son nom sans qualificatif : `new Foo();`

Avantages des espaces de noms

- Evite les collisions de noms.
- Facilite l'autochargement et donc portabilité et déploiement.
- Répond aux normes du développement logiciel (Web et autre).

Définition d'espaces de noms

Un NS se définit avec le mot-clé `namespace`

- En début de fichier (seul `declare ...` peut précéder).
- Suivi éventuellement d'accolades englobant les éléments visés.
- Défini éventuellement dans plusieurs fichiers (eg. plusieurs classes).
- Ne pas préfixer par l'antislash.
- Eventuellement imbriqué avec l'antislash comme délimiteur.

NS global

- Sans nom.
- C'est le NS par défaut si l'on n'en définit aucun.
- Contient notamment tous les éléments du langage PHP.

Espace de noms : règles de résolution

Les règles de résolution par rapport au NS courant (global ou non)

\$obj = new Foo(); (*unqualified*)

=> currentnamespace\Foo **OU** Foo

\$obj = new subnamespace\Foo(); (*qualified*)

=> currentnamespace\subnamespace\Foo **OU** subnamespace\Foo

\$obj = new \currentnamespace\Foo(); (*fully-qualified*)

=> currentnamespace\Foo

Définition d'espaces de noms

Accès

- A l'espace de nom avec constante `__NAMESPACE__`
- Aux noms de classes pleinement qualifiés (FQCN) avec mot-clé `class`, eg. `MaClasse::class`

namespace-definition.php

```
1 namespace Cours;
2 class MaClasse {
3     function __toString() {
4         return 'NS : ' . __NAMESPACE__ . ' -- FQCN : ' . MaClasse::class . '<br/>';
5     }
6 }
7 function maFonction() {
8     return 'NS : ' . __NAMESPACE__ . ' -- FQN : ' . __FUNCTION__ . '<br/>';
9 }
10 const MA_CONST = 'NS : ' . __NAMESPACE__ . '<br/>';
```

Utilisation d'espaces de noms

On peut faire référence aux éléments d'un NS

- De manière absolue : en préfixant le NS de l'élément par l'antislash

namespace-utilisation.php

```
1 require 'namespace-definition.php';
2 class MaClasse {
3     function __toString() {
4         return 'NS : ' . __NAMESPACE__ . ' -- FQCN : ' . MaClasse::class . '<br/>';
5     }
6 }
7 function maFonction() {
8     return 'NS : ' . __NAMESPACE__ . ' -- FQN : ' . __FUNCTION__ . '<br/>';
9 }
10 const MA_CONST = __NAMESPACE__ . '<br/>';
11 $x = new \Cours\MaClasse();
12 echo $x; //NS:Cours--FQCN:Cours\MaClasse
13 $x = new MaClasse();
14 echo $x; //NS:--FQCN:MaClasse
15 echo \Cours\maFonction(); //NS:--FQN:Cours\maFonction
16 echo maFonction(); //NS:--FQN:maFonction
17 echo \Cours\MA_CONST; //NS:Cours
18 echo MA_CONST; //
```

Utilisation d'espaces de noms

On peut faire référence aux éléments du NS courant

- De manière absolue, directe (sans qualificatif) ou avec la commande namespace

namespace-utilisation-1.php

```
1 namespace Tutorial;
2 require 'namespace-definition.php';
3 class MaClasse {
4     function __toString() {
5         return 'NS : ' . __NAMESPACE__ . ' -- FQCN : ' . MaClasse::class . '<br/>';
6     }
7 }
8 $x = new \Cours\MaClasse();
9 echo $x; // NS : Cours -- FQCN : Cours\MaClasse
10 // Plusieurs alternatives pour référer aux éléments courants
11 // relative
12 $x = new MaClasse();
13 echo $x; // NS : Tutorial -- FQCN : Tutorial\MaClasse
14 // absolue
15 $x = new \Tutorial\MaClasse();
16 echo $x; // NS : Tutorial -- FQCN : Tutorial\MaClasse
17 // avec commande 'namespace'
18 $x = new
19     namespace \MaClasse();
```

Import et aliasing

Un NS ou tout élément de NS

- S'importe avec le mot-clé `use`
- Peut être renommé avec le mot-clé `as`

namespace-import-alias.php

```
1 namespace Tutorial;
2 require 'namespace-definition.php';
3 use Cours as Course;
4 $x = new Course\MaClasse(); echo $x; // NS : Cours -- FQCN : Cours\MaClasse
5 use Cours\MaClasse as MyClass;
6 $x = new MyClass(); echo $x; // NS : Cours -- FQCN : Cours\MaClasse
```

Espace de noms : exemple récapitulatif

namespaces.php

```
1 namespace foo;
2 use My\Full\Classname as Another;
3 $obj = new Another(); // instantiates object of class My\Full\Classname
4 $obj = new namespace\Another(); // instantiates object of class foo\Another
5 // this is the same as use My\Full\NSname as NSname
6 use My\Full\NSname;
7 NSname\subns\func(); // calls function My\Full\NSname\subns\func
8 // importing a global class
9 use ArrayObject;
10 $a = new ArrayObject(array(1)); // instantiates object of class ArrayObject
11 // without "use ArrayObject" we would instantiate an object of class foo\ArrayObject
12 // importing a function
13 use function My\Full\functionName;
14 // aliasing a function
15 use function My\Full\functionName as func;
16 func(); // calls function My\Full\functionName
17 // importing a constant
18 use const My\Full\CONSTANT;
19 echo CONSTANT; // echoes the value of My\Full\CONSTANT
```

Autochargement (alias autoloading)

Permet l'import automatique de fichiers

- Evite d'importer nommément les fichiers de classes.
- Mise en oeuvre par appel de `spl_autoload_register` avec une fonction de chargement (`spl_autoload` par défaut) :
 - Cette dernière sera appelée en cas d'utilisation d'une classe `x` non déclarée.
 - Et tentera d'inclure un fichier sur la base du nom de `x`.
- Permet d'exploiter les espaces de noms et leur mise en correspondance avec une structure de répertoire.
- Normalisé dans **PSR-4** et systématiquement utilisé depuis la généralisation de Composer pour l'installation de modules tiers.

Exemple d'autochargement

CamelCase ↔ ./CamelCase.php (spl-autoloader.php)

```
1 spl_autoload_register(); // appelle spl_autoload() l'implémentation par défaut de __autoload()  
2 $obj = new UneClasse(); // déclenche : include "UneClasse.php";  
3 // erreur : fichier introuvable  
4 $obj2 = new PasDeClasse(); // déclenche : include "PasDeClasse.php";
```

CamelCase ↔ ./camel_case.php (spl-autoloader1.php)

```
1 spl_autoload_register(function ($class_name) {  
2     $str = strtolower(preg_replace('/(?!^)[A-Z]/', '_$0', $class_name));  
3     include $str . '.php';  
4 });  
5 $obj = new UneClasse(); // déclenche : include "une_classe.php";  
6 // erreur : fichier introuvable  
7 $obj2 = new PasDeClasse(); // déclenche : include "pas_de_classe.php";
```

Extensions et paquets PHP

Extension (alias module)

Librairie à compiler avec PHP CLI ou module PHP du serveur web.

- Typiquement implantée en C.
- Requiert une installation système, eg. via apt
- A activer dans le fichier `php.ini` approprié : CLI et/ou module.

Listing des extensions chargées

- PHP CLI : `$ php -m`
- Module PHP : fonction `get_loaded_extensions()` ou `phpinfo(INFO_MODULES)`.

Certaines extensions sont aussi fournies en paquet (moindres performances).

Extensions et paquets PHP

Paquet de code source PHP (alias package)

- Soit outil/utilitaire : phpunit, phpdoc ...
- Soit librairie : PHP-DI, monolog ...
- Soit framework : Slim, Laravel ...

Installation

- Manuelle : téléchargement d'archive (phar, zip, tar)
 - Une archive PHAR peut être utilisée sans désarchivage, eg.
`php php-unit.phar tests/EmployeeTest`
`php composer.phar list`
- Automatique avec Composer, eg.
`php composer.phar install monolog/monolog`

Composer

Gestionnaire de dépendances par projet

Utilisé en ligne de commande

- Télécharge les paquets à partir d'un dépôt
 - **Packagist** par défaut.
 - Possibilité de télécharger à partir de dépôts privés sous VCS (git/svn/...) accessibles via github, gitlab ...
- Place les paquets dans un répertoire choisi du projet (par défaut vendor).
- Stocke les méta-données d'installation (noms et versions de paquets, ...) dans deux fichiers JSON qui sont à engager dans le VCS.
- Génère un autoloader PSR-4 pour importer les fonctionnalités de paquets par espaces de noms.

Composer : installation et usage

Installation globale

```
$ sudo apt update  
$ sudo apt install wget php-cli php-zip unzip  
$ wget -O composer-setup.php  
https://getcomposer.org/installer  
$ sudo php composer-setup.php --install-dir=/usr/local/bin  
--filename=composer
```

Installation par projet (eg. dans Mes_projets_web)

```
$ mkdir Composer  
$ cd Composer  
$ php -r "copy('https://getcomposer.org/installer',  
'composer-setup.php');"  
$ php composer-setup.php  
--install-dir=$HOME/Mes_projets_web/Composer  
$ php -r "unlink('composer-setup.php');"
```

Composer : exemple d'installation locale de paquet

Installer version 1.22 du module Monolog

```
$ composer require monolog/monolog:1.22
// crée le répertoire vendor
// y installe le(s) paquet(s) - ici monolog version 1.22 - et autoloader
// crée composer.json et composer.lock
```

Utiliser le paquet monolog dans un script

composer-monolog-full.php

```
1 require __DIR__ . '/Composer/vendor/autoload.php'; // autoloader créé/géré par Composer
2 use Monolog\Logger;
3 use Monolog\Handler\StreamHandler;
4 // create a log channel
5 $log = new Logger('name');
6 $log->pushHandler(new StreamHandler(__DIR__.'/my.log', Logger::WARNING));
7 // add records to the log
8 $log->warning('Foo');
9 $log->error('Bar');
```

PEAR et PECL

PEAR (PHP Extension and Application Repository)

- Une librairie structurée de code source libre.
- Un système de distribution et de maintenance des paquets.
- Un style de codage pour les programmes écrits en PHP.
- Un site Web, des listes de diffusion et des sites miroirs pour supporter la communauté PHP/PEAR.

NB Graduellement “supplanté” par Composer.

PECL (PHP Extension Community Library)

- Projet distinct de PEAR pour distribuer les extensions de PHP (code écrit en C et compilé telle que l'extension PDO).
- Les extensions PECL sont aussi distribuées en paquets et peuvent être installées avec l'installeur de PEAR.

CM PHP : Standard PHP Library

Introduction à la SPL

Standard PHP Library

Un ensemble de classes et d'interfaces :

- Structures de données dédiées (liste chaînée, tas, ...).
- Itérateurs.
- Exceptions (débordements, ...).
- Fonctions (réflexion, autochargement, ...).
- Gestion des répertoires et fichiers.
- Autres (gestion des tableaux sous forme d'objets, patron Observer).

Intégrée au langage depuis PHP 5.0.

Pour connaître les classes et interfaces implantées par la SPL

```
1 print_r(spl_classes());
```

Structures de données

Les classes SPL

SplDoublyLinkedList	Liste doublement chaînée.
SplStack	Pile (hérite de SplDoublyLinkedList).
SplQueue	File (hérite de SplDoublyLinkedList).
SplHeap	Tas.
SplMaxHeap	Tas (maximum sauvegardé en haut).
SplMinHeap	Tas (minimum sauvegardé en haut).
SplPriorityQueue	File de priorité.
SplFixedArray	Tableau indicé de taille fixe.
SplObjectStorage	Ensemble ou dictionnaire d'objets.

Itérateurs

Itérateur

Objet permettant de parcourir tous les éléments d'un conteneur
(tableau, liste, pile, file, dictionnaire, ...)

Les classes SPL implémentent

- Des interfaces prédéfinies : Traversable, Iterator, IteratorAggregate, ArrayAccess
- Des interfaces de la SPL : Countable, OuterIterator, RecursiveIterator, SeekableIterator

L'itération sur un conteneur peut se faire avec `foreach` sur un objet itérateur.

Arbre de la classe SPL Iterators

- [ArrayIterator](#)
 - [RecursiveArrayIterator](#)
- [EmptyIterator](#)
- [IteratorIterator](#)
 - [AppendIterator](#)
 - [CachingIterator](#)
 - [RecursiveCachingIterator](#)
- [FilterIterator](#)
 - [CallbackFilterIterator](#)
 - [RecursiveCallbackFilterIterator](#)
 - [RecursiveFilterIterator](#)
 - [ParentIterator](#)
 - [RegexIterator](#)
 - [RecursiveRegexIterator](#)
- [InfinitIterator](#)
- [LimitIterator](#)
- [NoRewindIterator](#)
- [MultipleIterator](#)
- [RecursiveIteratorIterator](#)
 - [RecursiveTreeIterator](#)
- [DirectoryIterator](#) (*extends* [SplFileInfo](#))
 - [FilesystemIterator](#)
 - [GlobIterator](#)

L'interface Iterator

Iterator

- Pour itérateur bidirectionnel utilisable avec constructeur `foreach`.
- Etend `Traversable` (interface “interne” ne pouvant être implémentée seule).

interface-iterator.php

```
1 interface Iterator extends Traversable {  
2     public function current(); // valeur de l'élément courant  
3     public function key(); // clé de l'élément courant  
4     public function next(); // passe à l'élément suivant  
5     public function rewind(); // revient sur le 1er élément  
6     public function valid(); // TRUE si on n'est pas à la fin  
7 }
```

Implémentation et usage avec foreach

interface-iterator-use.php

```
1 class myIterator implements Iterator {
2     private $position = 0;
3     private $array = array("first", "second", "last");
4     public function __construct() { $this->position = 0; }
5     function current() { var_dump(__METHOD__);
6         return $this->array[$this->position];
7     }
8     function key() { var_dump(__METHOD__);
9         return $this->position;
10    }
11    function next() { var_dump(__METHOD__);
12        ++$this->position;
13    }
14    function rewind() { var_dump(__METHOD__);
15        $this->position = 0;
16    }
17    function valid() { var_dump(__METHOD__);
18        return isset($this->array[$this->position]);
19    }
20 }
21 $it = new myIterator();
22 foreach ($it as $key => $value) { var_dump($key, $value); }
```

L'interface ArrayAccess

Permet de manipuler comme un tableau un objet encapsulant un conteneur, eg. accès via `$obj[2]`.

interface-arrayaccess.php

```
1 interface ArrayAccess {  
2     public function offsetExists($offset);  
3     public function offsetSet($offset, $value);  
4     public function offsetGet($offset);  
5     public function offsetUnset($offset);  
6 }
```

Description des méthodes

`offsetExists` détermine si l'indice existe.

`offsetSet` attribue une valeur à l'indice donné.

`offsetGet` retourne la valeur à l'indice donné.

`offsetUnset` supprime l'élément à l'indice donné.

L'interface ArrayAccess

interface-arrayaccess-use.php

```
1 class Obj implements ArrayAccess {
2     private $cnr = array();
3     public function __construct() {
4         $this->cnr = array("un"=>1, "deux"=>2);
5     }
6     public function offsetExists($offset) {
7         return isset($this->cnr[$offset]);
8     }
9     public function offsetGet($offset) {
10        return isset($this->cnr[$offset]) ? $this->cnr[$offset] : null;
11    }
12    public function offsetSet($offset, $value) {
13        if (is_null($offset)) {
14            $this->cnr[] = $value;
15        } else {
16            $this->cnr[$offset] = $value;
17        }
18    }
19    public function offsetUnset($offset) {
20        unset($this->cnr[$offset]);
21    }
22 }
23 $obj = new Obj();
24 var_dump($obj["deux"]);
25 $obj["deux"] = "Une valeur";
26 $obj[] = 'Ajout 1';///<=>$obj[0]<=>$obj["0"]
27 print_r($obj);
```

La classe `ArrayIterator` (SPL)

Permet d'itérer sur un tableau ou un objet et d'en modifier ou supprimer les éléments.

class-arrayiterator.php

```
1 $tab=range ('a','c');
2 $itr=new ArrayIterator ($tab);
3 $itr [0]='z';
4 unset ($itr [1]);
5 //zc
6 foreach ($itr as $key=>$val) { $itr[$key]=strtoupper ($val); }
7 foreach ($itr as $val) { echo $val."\n"; }
```

L'interface IteratorAggregate

Permet à un objet conteneur de fournir un itérateur.

interface-iteratoraggregate.php

```
1 interface IteratorAggregate extends Traversable {  
2     public function getIterator();  
3 }
```

interface-iteratoraggregate-use.php

```
1 class MyContainer implements IteratorAggregate {  
2     protected $tab;  
3     public function __construct() { $this->tab=array(1,2,3); }  
4     public function getIterator() {  
5         return new ArrayIterator($this->tab);  
6     }  
7 }  
8 // conversion en itérateur via appel implicite à getIterator  
9 foreach (new MyContainer() as $value) {  
10    echo $value. "\n";  
11 }
```

Countable et LimitIterator (SPL)

Interface Countable

Permet de connaître le nombre d'éléments d'un objet.

interface-countable.php

```
1 Interface Countable { public function count(); }
```

Classe LimitIterator

Permet de générer un itérateur sur un tableau dont on fixe la plage de valeurs.

class-limititerator.php

```
1 $tab=range('c','r');
2 $itr=new ArrayIterator($tab);
3 $limit=new LimitIterator($itr, 3, 2);
4 //fg
5 foreach ($limit as $val) { echo $val."\n"; }
```

La classe AppendIterator (SPL)

Permet de générer un itérateur sur plusieurs tableaux.

class-appenditerator.php

```
1 $itr1=new ArrayIterator(range(1,5));
2 $itr2=new ArrayIterator(range(10,15));
3 $itr=new AppendIterator();
4 $itr->append($itr1);
5 $itr->append($itr2);
6 //1 2 3 4 5 10 11 12 13 14 15
7 foreach ($itr as $value) {
8     echo $value . " ";
9 }
```

La classe FilterIterator (SPL)

Permet de filtrer les valeurs en redéfinissant la méthode `accept`.

class-filteriterator.php

```
1 class PlusGrandQue12 extends FilterIterator {  
2     public function accept() {  
3         return ($this->current() > 12);  
4     }  
5 }  
6 $itr=new ArrayIterator(range(1,15));  
7 $filter=new PlusGrandQue12($itr);  
8 print_r(iterator_to_array($filter));
```

La classe RegexIterator (SPL)

Permet de filtrer les valeurs en utilisant une expression régulière.

class-regexiterator.php

```
1 $tab=array('pomme', 'abricot',
2             'poire', 'banane', 'pomelos' );
3 $itr=new ArrayIterator($tab);
4 $registrator=new RegexIterator($itr, '/^po/');
5 // [0] => pomme [2] => poire [4] => pomelos
6 print_r(iterator_to_array($registrator));
```

La classe IteratorIterator (SPL)

Permet de créer un itérateur sur toute classe implémentant uniquement Traversable (wrapper).
Utilisée notamment avec PDO.

class-iteratoriterator.php

```
1 $pdoStatement=$db->query('SELECT * FROM table');
2 $itr=new IteratorIterator($pdoStatement);
3 $limit=new LimitIterator($itr,0,10);
4 print_r(iterator_to_array($limit));
```

Fonctions agissant sur les itérateurs

`iterator_apply(iterator, callback)`

Applique une fonction de rappel sur chaque élément parcouru (à la `array_map`).

`iterator_count(iterator)`

Compte le nombre d'éléments de l'itérateur.

`iterator_to_array(iterator)`

Retourne un tableau des éléments de l'itérateur.

Exemple

Exemple (iterator-to-array.php)

```
1 $tableau=array(1,2,3);
2 $iterator=new ArrayIterator($tableau);
3 echo "il y a ". iterator_count($iterator)
4     ." elements\n";
5 print_r(iterator_to_array($iterator));
6 print_r($iterator);
7 /**
8 $it=new InfinitelIterator($iterator);
9 echo $it->current()."\n";
10 $it->next();
11 echo $it->current()."\n";
12 $it->next();
13 echo $it->current()."\n";
14 $it->next();
15 echo $it->current()."\n";
16 $it->next();echo $it->current()."\n";
17 $it->next();
18 echo $it->current()."\n";
19 $it->next();
20 echo $it->current()."\n";
21 $it->next();
```

Exemple

Résultat

```
ArrayIterator Object
(
    [storage:ArrayIterator:private] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 3
        )
)
il y a 3 elements
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
123123
```

La classe `ArrayObject` (SPL)

Permet de gérer un tableau sous forme d'objet.

class-arrayobject.php

```
1 class ArrayObject implements
2     IteratorAggregate, Traversable,
3     ArrayAccess, Countable    {
4 //...
5 }
```

class-arrayobject-use.php

```
1 $tableau=range(2,5);
2 $object=new ArrayObject($tableau);
3 $object->append('hello');
4 $object[2]='a';
5 $object['color']='red';
6 print_r($object->getIterator());
```

La classe ArrayObject

Résultat

```
ArrayIterator Object
(
    [storage:ArrayIterator:private] => ArrayObject Object
        (
            [storage:ArrayObject:private] => Array
                (
                    [0] => 2
                    [1] => 3
                    [2] => a
                    [3] => 5
                    [4] => hello
                    [color] => red
                )
        )
)
```

Classes d'exceptions de SPL

BadFunctionCallException	Fonction ou arguments manquants.
BadMethodCallException	Méthode ou arguments manquants.
DomainException	Une valeur n'est pas du domaine.
InvalidArgumentException	Un argument n'est pas du type attendu.
LengthException	Une taille est invalide.
LogicException	Erreur dans la logique du programme.
OutOfBoundsException	Clé d'accès invalide.
OutOfRangeException	Clé d'accès invalide.
RangeException	Une valeur est hors de la plage attendue.
RuntimeException	Erreur rencontrée à l'exécution.
OverflowException	Ajout à un conteneur plein.
UnderflowException	Suppression dans un conteneur vide.
UnexpectedValueException	Une valeur retour ne fait pas partie d'une liste attendue.

CM PHP : L'architecture MVC

Objectif

Se familiariser avec l'architecture MVC

- Comment est organisée l'architecture MVC ?
- Comment peut-t'on l'appliquer pour le Web ?

Illustration

Au travers d'un site jouet de type "blog" :

- Basé sur le cours de **B. Pesquet (2015)**.
- **Codes sources.**

L'architecture MVC

L'architecture MVC

Le modèle 1

Tous les traitements sont réalisés dans la même page.

Du modèle 1 au MVC

Une plus grande maîtrise du Développement Web requiert le passage du modèle 1 au modèle MVC.

Patron de conception

Patron de conception (alias design pattern)

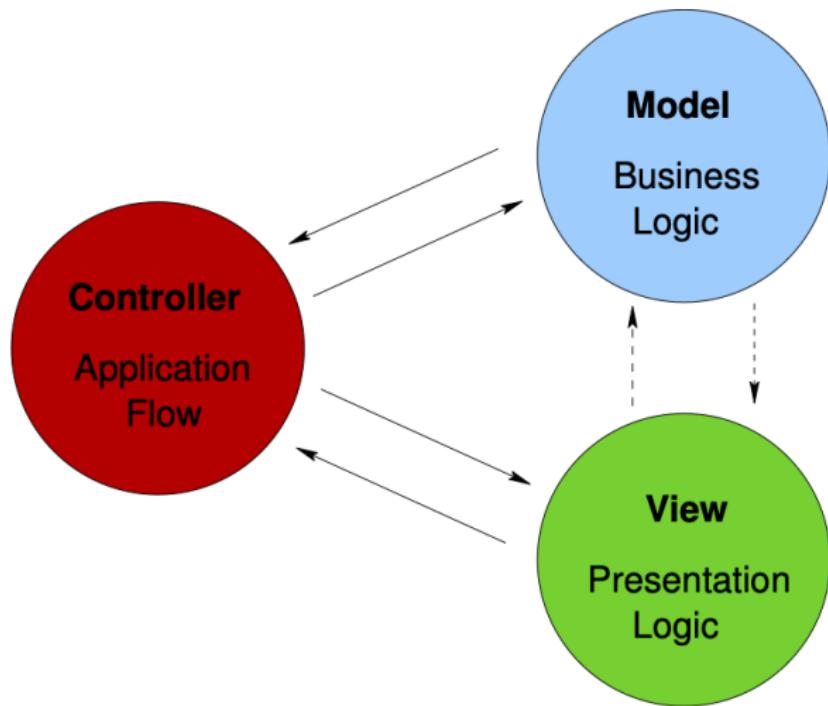
- Décrit une solution standard suivant le paradigme objet pour répondre à un problème récurrent de conception logicielle.
- On encourage les développeurs à les mettre en oeuvre même si c'est parfois contraignant.

L'architecture MVC

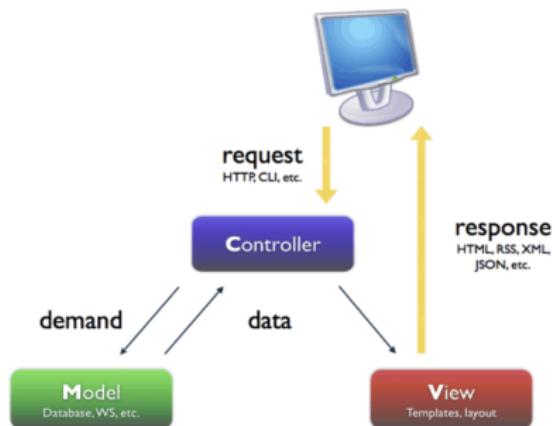
Model View Controller

- Patron de conception orienté objet.
- Elaboré par Trygve Reenskaug en 1979 au Xerox PARC.
- Initialement pour le langage Smalltalk.
- Formalisé par Steve Burbeck.
- Repose sur la séparation des préoccupations (separation of concerns) :
 - **Modèle** : L'accès aux données et la logique métier.
 - **Vue** : L'interaction avec l'utilisateur (présentation, saisie, validation).
 - **Contrôleur** : La dynamique de l'application.
- Différentes interprétations et implantations existent.

MVC schématique



MVC schématique (documentation Symfony)



Les trois parties du MVC

Le modèle

- Accède aux données : création, lecture, mise à jour, destruction (CRUD).
- Met en oeuvre la logique métier (business logic).

La vue

- Concerne l'interaction avec l'utilisateur : présentation des données du modèle, saisie, validation.
- Pour un ou plusieurs périphériques de sortie.

Le contrôleur

- Gère la dynamique de l'application en déterminant les traitements à réaliser en fonction des requêtes de l'utilisateur.
- Réécrit les URL.

Avantages du MVC et difficultés liées au Web

Apports du MVC

- Adaptée aux applications graphiques.
- Facilite l'ingénierie logicielle :
 - Conception modulaire.
 - Activités de développement naturellement distribuées.
 - Maintenance et évolution facilitées.

Difficultés liées au Développement Web

- Prendre en compte la persistence des données.
- Prendre en compte la distribution des objets (cas des applications distribuées).
- La “couche” Contrôleur doit-elle prendre en compte les traitements ?

Persistence et Distribution

Persistence et MVC

La “couche” persistence traite de l’échange de l’information avec les bases de données.

- Certains considèrent qu’elle fait partie du modèle.
- Il est préférable de l’extraire du modèle et de mettre en correspondance modèle relationnel et modèle objet ⇒ technique de l’ORM (**Object Relational Mapping**).

Distribution et MVC

Certaines applications Web sont distribuées.

- Les classes du modèle ne sont pas toutes stockées sur le même serveur.

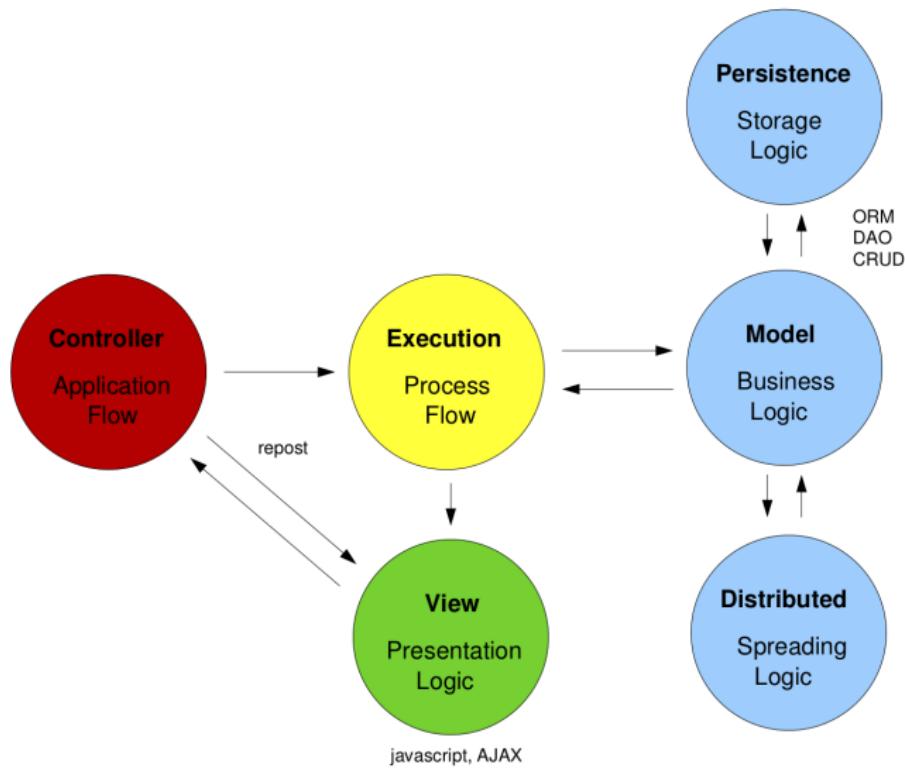
A prendre en compte lors de la conception et l’implantation.

Contrôleur et traitements

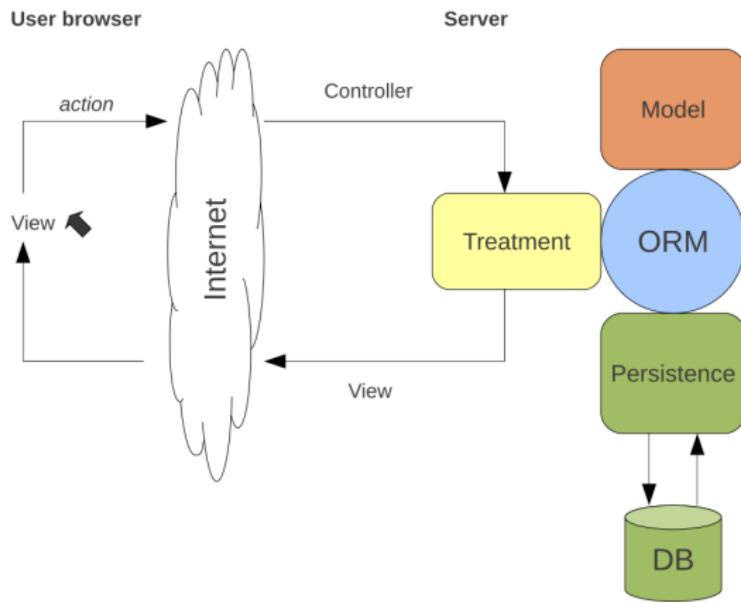
Le contrôleur doit-il réaliser les traitements ?

- C'est le cas pour les applications standards (non Web).
- On peut ajouter une nouvelle partie **Exécution** à l'architecture MVC qui se charge de réaliser les traitements.
- L'objectif est de simplifier l'implantation.
- Nouvelle architecture : MPD-V-CE.

M(PD)-V-C(E) schématique



L'architecture M(PD)-V-C(E) en action



Présentation de l'exemple

Présentation de l'exemple

Mise en oeuvre d'une page Web de type "blog"

- Articles (billet) et commentaires sur articles.

Base de données

- Une table pour stocker les articles.
- Une table pour stocker les commentaires.

monblog T_BILLET	
⌚	BIL_ID : int(11)
⌚	BIL_DATE : datetime
⌚	BIL_TITRE : varchar(100)
⌚	BIL_CONTENU : varchar(400)

monblog T_COMMENTAIRE	
⌚	COM_ID : int(11)
⌚	COM_DATE : datetime
⌚	COM_AUTEUR : varchar(100)
⌚	COM_CONTENU : varchar(200)
#	BIL_ID : int(11)

Page principale

index.php

- Ecrite en HTML5 : usage de la balise article, ...
- Affichage
 - Abrégé avec : <?= ...?>
 - Plutôt que <php echo ...?>
- Utilisation de PDO pour accès à la BD.

Critiques

Limites de l'approche

- Mélange balises HTML et code PHP.
- Structure monolithique difficile à maintenir et faire évoluer.

Principe de responsabilité unique

Décomposer l'application en sous-parties ayant chacune un rôle et une responsabilité particulière :

- **Interactions utilisateur** : affichage, saisie et validation des données.
- **Données** : accès aux données manipulées et stockage.
- **Traitements** : opérations sur les données en lien avec les règles métiers.

Mise en place d'une architecture MVC simple

Isolation de l'affichage et de l'accès aux données

Méthode

- ➊ Placer le code de présentation dans un fichier dédié `vueAccueil.php`.
- ➋ Placer une fonction d'accès BDD aux billets dans un fichier dédié `Modele.php` :
 - `getBillets()`
- ➌ Lier présentation et accès BDD dans le fichier `index.php` via
 - require "Modele.php" au début.
 - require "vueAccueil.php" à la fin.

Factorisation des éléments d'affichage communs

`gabarit.php`

Un gabarit (template)

- Contient les éléments communs aux pages constituant un site web : en-tête, pied de page, menu. . . .
- Permet l'ajout d'éléments spécifiques à chaque vue.

`vueAccueil.php`

Définition des éléments spécifiques `$titre` et `$contenu`

- ① Crédit et mise en tampon d'un flux HTML contenant la liste des articles avec `ob_start` et `ob_get_clean`.
- ② Initialisation de `$contenu` avec ce flux puis inclusion ("instantiation") du gabarit.

Factorisation de la connexion à la BDD et gestion des erreurs

Ajout d'une fonction de connexion `getBdd()` à la BD dans `Modele.php`

- Instancie et renvoie l'objet PDO.
- Réutilisable par toute fonction exécutant des requêtes.

Gestion des erreurs par le contrôleur (`index.php`)

- ① Signaler les erreurs de connexion dans `getBdd()`.
- ② Ajouter l'affichage d'erreur à `index.php` en réutilisant le gabarit par import d'un fichier dédié `vueErreur.php`.
- ③ Ajouter un `try/catch` pour importer `vueAccueil.php` ou bien `vueErreur.php`.

Ajouter de nouvelles fonctionnalités

De manière méthodique

- ① Ecriture des fonctions d'accès aux données dans le modèle.
- ② Création d'une nouvelle vue utilisant le gabarit.
- ③ Ajout d'une page contrôleur pour lier modèle et vue.

Exemple : affichage des détails d'un billet

Cliquer sur le titre d'un billet doit afficher son contenu et les commentaires sur une nouvelle page

- ① Ajouter à `Model.php` deux fonctions `getBillet($id)` et `getCommentaires($id)`.
- ② Créer une vue `vueBillet.php` qui repose sur des variables fournies par le contrôleur.
- ③ Créer un contrôleur `billet.php` qui
 - Importe le modèle.
 - Reçoit l'identifiant du billet en paramètre (HTTP GET).
 - Récupère billet et commentaire par le biais du modèle importé.
 - Importe la vue.
- ④ Ajouter dans `vueAccueil.php` un lien vers `billet.php` paramétré avec l'id du billet pour chaque billet.

```

index.php
1 <?php
2
3 require 'Model.php';
4
5 try {
6     $billets = getBillets();
7     require 'vueAccueil.php';
8 }
9 catch (Exception $e) {
10    $msgErreur = $e->getMessage();
11    require 'vueErreur.php';
12 }

Model.php
1 <?php
2
3 // Renvoie la liste des billets du blog
4 function getBillets() {
5     $bdd = getBdd();
6     $billets = $bdd->query('select BIL_ID as id, BIL_DATE as date,
7         . BIL_TITRE as titre, BIL_CONTENU as contenu from T_BILLET'
8         . ' order by BIL_ID desc');
9     return $billets;
10 }
11
12 // Renvoie les informations sur un billet
13 function getBillet($idBillet) []
14
15 // Renvoie la liste des commentaires associés à un billet
16 function getCommentaires($idBillet) []
17
18 // Effectue la connexion à la BDD
19 // Instancie et renvoie l'objet PDO associé
20 function getBdd() []

billet.php
1 <?php
2
3 require 'Model.php';
4
5 try {
6     if (isset($_GET['id'])) {
7         // intval renvoie la valeur numérique du paramètre ou 0 en cas d'échec
8         $id = intval($_GET['id']);
9         if ($id != 0) {
10             $billet = getBillet($id);
11             $commentaires = getCommentaires($id);
12             require 'vueBillet.php';
13         }
14     }
15     else
16         throw new Exception("Identifiant de billet incorrect");
17 }
18
19 catch (Exception $e) {
20    $msgErreur = $e->getMessage();
21    require 'vueErreur.php';
22 }

vueAccueil.php
1 <?php $titre = 'Mon Blog'; ?>
2
3 <?php ob_start(); ?>
4 <?php foreach ($billets as $billet): ?>
5 <article>
6 <header>
7 <a href="?> billet.php?id=" . $billet['id'] ?>>>
8     <h1 class="titreBillet"><?= $billet['titre'] ?></h1>
9 </>
10 <time><?= $billet['date'] ?></time>
11 </header>
12 <p><?= $billet['contenu'] ?></p>
13 </article>
14 <br />
15 <?php endforeach; ?>
16 <?php $contenu = ob_get_clean(); ?>
17
18 <?php require 'gabarit.php'; ?>

gabarit.php
1 <!doctype html>
2 <html lang="fr">
3 <head>
4     <meta charset="UTF-8" />
5     <link rel="stylesheet" href="style.css" />
6     <title><?= $titre ?></title>
7 </head>
8 <body>
9     <div id="global">
10        <header>
11            <a href="index.php"><?= $titre ?> Mon Blog </a></h1></p>
12            <p>Je vous souhaite la bienvenue sur ce modeste blog.</p>
13        </header>
14        <div id="contenu">
15            <?= $contenu ?>
16            </div> <!-- #contenu -->
17            <footer id=" piedBlog " >
18                Blog réalisé avec PHP, HTML5 et CSS.
19            </footer>
20            </div> <!-- #global -->
21        </body>
22    </!>
```

```

vueBillet.php
1 <?php $titre = 'Mon Blog - ' . $billet['titre']; ?>
2
3 <?php ob_start(); ?>
4 <article>
5 <header>
6     <h1 class="titreBillet"><?= $billet['titre'] ?></h1>
7     <time><?= $billet['date'] ?></time>
8 </header>
9     <p><?= $billet['contenu'] ?></p>
10 </article>
11 <br />
12 <header>
13     <h1 id="titreReponses">Réponses à <?= $billet['titre'] ?></h1>
14 </header>
15 <?php foreach ($commentaires as $commentaire): ?>
16     <p><?= $commentaire['auteur'] ?> dit :</p>
17     <p><?= $commentaire['contenu'] ?></p>
18 <?php endforeach; ?>
19 <?php $contenu = ob_get_clean(); ?>
20
21 <?php require 'gabarit.php'; ?>
```

Bilan

Modèle MVC simple

- Sépare vue, modèle et contrôleur.
- Permet d'ajouter de nouvelles fonctionnalités de manière méthodique.

Limites de l'architecture

- Contrôleurs indépendants rendant difficile l'usage de politiques communes : authentification, sécurité ...
- Les noms de fichiers sont exposés dans les hyperliens.
- Le code reste procédural.

Etape suivante

- Introduire un vocable d'actions et un contrôleur frontal.
- Passer à un modèle objet.

Passage à une architecture orientée objet avec contrôleur frontal

Mise en oeuvre d'un contrôleur frontal

Introduction d'un contrôleur frontal

- Constitue le point d'entrée unique du site.
- Centralise la gestion des requêtes de l'utilisateur.
- Utilise les autres contrôleurs pour réaliser l'action demandée et renvoyer la vue.

Rôles du contrôleur frontal

- ① Analyse la requête entrante : demande d'affichage de la liste des billets ou demande d'un billet précis.
- ② Vérifie les paramètres fournis.
- ③ Déclenche l'*action* à réaliser.
- ④ Signale l'erreur à l'utilisateur en cas d'incohérence.

Les actions

Objectif

Masquer la structure du site en utilisant un vocable d'actions désignant les différentes requêtes possibles.

- Seul le fichier du contrôleur frontal est visible dans les URL.
- Les URL pour utiliser le site ne changent pas et sont indépendantes de l'organisation en répertoires.

Mise en oeuvre

- Associer à chaque requête un nom d'action et, selon la requête, des paramètres.
- Modifier les hyperliens dans les fichiers de vue pour utiliser ces actions.
- Implémenter chaque action sous forme de méthode dans un contrôleur.

Passage à des contrôleurs orienté objet

Le contrôleur frontal

- Classe `Routeur` instanciée par `index.php`.
- Le `Routeur` construit et stocke les contrôleurs dédiés.
- Il analyse, vérifie, et aiguille la requête entrante vers le bon contrôleur en déclenchant la méthode appropriée.

Répartir les actions dans des contrôleurs dédiés

- Classe `ControleurAccueil` : gère la page d'accueil.
- Classe `ControleurBillet` : gère l'affichage d'un billet.

Chaque contrôleur

- Instancie la classe modèle requise pour récupérer les données.
- Instancie la classe `Vue` avec l'action (`string`) puis appelle la méthode `Vue::generer` avec les données.

Passage à un modèle orienté objet

Classes métiers et superclasse

- Créer des classes “métiers” modélisant les entités du domaine :
 - Billet (*billet*)
 - Commentaire (*commentaires*)
- Regrouper les services communs (connexion à la BD, exécution d'une requête SQL passée sous forme de chaîne de caractères) dans une superclasse commune `Modele`.

Remarques

- L'accès à la BD est masqué aux classes concrètes.
- La superclasse peut retarder l'instanciation de l'objet PDO à sa première utilisation (*lazy loading*).

Passage à une vue orientée objet

Classe de génération de vue

Objectif : centraliser la mise en tampon de(s) flux HTML.

- Créer une classe `Vue` qui générera chaque vue.
- L'affichage d'une vue se fera par instantiation de cette classe puis invocation de la méthode `Vue::generer($donnees)`.

Instantiation

- L'objet `Vue` est construit par le contrôleur qui lui communique l'action à réaliser.
- Il en déduit le fichier vue à utiliser.

Passage à une vue orientée objet

Méthode `Vue:::generer($donnees)`

- Construit et stocke les variables communes (`$titre`, `$contenu`) dans un tableau associatif.
- Génère la partie spécifique de la vue puis le gabarit en appelant la méthode `generer_fichier()` avec le nom de fichier et le tableau de données.

Passage à une vue orientée objet

Méthode `Vue::generer_fichier($fichier, $donnees)`

- Vérifie l'existence du fichier.
- Extrait les données du tableau sous forme de variables par la fonction PHP `extract`.
- Met le flux HTML dans le tampon de sortie (`ob_start()`).
- Inclut le fichier.
- Renvoie le tampon de sortie (`ob_get_clean()`).

```

index.php
1 <?php
2
3 require 'Controleur/Routeur.php';
4
5 $routeur = new Routeur();
6 $routeur->routerRequete();
7

Routeur.php
1 <?php
2
3 require_once 'Controleur/ControleurAccueil.php';
4 require_once 'Controleur/ControleurBillet.php';
5 require_once 'Vue/Vue.php';
6
7 class Routeur {
8
9     private $ctrlAccueil;
10    private $ctrlBillet;
11
12    public function __construct() {
13        $this->ctrlAccueil = new ControleurAccueil();
14        $this->ctrlBillet = new ControleurBillet();
15    }
16
17    // Route une requête entrante : exécution l'action associée
18    public function routerRequete() {
19        try {
20            if (isset($_GET['action'])) {
21                if ($_GET['action'] == 'Billet') {
22                    $idBillet = intval($this->getParametre($_GET, 'id'));
23                    if ($idBillet != 0) {
24                        $this->ctrlBillet->billet($idBillet);
25                    } else
26                        throw new Exception("Identifiant de billet non valide");
27                }
28                else if ($_GET['action'] == 'commenter') {
29                    $auteur = $this->getParametre($_POST, 'auteur');
30                    $contenu = $this->getParametre($_POST, 'contenu');
31                    $idBillet = $this->getParametre($_POST, 'id');
32                    $this->ctrlBillet->commenter($auteur, $contenu, $idBillet);
33                }
34                else
35                    throw new Exception("Action non valide");
36            }
37            else { // aucune action définie : affichage de l'accueil
38                $this->ctrlAccueil->accueil();
39            }
40        }
41        catch (Exception $e) {
42            $this->erreur($e->getMessage());
43        }
44    }
45
46    // Affiche une erreur
47    private function erreur($msgErreur) {
48        $vue = new Vue("Erreur");
49        $vue->generer(array('msgErreur' => $msgErreur));
50    }
51
52    // Recherche un paramètre dans un tableau
53    private function getParametre($tableau, $nom) {
54        if (isset($tableau[$nom])) {
55            return $tableau[$nom];
56        }
57        else
58            throw new Exception("Paramètre '$nom' absent");
59    }
60

ControleurAccueil.php
1 <?php
2
3 require_once 'Modele/Billet.php';
4 require_once 'Vue/Vue.php';
5
6 class ControleurAccueil {
7
8     private $billet;
9
10    public function __construct() {
11        $this->billet = new Billet();
12    }
13
14    // Affiche la liste de tous les billets du blog
15    public function accueil() {
16        $billets = $this->billet->getBillets();
17        $vue = new Vue("Accueil");
18        $vue->generer(array('billets' => $billets));
19    }
20
21 }

ControleurBillet.php
1 <?php
2
3 require_once 'Modele/Billet.php';
4 require_once 'Modele/Commentaire.php';
5 require_once 'Vue/Vue.php';
6
7 class ControleurBillet {
8
9     private $billet;
10    private $commentaire;
11
12    public function __construct() {
13        $this->billet = new Billet();
14        $this->commentaire = new Commentaire();
15    }
16
17    // Affiche les détails sur un billet
18    public function billet($idBillet) {
19        $billet = $this->billet->getBillet($idBillet);
20        $commentaires = $this->commentaire->getCommentaires($idBillet);
21        $vue = new Vue("Billet");
22        $vue->generer(array('billet' => $billet, 'commentaires' => $commentaires));
23    }
24
25    // Ajoute un commentaire à un billet
26    public function commenter($auteur, $contenu, $idBillet) {
27        // Sauvegarde du commentaire
28        $this->commentaire->ajouterCommentaire($auteur, $contenu, $idBillet);
29        // Actualisation de l'affichage du billet
30        $this->billet($idBillet);
31
32    }
33
34
35

```

```
Model.php
```

```
1@ abstract class Modele {
2
3    /** Objet PDO d'accès à la BD */
4    private $bdd;
5
6    * Exécute une requête SQL éventuellement paramétrée
7    protected function executerRequete($sql, $params = null) {
8        if ($params == null) {
9            $resultat = $this->getBdd()->query($sql); // exécution directe
10       }
11       else {
12           $resultat = $this->getBdd()->prepare($sql); // requête préparée
13           $resultat->execute($params);
14       }
15       return $resultat;
16   }
17
18   * Renvoie un objet de connexion à la BD en initialisant la connexion au besoin
19   private function getBdd() {
20       if ($this->bdd == null) {
21           // Création de la connexion
22           $this->bdd = new PDO('mysql:host=localhost;dbname=l3_crs_blog;charset=utf8',
23                               'lesaint', 'lesaint',
24                               array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
25       }
26       return $this->bdd;
27   }
28 }
```

```
Commentaire.php <-->
1 <?php
2
3 require_once 'Modele/Modele.php';
4 /**
5 * Fournit les services d'accès aux genres musicaux
6 *
7 * @author Baptiste Pesquet
8 */
9 class Commentaire extends Modele {
10
11 // Renvoie la liste des commentaires associés à un billet
12 public function getCommentaires($idBillet) {
13     $sql = 'select COM_ID as id, COM_DATE as date,
14             . COM_AUTEUR as auteur, COM_CONTENU as contenu from T_COMMENTAIRE'
15     . ' where BIL_ID=?';
16     $commentaires = $this->executerRequete($sql, array($idBillet));
17     return $commentaires;
18 }
19
20 // Ajoute un commentaire dans la base
21 public function ajouterCommentaire($auteur, $contenu, $idBillet) {
22     $sql = 'insert into T_COMMENTAIRE(COM_DATE, COM_AUTEUR, COM_CONTENU, BIL_ID)'
23     . ' values (?, ?, ?, ?)';
24     $date = date('Y-m-d H:i:s'); // date(@DATE_W3C); // Récupère la date courante
25     $this->executerRequete($sql, array($date, $auteur, $contenu, $idBillet));
26 }
```

```
1 <?php
2
3 require_once 'Modele/Modele.php';
4
5 /**
6 * Fournit les services d'accès aux genres musicaux
7 *
8 * @author Baptiste Pesquet
9 */
10 class Billet extends Modele {
11
12     /** Renvoie la liste des billets du blog */
13     public function getBillets() {
14         $sql = 'select BIL_ID as id, BIL_DATE as date,
15                 . BIL_TITRE as titre, BIL_CONTENU as contenu from T_BILLET'
16         . ' order by BIL_ID desc';
17         $billets = $this->executerRequete($sql);
18         return $billets;
19     }
20
21     /** Renvoie les informations sur un billet */
22     public function getBillet($idBillet) {
23         $sql = 'select BIL_ID as id, BIL_DATE as date,
24                 . BIL_TITRE as titre, BIL_CONTENU as contenu from T_BILLET'
25         . ' where BIL_ID=?';
26         $billet = $this->executerRequete($sql, array($idBillet));
27         if ($billet->rowCount() > 0)
28             return $billet->fetch();
29         else
30             throw new Exception("Aucun billet ne correspond à l'identifiant '$idBillet'");
31     }
32 }
```

```

P Vue.php ✘
1 <?php
2
3 class Vue
4 {
5
6 // Nom du fichier associé à la vue
7 private $fichier;
8
9 // Titre de la vue (défini dans le fichier vue)
10 private $titre;
11
12 public function __construct($section)
13 {
14 // Détermination du nom du fichier vue à partir de l'action
15 $this->fichier = "Vue/vue". $section . ".php";
16 }
17
18 // Génère et affiche la vue
19 public function generer($donnees)
20 {
21 // Génération de la partie spécifique de la vue
22 $contenu = $this->genererFichier($this->fichier, $donnees);
23 // Génération du gabarit commun utilisant la partie spécifique
24 $vue = $this->genererFichier("Vue/gabarit.php", array(
25 'titre' => $this->titre,
26 'contenu' => $contenu
27));
28 // Renvoi de la vue au navigateur
29 echo $vue;
30 }
31
32 // Génère un fichier vue et renvoie le résultat produit
33 private function genererFichier($fichier, $donnees)
34 {
35 if (file_exists($fichier)) {
36 // Rend les éléments du tableau $donnees accessibles dans la vue
37 extract($donnees);
38 // Démarrage de la temporisation de sortie
39 ob_start();
40 // Inclut le fichier vue
41 // Son résultat est placé dans le tampon de sortie
42 require $fichier;
43 // Arrêt de la temporisation et renvoi du tampon de sortie
44 return ob_get_clean();
45 } else {
46 throw new Exception("Fichier '$fichier' introuvable");
47 }
48 }
49 }
```

```

P vueBillet.php ✘
1 <?php $this->titre = "Mon Blog - ". $billet['titre']; ?>
2
3 <article>
4 <header>
5 <h1 class="titreBillet"><?= $billet['titre'] ?></h1>
6 <time><?= $billet['date'] ?></time>
7 </header>
8 <p><?= $billet['contenu'] ?></p>
9 </article>
10 <hr />
11 <header>
12 <h1 id="titreReponses">Réponses à <?= $billet['titre'] ?></h1>
13 </header>
14 <?php foreach ($commentaires as $commentaire): ?>
15 <p><?= $commentaire['auteur'] ?> dit :</p>
16 <p><?= $commentaire['contenu'] ?></p>
17 <?php endforeach; ?>
18 <hr />
19 <form method="post" action="index.php?action=commenter">
20 <input id="auteur" name="auteur" type="text" placeholder="Votre pseudo"
21 required /><br />
22 <textarea id="txtCommentaire" name="contenu" rows="4"
23 placeholder="Votre commentaire" required></textarea><br />
24 <input type="hidden" name="id" value=<?= $billet['id'] ?> />
25 <input type="submit" value="Commenter" />
26 </form>
```

```

P gabarit.php ✘
1 <!doctype html>
2 <html lang="fr">
3 <head>
4 <meta charset="UTF-8" />
5 <link rel="stylesheet" href="Contenu/style.css" />
6 <title><?= $titre ?></title>
7 </head>
8 <body>
9 <div id="global">
10 <header>
11 <a href="index.php"><h1 id="titreBlog">Mon Blog</h1></a>
12 <p>Je vous souhaite la bienvenue sur ce modeste blog.</p>
13 </header>
14 <div id="contenu">
15 <?= $contenu ?>
16 </div>
17 <?= $contenu -->
18 <footer id=" piedBlog" >Blog réalisé avec PHP, HTML5 et CSS. </fo
19 </div>
20 <!-- #global -->
21 </body>
```

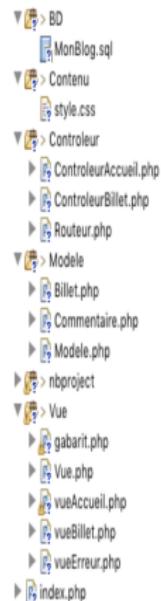
```

P vueErreur.php ✘
1 <?php $this->titre = "Mon Blog - Erreur !"; ?>
2
3 <p><?= $msgErreur ?></p>
4
```

Réorganisation des fichiers source

Pour gagner en lisibilité, on regroupe les différents types de fichiers dans des répertoires dédiés

- BD : le script de création de la BD.
- Contenu : fichiers ressources “statiques” (feuilles CSS, images, ...).
- Contrôleur : contrôleurs.
- Modèle : modèles.
- Vue : gabarit et vues.
- index.php : contrôleur frontal à la racine.



Prise en compte de nouvelles fonctionnalités

Par ajout ou spécialisation

- ① De la classe modèle associée.
- ② D'une vue utilisant le gabarit.
- ③ D'une classe contrôleur pour lier modèle et vue.

Exemple : ajout d'un commentaire sur un billet

- 1 Ajouter une méthode `ajouterCommentaire` à la classe `Commentaire` réalisant l'insertion SQL.
- 2 Ajouter le formulaire de saisie d'un commentaire à la vue `vueBillet.php` en créant l'action `commenter`.
- 3 Mettre à jour `style.css` (eg. pour dimensionner `textarea`).
- 4 Ajouter à la classe `ControlleurBillet` une méthode `commenter($auteur, $contenu, $billet)` associée à la nouvelle action qui stocke le commentaire et réactualise la page.
- 5 Mettre à jour la méthode de routage dans la classe `Routeur` pour intégrer cette action.

Construction d'un framework MVC

Les frameworks

Un framework de type MVC

- Fournit un ensemble de services de base sous la forme de classes prédefinies
 - Routage des requêtes.
 - Sécurité.
 - Gestion du cache ...
- L'utilisateur du framework (développeur) peut se focaliser sur les classes métiers de son application.

Frameworks PHP

Laravel, Symfony, Zend Framework, CodeIgniter, CakePHP, Slim ...

Apport d'un framework à notre architecture MVC

Limites de l'architecture précédente

- Non-configurabilité des paramètres d'accès à la BD.
- Multiplication des objets de connexion PDO (un par classe modèle).
- Illisibilité des URL

`monsite.fr/index.php?action=yyy&id=zzz`

vs

`monsite.fr/action/id`

- Routage “manuel” des requêtes par le routeur.
- Aucun filtrage sur les paramètres des requêtes.
- Aucun nettoyage des données insérées dans les vues (risques de failles XSS).

Accès générique aux données

Classe abstraite Modele

- On externalise les paramètres d'accès à la BD à l'aide d'une classe statique Configuration.php :
 - Lit un fichier de configuration dev.ini ou prod.ini
 - Avec la fonction parse_ini(\$file) : renvoie un tableau associatif des propriétés.
- On fait de l'objet PDO un attribut statique.

Automatisation du routage de la requête

Classe Requete

Contient les paramètres de la requête (auxquels on peut rajouter en-têtes HTTP, session, ...).

Instanciée en début de routage pour modéliser la requête.

Ajout d'un paramètre "contrôleur" aux URL en plus de l'action et paramètres d'actions :

- monsite.fr/index.php?controleur=xxx&action=yyy&id=zzz

Permet au Routeur de déduire de la requête quels

- Contrôleur instancier : `creerControleur($requete)`.
- Méthode invoquer : `creerAction($requete)`.

Puis d'invoquer : `$controleur->executerAction($action)`.

Usage de variable dynamique nommant la classe à instancier.

Controleur

Classe abstraite Controleur

- Contient l'action à réaliser et la requête.
- La méthode `executerAction ($action)` utilise la réflexion pour déduire quelle méthode de classe fille invoquer.
- La méthode `genererVue ($donnees)` détermine le nom du fichier Vue à importer sur la base du nom de contrôleur utilisé.

Contraintes de nommage

Correspondances strictes à établir entre

- Nom d'action, nom de classe contrôleur, nom de fichier de classe.
- Nom de contrôleur et nom de fichier vue.

Mise en place d'URL génériques

Format standard d'URL

Pour une meilleure lisibilité et faciliter le référencement, on utilise un format “standard” d'URL pour les pages du site :

- `monsite.fr/controleur/action/id` remplace
`monsite.fr/index.php?controleur=xxx&action=yyy&id=zzz`
- Exemple : `monsite.fr/billet/index/2`

Réécriture d'URL

Le module `mod_rewrite` de Apache et le fichier de configuration `.htaccess` placé à la racine du site permettent de réécrire les URL avant chargement des scripts.

- Inclure l'élément `<base href="racineDuSite"/>` dans toutes les vues pour résoudre les liens relatifs.

<code>Requete.php</code>	<code>Modele.php</code>	<code>Routeur.php</code>
<pre> 1 <?php 2 3 /** 4 * Classe modélisant une requête HTTP entrante 5 * 6 * @version 1.0 7 * @author Baptiste Pesquet 8 */ 9 class Requete { 10 11 /** Tableau des paramètres de la requête */ 12 private \$parametres; 13 14 * Constructeur 15 public function __construct(\$parametres) { 16 \$this->parametres = \$parametres; 17 } 18 19 * Renvoie vrai si le paramètre existe dans la requête 20 public function existeParametre(\$nom) { 21 22 * Renvoie la valeur du paramètre demandé 23 public function getParametre(\$nom) { 24 25 } 26 27 /** 28 * Tableau des paramètres de la requête 29 * @param array \$parametres 30 */ 31 32 /** 33 * Renvoie la valeur d'un paramètre de configuration 34 * @param string \$nom, \$valeurParDefaut = null 35 */ 36 37 /** 38 * Renvoie le tableau des paramètres en les chargeant au besoin depuis le fichier configuration 39 */ 40 public static function getParametres() { 41 42 } 43 44 }</pre>	<pre> 1 <?php 2 3 require_once 'Configuration.php'; 4 5 /** 6 * Classe abstraite Modèle 7 */ 8 abstract class Modèle { 9 10 /** 11 * Objet PDO d'accès à la BD 12 * @private static \$bdd; 13 14 /** 15 * Exécute une requête SQL 16 * @protected function exécuterRequête(\$sql, \$params = null) 17 18 /** 19 * Renvoie un objet de connexion à la BD en initialisant la connexion 20 * @private static function getBdd() 21 22 */ 23 24 /** 25 * Instancie le contrôleur approprié en fonction de la requête reçue 26 * @private function créerContrôleur(Requête \$requete) 27 28 /** 29 * Détermine l'action à exécuter en fonction de la requête reçue 30 * @private function créerAction(Requête \$requete) 31 32 /** 33 * Gère une erreur d'exécution (exception) 34 * @private function gérerErreur(Exception \$exception) 35 36 */ 37 38 /** 39 * Méthode principale appelée par le contrôleur frontal 40 * @public function routerRequête() 41 42 */ 43 44 /** 45 * Détermine l'action à exécuter en fonction de la requête reçue 46 * @private function créerAction(Requête \$requete) 47 48 */ 49 50 /** 51 * Gère une erreur d'exécution (exception) 52 * @private function gérerErreur(Exception \$exception) 53 54 */ 55 56 }</pre>	
<code>Configuration.php</code>	<code>Contrôleur.php</code>	<code>Vue.php</code>
<pre> 1 <?php 2 3 /** 4 * Classe de gestion des paramètres de configuration 5 */ 6 class Configuration { 7 8 /** 9 * Tableau des paramètres de configuration 10 * @private static \$parametres; 11 12 /** 13 * Renvoie la valeur d'un paramètre de configuration 14 * @param string \$nom, \$valeurParDefaut = null 15 */ 16 17 /** 18 * Renvoie le tableau des paramètres en les chargeant au besoin depuis le fichier configuration 19 */ 20 21 /** 22 * Renvoie la vue associée au contrôleur courant 23 */ 24 25 /** 26 * Tableau des paramètres de la requête 27 */ 28 29 /** 30 * Constructeur 31 * @param array \$parametres 32 */ 33 34 /** 35 * Action à réaliser 36 */ 37 38 /** 39 * Requête entrante 40 */ 41 42 /** 43 * Définit la requête entrante 44 * @param Requête \$requete 45 */ 46 47 /** 48 * Exécute l'action à réaliser 49 */ 50 51 /** 52 * Méthode abstraite correspondant à l'action par défaut 53 * @public abstract function index(); 54 55 /** 56 * Génère la vue associée au contrôleur courant 57 */ 58 59 /** 60 * Génère une vue pour une action 61 */ 62 63 /** 64 * Nettoie une valeur insérée dans une page HTML 65 */ 66 67 /** 68 * Génère un fichier vue et renvoie le résultat produit 69 */ 70 71 /** 72 * Génère un fichier vue et renvoie le résultat produit 73 */ 74 75 /** 76 * Nettoie une valeur insérée dans une page HTML 77 */ 78 79 /** 80 * Génère un fichier vue et renvoie le résultat produit 81 */ 82 83 /** 84 * Nettoie une valeur insérée dans une page HTML 85 */ 86 87 /** 88 * Génère un fichier vue et renvoie le résultat produit 89 */ 90 91 /** 92 * Nettoie une valeur insérée dans une page HTML 93 */ 94 95 /** 96 * Génère un fichier vue et renvoie le résultat produit 97 */ 98 99 /** 100 * Nettoie une valeur insérée dans une page HTML 101 */ 102 103 /** 104 * Génère un fichier vue et renvoie le résultat produit 105 */ 106 107 /** 108 * Nettoie une valeur insérée dans une page HTML 109 */ 110 111 /** 112 * Génère un fichier vue et renvoie le résultat produit 113 */ 114 115 /** 116 * Nettoie une valeur insérée dans une page HTML 117 */ 118 119 /** 120 * Génère un fichier vue et renvoie le résultat produit 121 */ 122 123 /** 124 * Nettoie une valeur insérée dans une page HTML 125 */ 126 127 /** 128 * Génère un fichier vue et renvoie le résultat produit 129 */ 130 131 /** 132 * Nettoie une valeur insérée dans une page HTML 133 */ 134 135 /** 136 * Génère un fichier vue et renvoie le résultat produit 137 */ 138 139 /** 140 * Nettoie une valeur insérée dans une page HTML 141 */ 142 143 /** 144 * Génère un fichier vue et renvoie le résultat produit 145 */ 146 147 /** 148 * Nettoie une valeur insérée dans une page HTML 149 */ 150 151 /** 152 * Génère un fichier vue et renvoie le résultat produit 153 */ 154 155 /** 156 * Nettoie une valeur insérée dans une page HTML 157 */ 158 159 /** 160 * Génère un fichier vue et renvoie le résultat produit 161 */ 162 163 /** 164 * Nettoie une valeur insérée dans une page HTML 165 */ 166 167 /** 168 * Génère un fichier vue et renvoie le résultat produit 169 */ 170 171 /** 172 * Nettoie une valeur insérée dans une page HTML 173 */ 174 175 /** 176 * Génère un fichier vue et renvoie le résultat produit 177 */ 178 179 /** 180 * Nettoie une valeur insérée dans une page HTML 181 */ 182 183 /** 184 * Génère un fichier vue et renvoie le résultat produit 185 */ 186 187 /** 188 * Nettoie une valeur insérée dans une page HTML 189 */ 190 191 /** 192 * Génère un fichier vue et renvoie le résultat produit 193 */ 194 195 /** 196 * Nettoie une valeur insérée dans une page HTML 197 */ 198 199 /** 199 * Génère un fichier vue et renvoie le résultat produit 200 */ 201 202 /** 203 * Nettoie une valeur insérée dans une page HTML 204 */ 205 206 /** 207 * Génère un fichier vue et renvoie le résultat produit 208 */ 209 210 /** 211 * Nettoie une valeur insérée dans une page HTML 212 */ 213 214 /** 215 * Génère un fichier vue et renvoie le résultat produit 216 */ 217 218 /** 219 * Nettoie une valeur insérée dans une page HTML 220 */ 221 222 /** 223 * Génère un fichier vue et renvoie le résultat produit 224 */ 225 226 /** 227 * Nettoie une valeur insérée dans une page HTML 228 */ 229 230 /** 231 * Génère un fichier vue et renvoie le résultat produit 232 */ 233 234 /** 235 * Nettoie une valeur insérée dans une page HTML 236 */ 237 238 /** 239 * Génère un fichier vue et renvoie le résultat produit 240 */ 241 242 /** 243 * Nettoie une valeur insérée dans une page HTML 244 */ 245 246 /** 247 * Génère un fichier vue et renvoie le résultat produit 248 */ 249 250 /** 251 * Nettoie une valeur insérée dans une page HTML 252 */ 253 254 /** 255 * Génère un fichier vue et renvoie le résultat produit 256 */ 257 258 /** 259 * Nettoie une valeur insérée dans une page HTML 260 */ 261 262 /** 263 * Génère un fichier vue et renvoie le résultat produit 264 */ 265 266 /** 267 * Nettoie une valeur insérée dans une page HTML 268 */ 269 270 /** 271 * Génère un fichier vue et renvoie le résultat produit 272 */ 273 274 /** 275 * Nettoie une valeur insérée dans une page HTML 276 */ 277 278 /** 279 * Génère un fichier vue et renvoie le résultat produit 280 */ 281 282 /** 283 * Nettoie une valeur insérée dans une page HTML 284 */ 285 286 /** 287 * Génère un fichier vue et renvoie le résultat produit 288 */ 289 289 /** 290 * Nettoie une valeur insérée dans une page HTML 291 */ 292 293 /** 294 * Génère un fichier vue et renvoie le résultat produit 295 */ 296 297 /** 298 * Nettoie une valeur insérée dans une page HTML 299 */ 299 299 /** 300 * Génère un fichier vue et renvoie le résultat produit 301 */ 301 302 /** 303 * Nettoie une valeur insérée dans une page HTML 304 */ 305 306 /** 307 * Génère un fichier vue et renvoie le résultat produit 308 */ 309 309 /** 310 * Nettoie une valeur insérée dans une page HTML 311 */ 312 313 /** 314 * Génère un fichier vue et renvoie le résultat produit 315 */ 316 317 /** 318 * Nettoie une valeur insérée dans une page HTML 319 */ 320 321 /** 322 * Génère un fichier vue et renvoie le résultat produit 323 */ 324 325 /** 326 * Nettoie une valeur insérée dans une page HTML 327 */ 328 329 /** 330 * Génère un fichier vue et renvoie le résultat produit 331 */ 332 333 /** 334 * Nettoie une valeur insérée dans une page HTML 335 */ 336 337 /** 338 * Génère un fichier vue et renvoie le résultat produit 339 */ 339 340 /** 341 * Nettoie une valeur insérée dans une page HTML 342 */ 343 344 /** 345 * Génère un fichier vue et renvoie le résultat produit 346 */ 347 348 /** 349 * Nettoie une valeur insérée dans une page HTML 350 */ 351 352 /** 353 * Génère un fichier vue et renvoie le résultat produit 354 */ 355 356 /** 357 * Nettoie une valeur insérée dans une page HTML 358 */ 359 359 /** 360 * Génère un fichier vue et renvoie le résultat produit 361 */ 361 362 /** 363 * Nettoie une valeur insérée dans une page HTML 364 */ 365 366 /** 367 * Génère un fichier vue et renvoie le résultat produit 368 */ 369 369 /** 370 * Nettoie une valeur insérée dans une page HTML 371 */ 372 373 /** 374 * Génère un fichier vue et renvoie le résultat produit 375 */ 376 377 /** 378 * Nettoie une valeur insérée dans une page HTML 379 */ 380 381 /** 382 * Génère un fichier vue et renvoie le résultat produit 383 */ 384 385 /** 386 * Nettoie une valeur insérée dans une page HTML 387 */ 388 389 /** 390 * Génère un fichier vue et renvoie le résultat produit 391 */ 392 393 /** 394 * Nettoie une valeur insérée dans une page HTML 395 */ 396 397 /** 398 * Génère un fichier vue et renvoie le résultat produit 399 */ 399 399 /** 400 * Nettoie une valeur insérée dans une page HTML 401 */ 402 403 /** 404 * Génère un fichier vue et renvoie le résultat produit 405 */ 406 407 /** 408 * Nettoie une valeur insérée dans une page HTML 409 */ 410 411 /** 412 * Génère un fichier vue et renvoie le résultat produit 413 */ 414 415 /** 416 * Nettoie une valeur insérée dans une page HTML 417 */ 418 419 /** 420 * Génère un fichier vue et renvoie le résultat produit 421 */ 422 423 /** 424 * Nettoie une valeur insérée dans une page HTML 425 */ 426 427 /** 428 * Génère un fichier vue et renvoie le résultat produit 429 */ 430 431 /** 432 * Nettoie une valeur insérée dans une page HTML 433 */ 434 435 /** 436 * Génère un fichier vue et renvoie le résultat produit 437 */ 438 439 /** 440 * Nettoie une valeur insérée dans une page HTML 441 */ 442 443 /** 444 * Génère un fichier vue et renvoie le résultat produit 445 */ 446 447 /** 448 * Nettoie une valeur insérée dans une page HTML 449 */ 450 451 /** 452 * Génère un fichier vue et renvoie le résultat produit 453 */ 454 455 /** 456 * Nettoie une valeur insérée dans une page HTML 457 */ 458 459 /** 460 * Génère un fichier vue et renvoie le résultat produit 461 */ 462 463 /** 464 * Nettoie une valeur insérée dans une page HTML 465 */ 466 467 /** 468 * Génère un fichier vue et renvoie le résultat produit 469 */ 470 471 /** 472 * Nettoie une valeur insérée dans une page HTML 473 */ 474 475 /** 476 * Génère un fichier vue et renvoie le résultat produit 477 */ 478 479 /** 480 * Nettoie une valeur insérée dans une page HTML 481 */ 482 483 /** 484 * Génère un fichier vue et renvoie le résultat produit 485 */ 486 487 /** 488 * Nettoie une valeur insérée dans une page HTML 489 */ 490 491 /** 492 * Génère un fichier vue et renvoie le résultat produit 493 */ 494 495 /** 496 * Nettoie une valeur insérée dans une page HTML 497 */ 498 499 /** 499 * Génère un fichier vue et renvoie le résultat produit 500 */ 500 500 /** 501 * Nettoie une valeur insérée dans une page HTML 502 */ 503 504 /** 505 * Génère un fichier vue et renvoie le résultat produit 506 */ 507 508 /** 509 * Nettoie une valeur insérée dans une page HTML 510 */ 511 512 /** 513 * Génère un fichier vue et renvoie le résultat produit 514 */ 515 516 /** 517 * Nettoie une valeur insérée dans une page HTML 518 */ 519 519 /** 520 * Génère un fichier vue et renvoie le résultat produit 521 */ 522 523 /** 524 * Nettoie une valeur insérée dans une page HTML 525 */ 526 527 /** 528 * Génère un fichier vue et renvoie le résultat produit 529 */ 530 531 /** 532 * Nettoie une valeur insérée dans une page HTML 533 */ 534 535 /** 536 * Génère un fichier vue et renvoie le résultat produit 537 */ 538 539 /** 539 * Nettoie une valeur insérée dans une page HTML 540 */ 541 542 /** 543 * Génère un fichier vue et renvoie le résultat produit 544 */ 545 546 /** 547 * Nettoie une valeur insérée dans une page HTML 548 */ 549 549 /** 550 * Génère un fichier vue et renvoie le résultat produit 551 */ 552 553 /** 554 * Nettoie une valeur insérée dans une page HTML 555 */ 556 557 /** 558 * Génère un fichier vue et renvoie le résultat produit 559 */ 560 561 /** 562 * Nettoie une valeur insérée dans une page HTML 563 */ 564 565 /** 566 * Génère un fichier vue et renvoie le résultat produit 567 */ 568 569 /** 569 * Nettoie une valeur insérée dans une page HTML 570 */ 571 572 /** 573 * Génère un fichier vue et renvoie le résultat produit 574 */ 575 576 /** 577 * Nettoie une valeur insérée dans une page HTML 578 */ 579 579 /** 580 * Génère un fichier vue et renvoie le résultat produit 581 */ 582 583 /** 584 * Nettoie une valeur insérée dans une page HTML 585 */ 586 587 /** 588 * Génère un fichier vue et renvoie le résultat produit 589 */ 590 591 /** 592 * Nettoie une valeur insérée dans une page HTML 593 */ 594 595 /** 596 * Génère un fichier vue et renvoie le résultat produit 597 */ 598 599 /** 599 * Nettoie une valeur insérée dans une page HTML 600 */ 601 602 /** 603 * Génère un fichier vue et renvoie le résultat produit 604 */ 605 606 /** 607 * Nettoie une valeur insérée dans une page HTML 608 */ 609 609 /** 610 * Génère un fichier vue et renvoie le résultat produit 611 */ 612 613 /** 614 * Nettoie une valeur insérée dans une page HTML 615 */ 616 617 /** 618 * Génère un fichier vue et renvoie le résultat produit 619 */ 620 621 /** 622 * Nettoie une valeur insérée dans une page HTML 623 */ 624 625 /** 626 * Génère un fichier vue et renvoie le résultat produit 627 */ 628 629 /** 629 * Nettoie une valeur insérée dans une page HTML 630 */ 631 632 /** 633 * Génère un fichier vue et renvoie le résultat produit 634 */ 635 636 /** 637 * Nettoie une valeur insérée dans une page HTML 638 */ 639 639 /** 640 * Génère un fichier vue et renvoie le résultat produit 641 */ 642 643 /** 644 * Nettoie une valeur insérée dans une page HTML 645 */ 646 647 /** 648 * Génère un fichier vue et renvoie le résultat produit 649 */ 650 651 /** 652 * Nettoie une valeur insérée dans une page HTML 653 */ 654 655 /** 656 * Génère un fichier vue et renvoie le résultat produit 657 */ 658 659 /** 659 * Nettoie une valeur insérée dans une page HTML 660 */ 661 662 /** 663 * Génère un fichier vue et renvoie le résultat produit 664 */ 665 666 /** 667 * Nettoie une valeur insérée dans une page HTML 668 */ 669 669 /** 670 * Génère un fichier vue et renvoie le résultat produit 671 */ 672 673 /** 674 * Nettoie une valeur insérée dans une page HTML 675 */ 676 677 /** 678 * Génère un fichier vue et renvoie le résultat produit 679 */ 680 681 /** 682 * Nettoie une valeur insérée dans une page HTML 683 */ 684 685 /** 686 * Génère un fichier vue et renvoie le résultat produit 687 */ 688 689 /** 689 * Nettoie une valeur insérée dans une page HTML 690 */ 691 692 /** 693 * Génère un fichier vue et renvoie le résultat produit 694 */ 695 696 /** 697 * Nettoie une valeur insérée dans une page HTML 698 */ 699 699 /** 700 * Génère un fichier vue et renvoie le résultat produit 701 */ 702 703 /** 704 * Nettoie une valeur insérée dans une page HTML 705 */ 706 707 /** 708 * Génère un fichier vue et renvoie le résultat produit 709 */ 710 711 /** 712 * Nettoie une valeur insérée dans une page HTML 713 */ 714 715 /** 716 * Génère un fichier vue et renvoie le résultat produit 717 */ 718 719 /** 719 * Nettoie une valeur insérée dans une page HTML 720 */ 721 722 /** 723 * Génère un fichier vue et renvoie le résultat produit 724 */ 725 726 /** 727 * Nettoie une valeur insérée dans une page HTML 728 */ 729 729 /** 730 * Génère un fichier vue et renvoie le résultat produit 731 */ 732 733 /** 734 * Nettoie une valeur insérée dans une page HTML 735 */ 736 737 /** 738 * Génère un fichier vue et renvoie le résultat produit 739 */ 740 741 /** 742 * Nettoie une valeur insérée dans une page HTML 743 */ 744 745 /** 746 * Génère un fichier vue et renvoie le résultat produit 747 */ 748 749 /** 749 * Nettoie une valeur insérée dans une page HTML 750 */ 751 752 /** 753 * Génère un fichier vue et renvoie le résultat produit 754 */ 755 756 /** 757 * Nettoie une valeur insérée dans une page HTML 758 */ 759 759 /** 760 * Génère un fichier vue et renvoie le résultat produit 761 */ 762 763 /** 764 * Nettoie une valeur insérée dans une page HTML 765 */ 766 767 /** 768 * Génère un fichier vue et renvoie le résultat produit 769 */ 770 771 /** 772 * Nettoie une valeur insérée dans une page HTML 773 */ 774 775 /** 776 * Génère un fichier vue et renvoie le résultat produit 777 */ 778 779 /** 779 * Nettoie une valeur insérée dans une page HTML 780 */ 781 782 /** 783 * Génère un fichier vue et renvoie le résultat produit 784 */ 785 786 /** 787 * Nettoie une valeur insérée dans une page HTML 788 */ 789 789 /** 790 * Génère un fichier vue et renvoie le résultat produit 791 */ 792 793 /** 794 * Nettoie une valeur insérée dans une page HTML 795 */ 796 797 /** 798 * Génère un fichier vue et renvoie le résultat produit 799 */ 799 799 /** 800 * Nettoie une valeur insérée dans une page HTML 801 */ 802 803 /** 804 * Génère un fichier vue et renvoie le résultat produit 805 */ 806 807 /** 808 * Nettoie une valeur insérée dans une page HTML 809 */ 810 811 /** 812 * Génère un fichier vue et renvoie le résultat produit 813 */ 814 815 /** 815 * Nettoie une valeur insérée dans une page HTML 816 */ 817 818 /** 819 * Génère un fichier vue et renvoie le résultat produit 820 */ 821 822 /** 823 * Nettoie une valeur insérée dans une page HTML 824 */ 825 826 /** 827 * Génère un fichier vue et renvoie le résultat produit 828 */ 829 829 /** 830 * Nettoie une valeur insérée dans une page HTML 831 */ 832 833 /** 834 * Génère un fichier vue et renvoie le résultat produit 835 */ 836 837 /** 838 * Nettoie une valeur insérée dans une page HTML 839 */ 840 841 /** 842 * Génère un fichier vue et renvoie le résultat produit 843 */ 844 845 /** 846 * Nettoie une valeur insérée dans une page HTML 847 */ 848 849 /** 849 * Génère un fichier vue et renvoie le résultat produit 850 */ 851 852 /** 853 * Nettoie une valeur insérée dans une page HTML 854 */ 855 856 /** 857 * Génère un fichier vue et renvoie le résultat produit 858 */ 859 859 /** 860 * Nettoie une valeur insérée dans une page HTML 861 */ 862 863 /** 864 * Génère un fichier vue et renvoie le résultat produit 865 */ 866 867 /** 868 * Nettoie une valeur insérée dans une page HTML 869 */ 870 871 /** 872 * Génère un fichier vue et renvoie le résultat produit 873 */ 874 875 /** 875 * Nettoie une valeur insérée dans une page HTML 876 */ 877 878 /** 879 * Génère un fichier vue et renvoie le résultat produit 880 */ 881 882 /** 883 * Nettoie une valeur insérée dans une page HTML 884 */ 885 886 /** 887 * Génère un fichier vue et renvoie le résultat produit 888 */ 889 889 /** 889 * Nettoie une valeur insérée dans une page HTML 890 */ 891 892 /** 893 * Génère un fichier vue et renvoie le résultat produit 894 */ 895 896 /** 897 * Nettoie une valeur insérée dans une page HTML 898 */ 899 899 /** 899 * Génère un fichier vue et renvoie le résultat produit 900 */ 901 902 /** 903 * Nettoie une valeur insérée dans une page HTML 904 */ 905 906 /** 907 * Génère un fichier vue et renvoie le résultat produit 908 */ 909 909 /** 910 * Nettoie une valeur insérée dans une page HTML 911 */ 912 913 /** 914 * Génère un fichier vue et renvoie le résultat produit 915 */ 916 917 /** 918 * Nettoie une valeur insérée dans une page HTML 919 */ 920 921 /** 922 * Génère un fichier vue et renvoie le résultat produit 923 */ 924 925 /** 925 * Nettoie une valeur insérée dans une page HTML 926 */ 927 928 /** 929 * Génère un fichier vue et renvoie le résultat produit 930 */ 931 932 /** 933 * Nettoie une valeur insérée dans une page HTML 934 */ 935 936 /** 937 * Génère un fichier vue et renvoie le résultat produit 938 */ 939 939 /** 940 * Nettoie une valeur insérée dans une page HTML 941 */ 942 943 /** 944 * Génère un fichier vue et renvoie le résultat produit 945 */ 946 947 /** 948 * Nettoie une valeur insérée dans une page HTML 949 */ 950 951 /** 952 * Génère un fichier vue et renvoie le résultat produit 953 */ 954 955 /** 956 * Nettoie une valeur insérée dans une page HTML 957 */ 958 959 /** 959 * Génère un fichier vue et renvoie le résultat produit 960 */ 961 962 /** 963 * Nettoie une valeur insérée dans une page HTML 964 */ 965 966 /** 967 * Génère un fichier vue et renvoie le résultat produit 968 */ 969 969 /** 969 * Nettoie une valeur insérée dans une page HTML 970 */ 971 972 /** 973 * Génère un fichier vue et renvoie le résultat produit 974 */ 975 976 /** 977 * Nettoie une valeur insérée dans une page HTML 978 */ 979 979 /** 979 * Génère un fichier vue et renvoie le résultat produit 980 */ 981 982 /** 983 * Nettoie une valeur insérée dans une page HTML 984 */ 985 986 /** 987 * Génère un fichier vue et renvoie le résultat produit 988 */ 989 989 /** 989 * Nettoie une valeur insérée dans une page HTML 990 */ 991 992 /** 993 * Génère un fichier vue et renvoie le résultat produit 994 */ 995 996 /** 997 * Nettoie une valeur insérée dans une page HTML 998 */ 999 999 /** 999 * Génère un fichier vue et renvoie le résultat produit 1000 */ 1001 1002 /** 1003 * Nettoie une valeur insérée dans une page HTML 1004 */ 1005 1006 /** 1007 * Génère un fichier vue et renvoie le résultat produit 1008 */ 1009 1009 /** 1009 * Nettoie une valeur insérée dans une page HTML 1010 */ 1011 1012 /** 1013 * Génère un fichier vue et renvoie le résultat produit 1014 */ 1015 1016 /** 1017 * Nettoie une valeur insérée dans une page HTML 1018 */ 1019 1019 /** 1019 * Génère un fichier vue et renvoie le résultat produit 1020 */ 1021 1022 /** 1023 * Nettoie une valeur insérée dans une page HTML 1024 */ 1025 1025 /** 1025 * Génère un fichier vue et renvoie le résultat produit 1026 */ 1027 1028 /** 1029 * Nettoie une valeur insérée dans une page HTML 1030 */ 1031 1031 /** 1031 * Génère un fichier vue et renvoie le résultat produit 1032 */ 1033 1034 /** 1034 * Nettoie une valeur insérée dans une page HTML 1035 */ 1036 1036 /** 1037 * Génère un fichier vue et renvoie le résultat produit 1038 */ 1039 1039 /** 1039 * Nettoie une valeur insérée dans une page HTML 1040 */ 1041 1042 /** 1043 * Génère un fichier vue et renvoie le résultat produit 1044 */ 1045 1046 /** 1047 * Nettoie une valeur insérée dans une page HTML 1048 */ 1049 1049 /** 1049 * Génère un fichier vue et renvoie le résultat produit 1050 */ 1051 1052 /** 1053 * Nettoie une valeur insérée dans une page HTML 1054 */ 1055 1056 /** 1057 * Génère un fichier vue et renvoie le résultat produit 1058 */ 1059 1059 /** 1059 * Nettoie une valeur insérée dans une page HTML 1060 */ 1061 1062 /** 1063 * Génère un fichier vue et renvoie le résultat produit 1064 */ 1065 1066 /** 1067 * Nettoie une valeur insérée dans une page HTML 1068 */ 1069 1069 /** 1069 * Génère un fichier vue et renvoie le résultat produit 1070 */ 1071 1072 /** 1073 * Nettoie une valeur insérée dans une page HTML 1074 */ 1075 1075 /** 1075 * Génère un fichier vue et renvoie le résultat produit 1076 */ 1077 1078 /** 1079 * Nettoie une valeur insérée dans une page HTML 1080 */ 1081 1081 /** 1081 * Génère un fichier vue et renvoie le résultat produit 1082 */ 1083 1084 /** 1084 * Nettoie une valeur insérée dans une page HTML 1085 */ 1086 1086 /** 1087 * Génère un fichier vue et renvoie le résultat produit 1088 */ 1089 1089 /** 1089 * Nettoie une valeur insérée dans une page HTML 1090 */ 1091 1092 /** 1093 * Génère un fichier vue et renvoie le résultat produit 1094 */ 1095 1096 /** 1097 * Nettoie une valeur insérée dans une page HTML 1098 */ 1099 1099 /** 1099 * Génère un fichier vue et renvoie le résultat produit 1100 */ 1101 1102 /** 1103 * Nettoie une valeur insérée dans une page HTML 1104 */ 1105 1106 /** 1107 * Génère un fichier vue et renvoie le résultat produit 1108 */ 1109 1109 /** 1109 * Nettoie une valeur insérée dans une page HTML 1110 */ 1111 1112 /** 1113 * Génère un fichier vue et renvoie le résultat produit 1114 */ 1115 1116 /** 1117 * Nettoie une valeur insérée dans une page HTML 1118 */ 1119 1119 /** 1119 * Génère un fichier vue et renvoie le résultat produit 1120 */ 1121 1122 /** 1123 * Nettoie une valeur insérée dans une page HTML 1124 */ 1125 1125 /** 1125 * Génère un fichier vue et renvoie le résultat produit 1126 */ 1127 1128 /** 1129 * Nettoie une valeur insérée dans une page HTML 1130 */ 1131 1131 /** 1131 * Génère un fichier vue et renvoie le résultat produit 1132 */ 1133 1134 /** 1134 * Nettoie une valeur insérée dans une page HTML 1135 */ 1136 1136 /** 1137 * Génère un fichier vue et renvoie le résultat produit 1138 */ 1139 1139 /** 1139 * Nettoie une valeur insérée dans une page HTML 1140 */ 1141 1142 /** 1143 * Génère un fichier vue et renvoie le résultat produit 1144 */ 1145 1146 /** 1147 * Nettoie une valeur insérée dans une page HTML 1148 */ 1149 1149 /** 1149 * Génère un fichier vue et renvoie le résultat produit 1150 */ 1151 1152 /** 1153 * Nettoie une valeur insérée dans une page HTML 1154 */ 1155 1156 /** 1157 * Génère un fichier vue et renvoie le résultat produit 1158 */ 1159 1159 /** 1159 * Nettoie une valeur insérée dans une page HTML 1160 */ 1161 1162 /** 1163 * Génère un fichier vue et renvoie le résultat produit 1164 */ 1165 1166 /** 1167 * Nettoie une valeur insérée dans une page HTML 1168 */ 1169 1169 /** 1169 * Génère un fichier vue et renvoie le résultat produit 1170 */ 1171 1172 /** 1173 * Nettoie une valeur insérée dans une page HTML 1174 */ 1175 1175 /** 1175 * Génère un fichier vue et renvoie le résultat produit 1176 */ 1177 1178 /** 1179 * Nettoie une valeur insérée dans une page HTML 1180 */ 1181 1181 /** 1181 * Génère un fichier vue et renvoie le résultat produit 1182 */ 1183 1184 /** 1184 * Nettoie une valeur insérée dans une page HTML 1185 */ 1186 1186 /** 1186 * Génère un fichier vue et renvoie le résultat produit 1187 */ 1188 1188 /** 1188 * Nettoie une valeur insérée dans une page HTML 1189 */ 1189 1189 /** 1189 * Génère un fichier vue et renvoie le résultat produit 1190 */ 1191 1192 /** 1193 * Nettoie une valeur insérée dans une page HTML 1194 */ 1195 1195 /** 1195 * Génère un fichier vue et renvoie le résultat produit 1196 */ 1197 1198 /** 1199 * Nettoie une valeur insérée dans une page HTML 1200 */ 1201 1201 /** 1202 * Génère un fichier vue et renvoie le résultat produit 1203 */ 1204 1205 /** 1206 * Nettoie une valeur insérée dans une page HTML 1207 */ 1208 1208 /** 1208 * Génère un fichier vue et renvoie le résultat produit 1209 */ 1210 1211 /** 1212 * Nettoie une valeur insérée dans une page HTML 1213 */ 1214 1214 /** 1214 * Génère un fichier vue et renvoie le résultat produit 1215 */ 1216 1217 /** 1218 * Nettoie une valeur insérée dans une page HTML 1219 */ 1220 1220 /** 1220 * Génère un fichier vue et renvoie le résultat produit 1221 */ 1222 1223 /** 1224 * Nettoie une valeur insérée dans une page HTML 1225 */ 1226 1226 /** 1226 * Génère un fichier vue et renvoie le résultat produit 1227 */ 1228 1229 /** 1229 * Nettoie une valeur insérée dans une page HTML 1230 */ 1231 1231 /** 1231 * Génère un fichier vue et renvoie le résultat produit 1232 */ 1233 1234 /** 1234 * Nettoie une valeur insérée dans une page HTML 1235 */ 1236 1236 /** 1236 * Génère un fichier vue et renvoie le résultat produit 1237 */ 1238 1239 /** 1239 * Nettoie une valeur insérée dans une page HTML 1240 */ 1241 1241 /** 1242 * Génère un fichier vue et renvoie le résultat produit 1243 */ 1244 1245 /** 1246 * Nettoie une valeur insérée dans une page HTML 1247 */ 1248 1248 /** 1248 * Génère un fichier vue et renvoie le résultat produit 1249 */ 1250 1251 /** 1252 * Nettoie une valeur insérée dans une page HTML 1253 */ 1254 1254 /** 1254 * Génère un fichier vue et renvoie le résultat produit 1255 */ 1256 1257 /** 1257 * Nettoie une valeur insérée dans une page HTML 1</pre>		

Sécurisation des données reçues et affichées

- Utiliser des requêtes SQL paramétrées.
- Nettoyer les valeurs PHP insérées dans les vues à l'aide de `htmlspecialchars()`.

Autres suggestions

- Authentification avec contrôleur dédié pour mode “connecté”.
- Mise en place d’espaces de noms.
- Utiliser l’autochargement de classes.
- Ajouter des mécanismes de validation des données entrantes.
- Intégrer la journalisation d’évènements (logs).
- Utiliser un moteur de gabarits (eg. Twig).
- Utiliser des composants de frameworks.

La Persistance

La couche Persistance (Persistence layer)

Définition (Persistance)

Chargée de gérer les objets persistants, ie. gère les interactions avec une base de données.

Technologies associées

- ORM (Object Relational Mapping).
- DAO (Data Access Object).
- CRUD (Create Retrieve Update Delete).

ORM, CRUD, DAO

Définition (ORM - Object Relational Mapping)

Technique dédiée à la mise en relation entre les attributs de la classe avec les champs des tables de la base de données.

Définition (CRUD - Create Retrieve Update Delete)

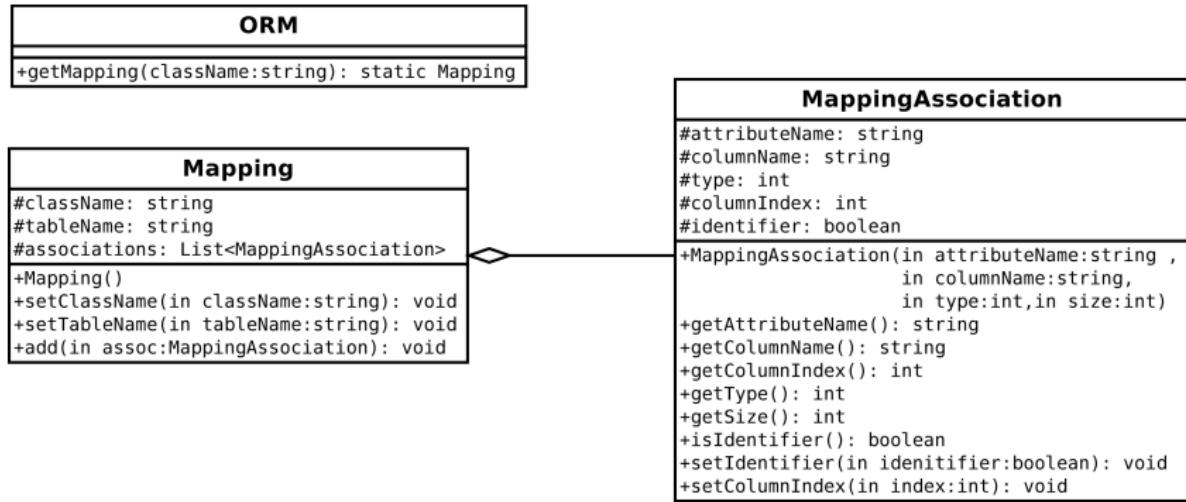
Opérations de base à implanter pour gérer l'échange d'informations entre objets et tables de la base de données.

Le CRUD peut être vu comme une interface.

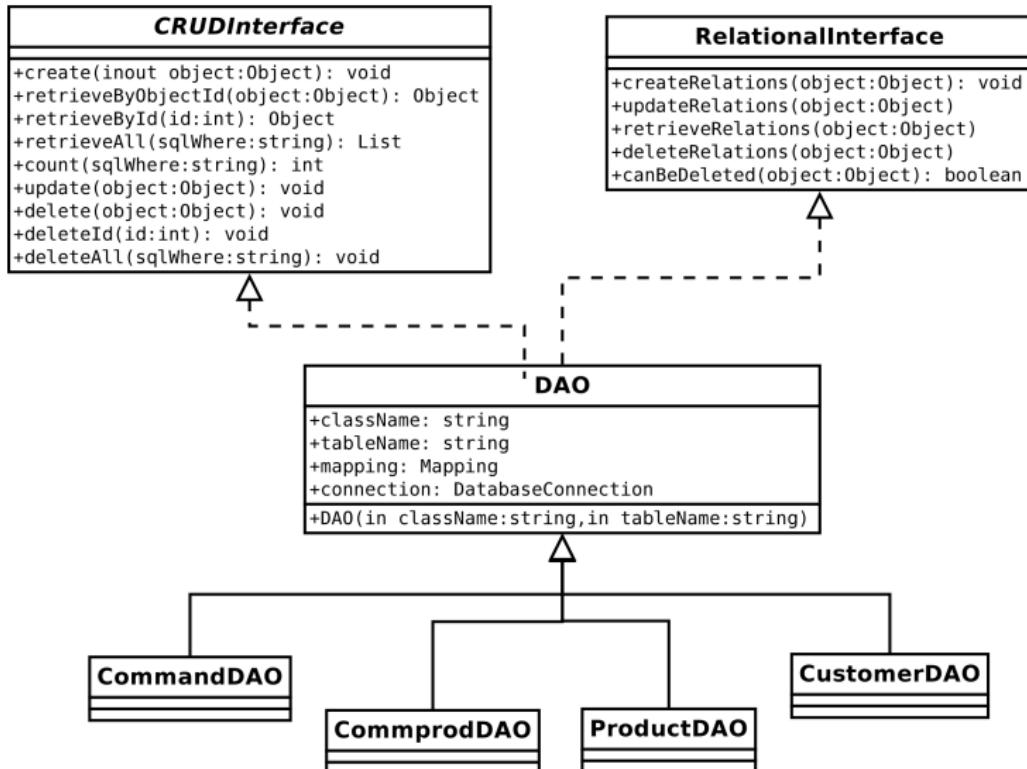
Définition (DAO - Data Access Object)

Implante le CRUD en gérant l'accès à la base de données.

ORM



CRUD et DAO



Fin

Fin

CM PHP : Compléments

Rappels SGML, XML et XSL

Qu'est ce que SGML ?

SGML - Standard Generalized Markup Language

- Langage de description de langages à balises.
- Norme ISO 8879 : 1986.
- Utilisé par les professionnels des domaines de la documentation et de l'édition.
- Assez complexe.
- Séparation du contenu et de la forme.

Qu'est ce que XML ?

XML - eXtensible Markup Language

- Issu de SGML.
- Standard qui permet de représenter l'information de manière structurée de façon et lui donner une sémantique.
- Repose sur un langage de balises prédéfinies.
- Facilite l'échange d'informations entre systèmes hétérogènes.

Avantages et inconvénients de XML

Avantages de XML

- Séparation du contenu et de la forme.
- Extensible.
- Données et méta-données.
- Supporté par les applications Web.
- Fonctions de recherche et transformation.
- Lié à **SOAP** (Simple Object Access Protocol) pour l'invocation de méthodes d'objets distants (Web services).

Inconvénients de XML

- Verbeux (augmentation de la taille des fichiers).
- Consommateur de ressources (représentation en mémoire, processeur).

Exemple de fichier XML

bibliotheque.xml

```
1 <?xml version="1.0" encoding="utf-8" standalone="no"?>
2 <!-- Déclaration externe de DTD -->
3 <!DOCTYPE bibliotheque SYSTEM "bibliotheque.dtd">
4 <!-- Référence à un fichier XSLT pour mise en forme -->
5 <?xmlstylesheet type="text/xsl" href="bibliotheque-liste.xsl" ?>
6 <bibliotheque>
7 <livre>
8   <titre>Apprendre XML en 10 mois</titre>
9   <liste_auteurs>
10    <auteur nom="Escargot" prenom="Jean" />
11    <auteur nom="Snail" prenom="John" />
12   </liste_auteurs>
13   <prix>20</prix>
14 </livre>
15 <livre>
16   <titre>Learn Java in 10 seconds</titre>
17   <liste_auteurs>
18    <auteur nom="Fast" prenom="Bob" />
19   </liste_auteurs>
20   <prix monnaie="dollars">25</prix>
21 </livre>
22 </bibliotheque>
```

Prologue du document XML

bibliotheque.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
```

Première ligne indiquant

- La version de XML utilisée.
- Le jeu de caractères utilisé dans le document (utf-8 recommandé).
- La dépendance du document à un document externe (eg. DTD, XSL) ou non.

Description du fichier XML

Une hiérarchie d'éléments

Contenant des sous-éléments, ou ne contenant que du texte, ou sans contenu (balise auto-fermante).

Elements

- bibliothèque est l'élément racine du document.
- bibliothèque comporte au moins un élément livre.
- Chaque livre comporte titre, liste_auteurs, et prix.
- Chaque liste_auteurs comporte au moins un auteur.
- titre et prix contiennent du texte, auteur est vide.

Attributs

- Chaque auteur a deux attributs nom et prenom.
- L'attribut monnaie (devise) peut être donné pour un prix.

Qu'est ce que XSL ?

XSL - eXtensible Stylesheet Language

Une famille de langages pour la transformation et la présentation de documents XML :

- **XSLT** (XSL Transformations) : langage de transformation de document XML.
- **XPath** (XML Path Language) : langage d'expressions utilisé par XSLT pour désigner une/des parties d'un arbre XML.
- **XSL-FO** (XSL-Formatting Objects) : un vocabulaire XML pour spécifier une sémantique de mise en forme.

XSLT

XSLT

- Permet de transformer un document XML vers tout autre schéma ou format : XHTML, HTML, texte ...
- Se compose d'un analyseur XSLT qui sur la base d'un document XML et d'un document XSLT produit un fichier de sortie au format désiré.

Exemple de fichier XSL - rendu en liste HTML

Exécution via navigateur ou en ligne de commande (`xsltproc`, `xalan`)

```
$ xsltproc bibliotheque.xml bibliotheque-liste.xsl
```

bibliotheque-liste.xsl

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3 <xsl:output method="html"/>
4 <xsl:template match="/">
5   <html>
6     <body bgcolor="#FFFFFF">
7       <ul>
8         <xsl:for-each select="bibliotheque/livre">
9           <li>
10             <xsl:value-of select="titre" />,
11             <xsl:for-each select="liste-auteurs/auteur">
12               <i>
13                 <xsl:value-of select=".//@prenom" /><xsl:text> </xsl:text>
14                 <xsl:value-of select=".//@nom" />
15               </i>,
16             </xsl:for-each>
17             prix <xsl:value-of select="prix" /><xsl:text> </xsl:text>
18             <xsl:value-of select="prix/@monnaie" />,
19           </li>
20         </xsl:for-each>
21       </ul>
22     </body>
23   </html>
24 </xsl:template>
```

Rendu sous forme de liste

- Apprendre XML en 10 mois,
Jean Escargot, Brad Snail,
prix 20
- Learn Java in 10 seconds,
John Fast,
prix 25 dollars

Exemple de fichier XSL - rendu en table HTML

Exécution via navigateur ou en ligne de commande (`xsltproc`, `xalan`)

```
$ xsltproc bibliotheque.xml bibliotheque-table.xsl
```

bibliotheque-table.xsl

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3 <xsl:output method="html"/>
4 <xsl:template match="/">
5   <html>
6     <body>
7       <table border="1" cellpadding="4">
8         <xsl:for-each select="bibliotheque/livre">
9           <tr valign="top">
10             <td><xsl:value-of select="titre"/></td>
11             <td><xsl:for-each select="liste_auteurs/auteur">
12               <i>
13                 <xsl:value-of select=".//@prenom"/><xsl:text> </xsl:text>
14                 <xsl:value-of select=".//@nom"/>
15               </i><br/>
16             </xsl:for-each>
17           </td>
18           <td><xsl:text>prix </xsl:text>
19             <xsl:value-of select="prix"/><xsl:text> </xsl:text>
20             <xsl:value-of select="prix/@monnaie"/>
21           </td>
22         </tr>
23       </xsl:for-each>
24     </table>
25   </body>
```

Rendu sous forme de table

Apprendre XML en 10 mois	<i>Jean Escargot Brad Snail</i>	prix 20
Learn Java in 10 seconds	<i>John Fast</i>	prix 25 dollars

Document Type Definition

Définition du type de document

Document Type Definition (DTD)

Utilisé pour normer des documents XML.

- En décrit la grammaire - la liste des éléments et balises autorisés, leurs attributs, contenu et agencement - ainsi que le vocabulaire supplémentaire sous la forme d'une liste d'entités de caractères.

XML Schema est une alternative plus puissante.

Utilisation d'une DTD (fichier d'extension .dtd)

- En la définissant directement dans le document XML.
- En la définissant dans un fichier externe déclaré dans le document XML (recommandé).

DTD pour documents (X)HTML

XHTML 1.0 Strict

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

XHTML 1.0 Transitional

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 Frameset

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"  
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

HTML 5

```
1 <!DOCTYPE html>
```

Exemple de fichier XML déclarant une DTD externe

bibliotheque.xml

```
1 <?xml version="1.0" encoding="utf-8" standalone="no"?>
2 <!-- Déclaration externe de DTD -->
3 <!DOCTYPE bibliotheque SYSTEM "bibliotheque.dtd">
4 <!-- Référence à un fichier XSLT pour mise en forme -->
5 <?xmlstylesheet type="text/xsl" href="bibliotheque-liste.xsl" ?>
6 <bibliotheque>
7 <livre>
8   <titre>Apprendre XML en 10 mois</titre>
9   <liste_auteurs>
10    <auteur nom="Escargot" prenom="Jean" />
11    <auteur nom="Snail" prenom="John" />
12   </liste_auteurs>
13   <prix>20</prix>
14 </livre>
15 <livre>
16   <titre>Learn Java in 10 seconds</titre>
17   <liste_auteurs>
18    <auteur nom="Fast" prenom="Bob" />
19   </liste_auteurs>
20   <prix monnaie="dollars">25</prix>
21 </livre>
22 </bibliotheque>
```

Déclaration du type du document

bibliotheque.xml

```
<!DOCTYPE bibliotheque SYSTEM "bibliotheque.dtd">
```

Document Type Declaration (DOCTYPE)

La déclaration du type du document indique

- La DTD à laquelle doit se conformer le document XML.
- La balise de l'élément racine : bibliothèque.

Ne pas confondre *Document Type Definition* (DTD) et *Document Type Declaration* (DOCTYPE).

Exemple de DTD

bibliotheque.dtd

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <!ELEMENT bibliotheque (livre+) >
3 <!ELEMENT livre (titre, liste_auteurs, prix?) >
4 <!ELEMENT titre (#PCDATA) >
5 <!ELEMENT liste_auteurs (auteur+) >
6 <!ELEMENT auteur EMPTY>
7 <!ATTLIST auteur nom CDATA #REQUIRED >
8 <!ATTLIST auteur prenom CDATA #IMPLIED >
9 <!ELEMENT prix (#PCDATA) >
10 <!ATTLIST prix monnaie CDATA #IMPLIED>
```

Structure de la DTD

bibliotheque.dtd

```
<!ELEMENT bibliotheque (livre+)>
```

Règle imposée

L'élément (de balise) **bibliotheque** contiendra un ensemble d'éléments **livre**.

On peut quantifier le nombre d'éléments **livre** :

- + : au moins 1.
- * : 0 ou plus.
- ? : 0 ou 1.
- Sans indication, on attend une seule occurrence.

Structure de la DTD

bibliotheque.dtd

```
<!ELEMENT livre (titre, liste_auteurs, prix?)>
```

Règle imposée

Un élément **livre** contiendra :

- Un élément **titre**.
- Suivie d'un élément **liste_auteurs**.
- Suivie éventuellement d'un élément **prix**.

a | b signifie *soit* a, *soit* b.

Structure de la DTD

bibliotheque.dtd

```
<!ELEMENT titre (#PCDATA)>
```

Règle imposée

Un titre contiendra une chaîne de caractères.

- PCDATA : la chaîne sera analysée comme du XML (balisage reconnu comme tel et entités XML automatiquement développées). Utiliser les entités XML correspondant aux symboles <, > et &. Réservé pour le contenu d'élément.
- CDATA : la chaîne sera donnée entre guillemets mais non analysée. Réservé pour les attributs d'éléments.
- ANY : toute combinaison de données analysables.

Structure de la DTD

bibliotheque.dtd

```
<!ELEMENT auteur EMPTY>
<!ATTLIST auteur nom CDATA #REQUIRED>
<!ATTLIST auteur prenom CDATA #REQUIRED>
```

Règle imposée

Un **auteur** est un élément sans contenu (`EMPTY`) et qui possède deux attributs `nom` et `prenom`.

On peut contraindre chaque attribut :

- `#REQUIRED` : l'attribut doit être défini.
- `#IMPLIED` : l'attribut est optionnel.
- `#DEFAULT` : l'attribut possède une valeur par défaut définie par une valeur entre guillemets.

Entités

Alias pour du texte

Equivalent du `#define` en C.

Définition dans la DTD

```
<!ENTITY progc "programmation en langage C">
```

Usage dans le XML

```
<keyword> &proc </keyword>
```

Manipuler des documents XML

Manipuler XML : DOM et SAX

DOM et SAX

Il existe deux interfaces principales de manipulation pour la lecture des documents XML :

- DOM (Document Object Model) basée sur une représentation hiérarchique.
- SAX (Simple API for XML) basée sur des déclencheurs d'actions.

Le DOM

Interface de programmation (API)

Standardisée pour manipuler par scripts une page HTML chargée dans un navigateur et plus généralement tout document XML.

Appliquée aux pages Web (XHTML*, HTML5), le DOM

- Offre une représentation structurée et orientée objet des éléments et du contenu.
- Permet la modification des propriétés de ces objets par l'intermédiaire de méthodes.
- Permet l'ajout et la suppression d'objets.
- Permet la gestion des événements du navigateur.

Caractéristiques des API DOM et SAX

DOM

- Pas vraiment orientée objet.
- Construit une représentation en mémoire du document sous forme d'arbre.
- Adaptée aux petits fichiers et traitement du document entier.

SAX

- Basée sur des déclencheurs (triggers) qui se déclenchent lors de la lecture de balises.
- Adaptée au traitement local de fichiers volumineux.
- Adaptée à des langages compilés comme C++ ou Java.

Les normes du DOM

Recommandations du W3C

- DOM Level 1 (Core + HTML) depuis 1998.
- DOM Level 2 (Core + Event + Style + Views + Traversal + Range) depuis 2000.
- DOM Level 3 (Validation) depuis 2004.

Mise en oeuvre

- Standardisée autour de JavaScript dans tous les navigateurs.
- API dépendante du langage de scripts côté serveur pour la manipulation de documents XML (eg. les classes `DOMDocument`, `DOMElement` ...en PHP).

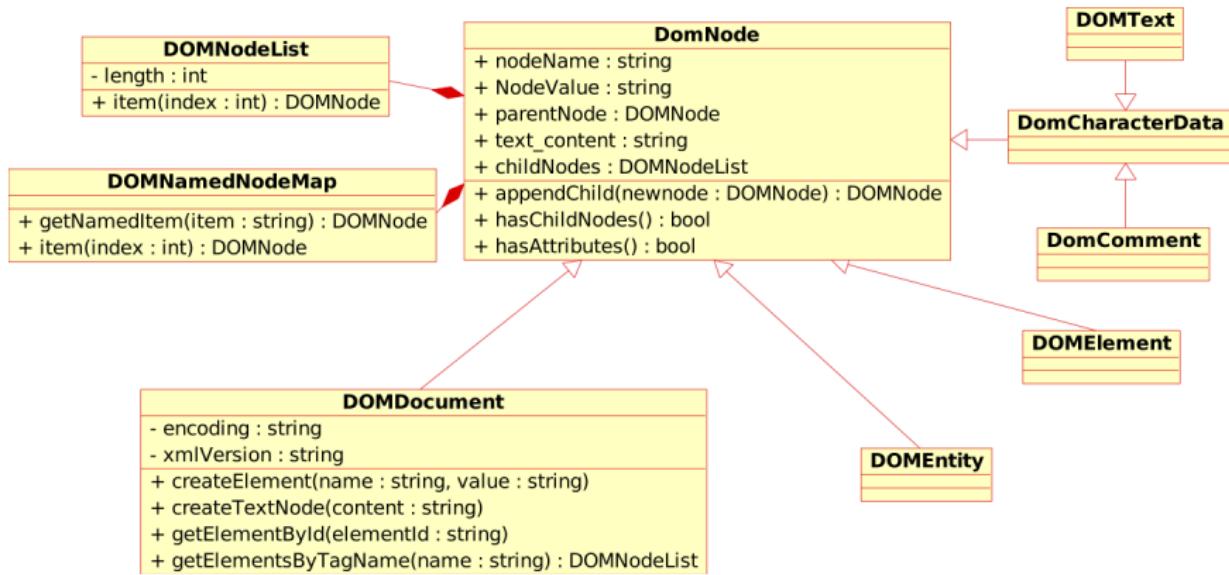
L'arbre d'éléments DOM

Arbre des éléments

- De part sa structure, un document XML est représenté sous forme arborescente.
- Cet arbre possède une racine, des noeuds et des feuilles de différents types.

Visualisation de la structure d'un document HTML sous Firefox avec DOM Inspecteur.

Diagramme UML de l'API DOM



Noeud DOM

DOMNode

Les attributs sont :

- `nodeName` (`string`) : nom du noeud.
- `nodeType` (`int`) : type du noeud.
- `nodeValue` (`string`) : valeur.
- `childNodes` (`DOMNodeList`) : liste des enfants.
- `parentNode` (`DOMNode`) : noeud parent.
- `firstChild`, `lastChild` (`DOMNode`).
- `previousSibling`, `nextSibling` (`DOMNode`).