



PHP : la programmation objet (PHP-OO) Partie 2

Benoît Roche
@rocheb83



OBJECTIFS DU COURS

Le cours de PHP OO est un prérequis à l'étude du framework symfony qui est entièrement bâti sur la notion de programmation orientée objet (POO)

Il se divise en deux parties :

- Les fondamentaux de PHPOO : qui a déjà été vu
- Les concepts avancées de PHPOO : c'est l'objet de ce présent cours



Prérequis à ce présent cours :

- ✓ Connaître les bases de la programmation objet
- ✓ Avoir suivi le cours de base de PHPOO



LES ESPACES DE NOMS (NAMESPACES)

Benoît Roche / @rocheb83

3



PHP OO AVANCÉ : LES NAMESPACES

Ils représentent des conteneurs capables d'encapsuler :

- ✓ Des classes,
- ✓ Des interfaces,
- ✓ Des fonctions,
- ✓ Des constantes



New...
php 5.3

2 avantages à bien comprendre :

Éviter les **conflits de noms** entre les éléments que l'on crée et les éléments que l'on utilise à travers les bibliothèques tierces

Avoir la possibilité de créer des **alias** pour utiliser facilement des éléments se trouvant très loin dans la hiérarchie des namespaces

Benoît Roche / @rocheb83

4



PHP OO AVANCÉ : LES NAMESPACES

Les espaces de noms sont déclarés avec le mot clé **namespace**

Ce mot clé doit être placé en début de fichier, avant tout autre code (php ou html), sauf declare

Le même namespace peut être défini dans **plusieurs fichiers** pour organiser le code proprement.



On peut définir plusieurs espaces de noms dans le même fichier, mais c'est déconseillé pour une meilleure organisation du code.

```
espace01.php
<?php
namespace MonProjet;

const MACONSTANTE='Cours POO';
function mafonction () { .....}
class MaClasse {.....}
?>
```

Exemple dans symfony

```
namespace Doctrine\ORM;

class EntityRepository implements ObjectRepository, Selectable
{
}
```

Benoît Roche / @rocheb83

5



PHP OO AVANCÉ : LES NAMESPACES

On peut déclarer des sous espaces de noms

Cela permet de hiérarchiser les projets ou les bibliothèques ou frameworks.

```
space02.php
<?php
namespace MonProjet\Niveau1\Niveau11;

const MACONSTANTE='Niveau11';
function mafonction () { .....}
class MaClasse {.....}
?>
```

La constante **__NAMESPACE__** permet d'afficher l'espace de noms courant

```
echo "espace de noms : " . __NAMESPACE__ ;
?>
```

localhost/Cours_php/POO/namespace02.php

espace de noms : MonProjet\Niveau1\Niveau11

Benoît Roche / @rocheb83

6



PHP OO AVANCÉ : LES NAMESPACES

Ainsi pour accéder à une classe, on doit faire précéder le nom de la classe de son namespace

```
try {
    $unePersonne = new MesClasses\Employe(1, "Martin", "Robert");
}
```

Ici, on instancie un objet de la classe Employe décrite dans l'espace de noms MesClasses :

```
namespace MesClasses;
class Employe {
}
```



PHP OO AVANCÉ : LES NAMESPACES

Ainsi, On peut accéder à une ressource dans un espace de noms :

- En relatif (et utilisation du mot clé : **namespace**)

mafonction()	Appel de la fonction mafonction() dans l'espace de noms courant
namespace\mafonction()	Appel de la fonction mafonction() dans l'espace de noms courant (mot clé namespace)
Niveau1\mafonction()	Appel de la fonction mafonction() dans l'espace de noms Niveau1 situé dans l'espace de noms courant



PHP OO AVANCÉ : LES NAMESPACES

➤ En absolu (utilisation du slash : \)

\MonProjet\mafonction()	Appel de la fonction mafonction() dans l'espace de noms MonProjet
\Monprojet\Niveau1\mafonction()	Appel de la fonction mafonction() dans l'espace de noms Niveau1 situé dans l'espace de noms MonProjet



PHP OO AVANCÉ : LES NAMESPACES

La fonction maFonction est déclarée dans l'espace de noms Niveau1 sous l'espace de noms du script principal.
Différentes façons d'accéder à la fonction :

```
1 <?php
2 namespace MonProjet;
3 include 'namespace03.php';
4
5 function mafonction() {
6     echo "<p> Dans espace MonProjet </p>";
7 }
8 echo "<H1><p>Résultat :<p>";
9 mafonction();
10 namespace\mafonction();
11 Niveau1\mafonction();
12 \MonProjet\mafonction();
13 \MonProjet\Niveau1\mafonction();
14 echo "</H1>";
15 ?>
```

```
1 <?php
2 namespace MonProjet\Niveau1;
3
4 function mafonction() {
5     echo "<p> Dans namespace MonProjet\Niveau1 </p>";
6 }
```

- Résultat :
- Dans espace MonProjet
 - Dans espace MonProjet
 - Dans namespace MonProjet\Niveau1
 - Dans espace MonProjet
 - Dans namespace MonProjet\Niveau1



PHP OO AVANCÉ : LES NAMESPACES

De même, on peut instancier une classe déclarée dans un espace de nom :

✓ **Sans qualificatif**

`$obj= new MaClasse ();` // MaClasse est dans l'espace de nom courant

`$obj =new namespace\MaClasse();` //idem

✓ **Avec un qualificatif (sous espace de nom)**

`$obj = new Niveau1\MaClasse() ;` //MaClasse est dans l'espace de nom Niveau1 situé au niveau immédiatement en dessous de l'espace de nom courant.

✓ **Avec un qualificatif absolu**

✓ `$obj= new \Niveau1\MaClasse() ;` // MaClasse est dans l'espace de nom Niveau1 qui est une racine d'espace de nom

Benoît Roche / @rocheb83

11



PHP OO AVANCÉ : LES NAMESPACES

Dans un espace de noms, il est possible d'appeler une ressource qui n'est dans aucun espace de nom :

Il suffit d'indiquer un chemin absolu :

`\mafonction();`

Mafonction dans aucun espace de noms

```
namespace05.php | namespace06.php
<?php
namespace MonProjet;
include 'namespace05.php';

function mafonction() {
    echo "<p> Dans espace MonProjet </p>";
}

echo "<H1><p>Résultat :<p>";
mafonction();
\mafonction();
echo "</H1>";
?>
```

```
namespace05.php | namespace06.php
1 <?php
2
3 function mafonction() {
4     echo "<p> Hors espace de nom </p>";
5 }
6 ?>
```

Résultat :

Dans espace MonProjet

Hors espace de nom

Benoît Roche / @rocheb83

12



PHP OO AVANCÉ : LES NAMESPACES

Utilisation des alias (**use**)

- ✓ Particulièrement intéressant dans le cas de grandes hiérarchies de namespaces.
- ✓ On utilise le mot clé **use**
- ✓ Les noms de chemin sont identiques (chemin absolu, relatif)
- ✓ On peut poser des alias sur :
 - Des noms de classe,
 - Des noms d'interface
 - Des espaces de nom



```
namespace Doctrine\ORM;

use Exception;
use Doctrine\Common\EventManager;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\DBAL\Connection;
```

Nom de la classe

Benoît Roche / @rocheb83

13



PHP OO AVANCÉ : LES NAMESPACES

```
1 <?php
2 namespace UnEspaceDeNom\SousNiveau;
3 class MACLASSE {
4     function __construct(){
5         echo "<p>Dans le constructeur</p>";
6     }
7 }
8 ?>
```

Utilisation avec et sans alias

- ✓ Dans le cas 1 SousNiveau sera l'alias du NS qui servira de point de départ
- ✓ Dans le cas 2 l'alias porte sur le nom de la classe

```
namespace MonProjet;
include 'namespace07.php';
// l'instruction suivante est idem à : use UnEspaceDeNom\SousNiveau as SousNiveau

use UnEspaceDeNom\SousNiveau;
use UnEspaceDeNom\SousNiveau\MACLASSE as AliasClasse;

echo "<H1><p>Résultat :<p>";
$obj1= new AliasClasse();
$obj2= new SousNiveau\MACLASSE();
echo "</H1>";
?>
```

localhost/Cours_php/POO/namespace08.php

Résultat :

Dans le constructeur

Dans le constructeur

Benoît Roche / @rocheb83

14



PHP OO AVANCÉ : LES NAMESPACES

Dès l'instant où on déclare un espace de nom (namespace), PHP résout toujours les noms de classe, de fonctions et de constante par rapport à l'espace de nom courant.

**C'est particulièrement vrai pour la classe Exception
(Qui n'est déclarée dans aucun espace de nom)**

php va chercher la classe Exception dans le namespace courant

```
<?php
namespace ExempleCours\Niveau1\Niveau2;

class Employe {
    private $numemp;
    private $nomemp;
    private $prenomemp;
    protected $salaire;

    public function __construct($num, $nom, $prenom) {
        throw new Exception("Erreur constructeur de Employe");
    }
}
```

```
<?php
namespace ApplicationCours;

include 'Namespace_ExempleV3.php';

use ExempleCours\Niveau1\Niveau2 as MesClasses;

try {
    $unePersonne = new MesClasses\Employe(1, "Martin", "Robert");
    echo "Employe instancié dans le namespace : " . __NAMESPACE__ . "<br />";
} catch (Exception $Ex) {
    echo $Ex->getMessage();
}
```

[Démonstration](#)
[code Classe](#) [code Exemple](#)

Benoît Roche / @rocheb83

#	Time	Memory	Function	Location
1	0.0003	248184	(main)()	.\Namespace_Exemple_v3.php:0
2	0.0006	254456	ExempleCours\Niveau1\Niveau2\Employe->__construct()	.\Namespace_Exemple_v3.php:9

15



PHP OO AVANCÉ : LES NAMESPACES

Il faut donc fournir un chemin absolu à la classe exception....

```
} public function __construct($num, $nom, $prenom) {
    throw new \Exception("Erreur constructeur de Employe");
}

try {
    $unePersonne = new MesClasses\Employe(1, "
    echo "Employe instancié dans le namespace
} catch (\Exception $Ex) {
    echo $Ex->getMessage();
}
```

[Démonstration](#) [code Classe](#) [code Exemple](#)

Benoît Roche / @rocheb83

16



PHP OO AVANCÉ : LES NAMESPACES

.... Ou poser des alias de noms de classe....

```
<?php
namespace ExempleCours\Niveau1\Niveau2;
use \Exception;
class Employe {
    private $numemp;
    private $nomemp;
    private $prenomemp;
    protected $salaire;
    public function __construct($num, $nom, $prenom) {
        throw new Exception("Erreur constructeur de Employe");
    }
}
```

On pose un alias sur le nom de la classe Exception

```
<?php
namespace ApplicationCours;
include 'Namespace_ExempleV5.php';
use ExempleCours\Niveau1\Niveau2 as MesClasses;
use \Exception;
try {
    $unePersonne = new MesClasses\Employe(1, "Martin", "N");
    echo "Employe instancié dans le namespace : " . __NAMESPACE__ . "\n";
} catch (Exception $Ex) {
    echo $Ex->getMessage();
}
?>
```

Benoît Roche / @rocheb83

[Démonstration](#) [code Classe](#) [code Exemple](#)

17



PHP OO AVANCÉ : LES NAMESPACES

Risque de collision : il y a très peu de chance que cela arrive Imaginez :

- ✓Même hiérarchie d'espace de noms,
- ✓Même nom de classe

Si toutefois cela arrivait, Php provoquerait une erreur au moment du chargement des classes

```
<?php
include('double.php');
echo "fichier double bien chargé";
include('trouble.php');
echo "fichier trouble bien chargé";
$monObjetA = new \NS1\MaClasse();
$monObjetB = new \NS1\MaClasse();
$monObjetA->methode($monObjetB);
echo $monObjetA->getMembre() . "<br />";
echo $monObjetB->getMembre() . "<br />";
?>
```



Erreur au moment du chargement du fichier Trouble.php

double.php

```
<?php
namespace NS1;
class MaClasse
{
    private $membre;
```

trouble.php

```
<?php
namespace NS1;
class MaClasse
{
    private $nom;
```

fichier double bien chargé

Fatal error: Cannot redeclare class NS1\MaClasse in E:\Wampsites\Cours_php\POO\trouble.php on line 3

Call Stack

#	Time	Memory	Function	Location
1	0.0012	140880	{main}()	.namespaceindex.php:0

Benoît Roche / @rocheb83

18



LES MÉTHODES MAGIQUES

Benoît Roche / @rocheb83

19



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

Une méthode magique est une méthode qui, lorsqu'elle est définie dans une classe, sera automatiquement appelée lors d'un événement.

- ✓ Une méthode magique intercepte donc un événement, le traite et retourne une valeur.
- ✓ Leur nom commence toujours par 2 underscores, leur nom est réservé
- ✓ Dans la plupart des cas, ces méthodes doivent être définies comme publiques et non statiques



On en connaît déjà 3 :

```
__construct()
__destruct()
__autoload()
```

```
public function __construct($num, $nom, $prenom) {
    try {
        $_numemp = $num;
```

Benoît Roche / @rocheb83

20



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

Il existe des méthodes magiques pour les propriétés et pour les méthodes.

Pour les attributs, il s'agit de gérer la surcharge des attributs. C'est-à-dire de créer dynamiquement des attributs.

Cela est possible lorsque l'on tente d'accéder à un attribut qui n'existe pas ou qui n'est pas accessible (accès à un attribut privé depuis l'extérieur de la classe)



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

Les méthodes magiques pour les attributs

Méthode	Définition
__get(\$attribut)	Accesseur : appelée lorsque l'on essaye de récupérer la valeur d'une propriété qui n'existe pas, où qui n'est pas accessible (déclarée private)
__set(\$attribut, \$valeur)	est appelé lorsque l'on essaye d'attribuer une valeur à une propriété qui n'existe pas

\$attribut : nom de l'attribut que l'on a essayé d'atteindre
\$valeur : valeur de l'attribut qu'on a essayé de modifier



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

```
class Personne {
    private $_num;
    private $_nom;
    private $_prenom;
    protected $_salaire;

    public function __set($attribut, $valeur) {
        $this->$attribut = $valeur;
    }

    public function __get($attribut) {
        return $this->$attribut;
    }
}

$unePersonne = new Personne(1, "Dupont", "Jean");

echo $unePersonne->_nom . " son prénom est " . $unePersonne->_prenom . "

$unePersonne->_nom = "Durand";
$unePersonne->_nomJeuneFille = "Ginestet";
echo "</n>";

if (isset($unePersonne->_salaire))
    echo "salaire isset : oui";
else
    echo "salaire isset : non";
echo "<br>";
if (isset($unePersonne->_nomJeuneFille))
    echo "nomJeuneFille isset : oui";
else
    echo "nomJeuneFille isset : non";
?>
```

Démonstration code



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

Il existe aussi les méthodes __isset et __unset pour la *gestion* des attributs :

Méthode	Définition
<u>__isset(\$attribut)</u>	appelée lorsque l'on essaye de savoir si une propriété qui n'existe pas, ou n'est pas accessible est définie. (voir utilisation avec tableau \$data)
<u>__unset(\$attribut)</u>	Appelée lorsque l'on essaie de supprimer une propriété qui n'existe pas, ou n'est pas accessible est définie. (voir utilisation avec tableau \$data)

\$attribut : nom de l'attribut que l'on a envoyé à la méthode isset ou unset



Ces 4 fonctions magiques peuvent être utilisées dans la gestion dynamique des attributs : voir diapo suivante



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

utilisation d'un tableau pour stocker les propriétés

Créer la classe Ctest qui contiendra :

- ✓ un tableau \$propriétés (privé)
- ✓ Une variable \$variableCachee (privée)
- ✓ Une méthode publique __set(\$unNomPropriete, \$saValeur), qui affecte la valeur \$saValeur à l'élément d'indice \$unNomPropriete.
- ✓ Une méthode publique __get(\$unNomPropriete) qui renvoie, si elle existe la valeur de la propriété \$unNomPropriete. Si elle n'existe pas, déclencher une exception.
- ✓ Une méthode publique __isset(\$unePropriete) qui renvoie vrai si la propriété \$unePropriete existe dans le tableau des proprietes
- ✓ Une méthode publique AfficherProprietes qui parcourt le tableau \$propriétés et affiche les noms des propriétés et leur valeur.
- ✓ Un constructeur admettant un paramètre valorisant la propriété privée \$variableCachee

Tester avec les propriétés de la classe Personnage

attribuer une valeur à la propriété privée (\$this-> variableCachee="nouvelleValeur")

Afficher le tableau

Essayer d'afficher une propriété qui n'existe pas

[Solution](#) [codeV1](#) [codeV2](#)

Benoît Roche / @rocheb83

25



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

Les méthodes magiques pour les méthodes.

Pour les méthodes, il s'agit de gérer la surcharge des méthodes, c'est-à-dire d'exécuter des méthodes inaccessibles ou qui n'existent pas... tout un programme !.

Méthode	Définition
<u>__call(\$methode, \$tableauArguments)</u>	Appelée lorsque l'on essaye d'exécuter une méthode qui n'existe pas ou qui n'est pas accessible.
<u>__callStatic(\$methode, \$tableauArguments)</u>	Idem mais pour les méthodes statiques

\$methode : nom de la méthode que l'on a essayé d'appeler

\$tableauArguments : liste des oaramètres sous forme de tableau

[Démonstration](#) [code](#)

Benoît Roche / @rocheb83

26



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

```
public function __call($nomMethode, $arguments) {
    echo "vous essayez d'appeler la méthode " . $nomMethode . "<br />";
    echo 'avec les paramètres : <br />';
    print_r($arguments);
    echo "<br>";
    if (method_exists($this, $nomMethode)) {
        echo $this->$nomMethode($arguments);
    } else
        throw new Exception("La methode que vous appelez n'existe pas");
}

try {
    $unePersonne = new Personne('Dupont', 'Jean');
    echo $unePersonne->afficheNomPrenom();
    $unePersonne->methodePublique();
    $unePersonne->test("EEEE", "234", "Test");
} catch (Exception $uneException) {
    echo "ERREUR .... " . $uneException->getMessage();
}
```

vous essayez d'appeler la methode test
avec les paramètres :
Array ([0] => EEEE [1] => 234 [2] => Test)
ERREUR La methode que vous appelez n'existe pas

[Démonstration](#) [code](#)

Benoît Roche / @rocheb83

27



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES



Exercice:

En utilisant la classe personnage créer :

- ✓ un constructeur qui admet en paramètres le nom et le prénom de la personne.
- ✓ la méthode privée afficheNomPrenom qui n'accepte pas de paramètre et affiche le nom et le prénom de la personne, et qui déclenche une exception si le nom et le prénom ne sont pas définis
- ✓ La méthode __call qui affiche le nom et les paramètres de la méthode que vous appelez. Elle teste si la méthode que vous tentez d'appeler existe (method_exists) et si elle existe, vous l'exécutez avec les paramètres passés lors de l'appel. Si elle n'existe pas, vous déclenchez une exception.
- ✓ La méthode publique methodePublique qui affiche "vous êtes dans la méthode publique"

Tester votre classe avec le code suivant :

```
$unePersonne = new Personnage('Dupont', 'Jean');
$unePersonne->afficheNomPrenom();
$unePersonne->methodePublique();
$unePersonne->test("EEEE", "234", "Test");
```

[Démonstration](#) [code](#)

Benoît Roche / @rocheb83

28



PHP OO AVANCÉ : LES MÉTHODES MAGIQUES

Méthode	Définition
<u>__toString</u>	Appelée lorsque l'on essaye de convertir un objet en chaine de caractères

On peut ainsi renvoyer un chaine prédéfinie si on fait par exemple echo \$obj;

```
class Personne {
    private $_num;
    private $_nom;
    private $_prenom;

    // Constructeur
    public function __construct($num, $nom, $prenom) {...9 lines }

    public function __toString() {
        return "Je m'appelle " . $this->_prenom . " " . $this->_nom . "<br>J'ai le numéro " . $this->_num;
    }
}

<?php
$unePersonne = new Personne(7, 'Bond', 'James');
echo $unePersonne;
```

Je m'appelle James Bond
J'ai le numéro 7



PHP OO AVANCÉ : LES CONSTANTES MAGIQUES

Il existe d'autres méthodes magiques qui seront vues plus tard dans le cours.

Mais il existe aussi des **CONSTANTES** magiques :

Hors de classe	Dans la classe
__LINE__	Ligne courante du script.
__FILE__	Nom et chemin absolu du fichier ou du fichier inclus
__DIR__	Dossier du fichier ou du fichier inclus
__FUNCTION__	Nom de la fonction, en tenant compte de la casse
__CLASS__	Nom de la classe avec espace de nom
__TRAIT__	Nom du trait avec son espace courant
__METHOD__	Nom de la méthode courante
__NAMESPACE__	Nom de l'espace courant



LES INTERFACES

Benoît Roche / @rocheb83

31



PHP OO AVANCÉ : LES INTERFACES

Une **interface** représente un **modèle de classe**.
Elle définit un comportement

- **Elle peut contenir :**
 - ✓ Des méthodes publiques
 - ✓ Des constantes
- **Elle ne peut contenir :**
 - ✓ Des méthodes privées ou protected
 - ✓ Des membres variables

Benoît Roche / @rocheb83

32



PHP OO AVANCÉ : LES INTERFACES

Une **interface** représente un **modèle de classe**.

- **Une interface peut hériter d'une autre interface ou de plusieurs interfaces**
- **Une classe héritant d'une interface :**
 - ✓ doit OBLIGATOIREMENT implémenter les méthodes publiques
 - ✓ Peut proposer d'autres méthodes et constantes

Une classe peut **implémenter** plusieurs interfaces



C'est un moyen de contourner l'héritage multiple.

Benoît Roche / @rocheb83

33



PHP OO AVANCÉ : LES INTERFACES

On utilisera les 2 mots clé :

interface pour déclarer une interface

```
interface ObjectManager
{
    /** Finds a object by its identifier ...10 lines */
    public function find($className, $id);
}
```

implements dans la classe pour signaler qu'elle implémente l'interface citée

```
/* final */class EntityManager implements EntityManagerInterface
{...913 lines }
```

[Code](#) [Démonstration](#)

Benoît Roche / @rocheb83

34



PHP OO AVANCÉ : LES INTERFACES

Une interface peut hériter d'une autre interface ou de plusieurs interfaces

```
interface EntityManagerInterface extends ObjectManager
{
    public function getConnection();
    public function getExpressionBuilder();
    public function beginTransaction();
    public function transactional($func);
}
```



Dans ce cas, les classes qui implémentent les interfaces doivent obligatoirement implémenter l'ensemble des méthodes figurant dans la hiérarchie d'interfaces.



PHP OO AVANCÉ : LES INTERFACES

```
namespace Interface2;

interface Humain {
    public function manger();

    const NBJAMBES = 2;
}

interface Personne extends Humain {
    public function conduire();
}
```



```
class Sportif implements Personne{

    private $_nom;

    public function __construct($unNom) {...3 lines }

    public function getNom() {...3 lines }

    public function manger() {...3 lines }

    public function conduire() {...3 lines }

    public function jouerMusique($instrument) {...3 lines }

}
```

On voit que l'interface a été définie dans un espace de noms

La classe personne devra implémenter les méthodes :
✓ manger()
✓ conduire ()

[Code Démonstration](#)



PHP OO AVANCÉ : LES INTERFACES

Une classe pourra **implémenter** plusieurs interfaces ...
C'est un moyen de simuler l'héritage multiple !!!

```
interface Humain {
    public function manger();
    public function telephoner();
    const NBJAMBES = 2;
}

interface Singe {
    public function grimperAuxArbres();
    public function manger();
}

class Bipede implements Humain, Singe {
    private $_nom;
    public function __construct($unNom) { ...3 lines }
    public function telephoner() { ...3 lines }
    public function manger() { ...3 lines }
    public function grimperAuxArbres() { ...3 lines }
}
```

...Impossible !!!
.. Avant Php 5.4

Double "héritage":

```
$unBipede = new Bipede("Bond");
echo $unBipede->manger();
echo $unBipede->telephoner();
echo $unBipede->grimperAuxArbres();
echo "Un bipède a " . Bipede::NBJAMBES . " jambes";
--
```

Un bipède a besoin de manger !
un bipède doit savoir téléphoner
Un bipède sait aussi grimper aux arbres !
Un bipède a 2 jambes

[Code Démonstration](#)



LES TRAITS



PHP OO AVANCÉ : LES TRAITS



New
php 5.4

Un trait est un moyen de réutiliser du code (méthodes et attributs) entre différentes classes.

Ainsi, un trait

- ✓ est comparable à une classe
- ✓ permet d'ajouter du code à une classe, héritée ou non
- ✓ enrichit l'héritage simple sans pour autant faire de l'héritage multiple
- ✓ contient des méthodes et des données membres
- ✓ Le niveau de visibilité des membres est **public ou privé**
- ✓ N'est pas instanciable

Benoît Roche / @rocheb83

39



PHP OO AVANCÉ : LES TRAITS

Un trait est un moyen de réutiliser du code (méthodes et attributs) entre différentes classes.

Syntaxe : `trait NomDuTrait {`
`...`
`}`

On déclare un trait dans une classe avec le mot clé **use**

[Code Démonstration](#)

```
trait HelloWorld {
    public function disHello() {
        echo 'Hello World!';
    }
}
// définition de la classe
class UneClasse {
    use HelloWorld;
```

Benoît Roche / @rocheb83

40



PHP OO AVANCÉ : LES TRAITS

```

<?php
// définition du trait
trait UnTrait {
    private $variableDuTrait="variable du trait";
    public function fonctionDuTrait(){
        echo "<p>Fonction du trait</p>";
    }
}

include 'traits01.php';

class UneClasse {
    use UnTrait;
    public function __construct(){
        echo "<p> Dans le constructeur </p>";
    }
    public function afficheVariable(){
        echo "<p>" . $this->variableDuTrait . " </p>";
    }
}

// début du programme
$obj = new UneClasse();
$obj->afficheVariable();
$obj->fonctionDuTrait();
?>
```

Dans le constructeur

variable du trait

Fonction du trait

[Code Démonstration](#)



PHP OO AVANCÉ : LES TRAITS

Priorité : cas où 2 méthodes entrent en concurrence

Règle 1 : la méthode héritée d'une classe mère est écrasée par la méthode issue d'un trait


Règle 2 La méthode d'une classe non héritée écrase la méthode issue d'un trait



Une classe peut utiliser plusieurs traits.
Déclaration


use Trait1, Trait2

[Code Démonstration](#)



PHP OO AVANCÉ : LES TRAITS

```
class Mere {
    public function disHello() {
        echo 'Hello ';
    }
}
// définition du trait
trait UnTrait {
    public function disHello() {
        parent::disHello();
        echo 'World!';
    }
}
// définition de la classe fille
class Fille extends Mere {
    use UnTrait;
}
// début du script
$objet = new Fille();
$objet->disHello();
?>
```



localhost/Cours_php/POO/traits03.php

Les plus visités PHP : révisions Débuter

Hello World!

la méthode du trait a gagné

```
trait HelloWorld {
    public function disHello() {
        echo 'Hello World!';
    }
}
// définition de la classe
class UneClasse {
    use HelloWorld;
    public function disHello() {
        echo 'Hello Universe!';
    }
}
// début du script
$obj = new UneClasse();
$obj->disHello();
?>
```

localhost/Cours_php/POO/traits04.php


Les plus visités PHP : révisions Débuter

Hello Universe!

la méthode de la classe a gagné

Benoît Roche / @rocheb83

43



AUTOCHARGEMENT DES CLASSES SPL_AUTOLOAD

Benoît Roche / @rocheb83

44



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

La méthode `__autoload` vue dans la première partie du cours a de gros inconvénients :

Elle n'admet quasiment qu'une seule règle en matière de

✓Nom de fichier

✓Emplacement de fichier



La solution :

Les fonctions `spl_autoload`



Benoît Roche / @rocheb83

45



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Si aucune fonction `__autoload` n'est définie, alors php utilisera la fonction native `spl_autoload()`

... à condition de lui signifier par la fonction :

`Spl_autoload_register()`



Deux fonctions à connaître :

✓**`Spl_autoload_register`** : activer la recherche automatique de classes

✓**`Spl_autoload_extensions`** : permet de rechercher des fichiers de classes avec plusieurs extensions possibles

Benoît Roche / @rocheb83

46



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Spl_autoload_register : En fait cette fonction va rechercher partout où la configuration dit de chercher, et elle prendra le premier fichier trouvé.

Spl_autoload ne fonctionne que :

- ✓ *si `__autoload` n'a pas été définie*
- ✓ ***Spl_autoload_register** a été activé.*
Il l'est lors de son premier appel dans le script

Benoît Roche / @rocheb83

47



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Spl_autoload recherchera :

- ✓ Dans les fonctions qu'on lui indiquera,
- ✓ Dans le dossier correspondant à la valeur de la directive **include_path** du fichier php.ini
- ✓ Dans les path redéfinis avec la fonction
set_include_path()
- ✓ Avec des extensions prédéfinies
spl_autoload_extensions()

Benoît Roche / @rocheb83

48



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Spl_autoload_register : active la recherche automatique de classes.

Exemple : on va indiquer à spl_autoload la fonction à utiliser pour l'autochargement (monChargeur)

```
include '__spl_chargeurSimple.php';

spl_autoload_register('monChargeur');
try {
    $unInformaticien = new personnelInformaticien('Dupont', 'Jean', 1000);
    echo "classe chargée Nom = " . $unInformaticien->afficheNomPrenom();
} catch (Exception $e) {
    echo $e->getMessage();
}

function monChargeur($UneClasse) {
    $fichier = 'classes\\' . $UneClasse . '.php';
    echo "classe a charger : " . $UneClasse . " <br />et nom fichier : " . $fichier . "<br />";
    if (file_exists($fichier)) {
        include ($fichier);
    } else {
        throw new Exception("Impossible de charger la classe " . $UneClasse);
    }
}
```

Benoît Roche / @rocheb83

[Démonstration code](#)

49



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

On peut indiquer à **Spl_autoload** de rechercher dans d'autres dossiers que ceux inclus dans la directive **include_path** du fichier php.ini.

✓ On redéfinit alors les paths à explorer avec la fonction **set_include_path()**

On peut dynamiquement **redéfinir** les dossiers à examiner avec la fonction **set_include_path()**

string set_include_path (chemin1,chemin2...)

On peut récupérer la valeur de la directive **include_path** avec la fonction **get_include_path()**

string get_include_path ();

Benoît Roche / @rocheb83

50



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Souvent, on l'inclut dans la nouvelle directive en redéfinissant la directive `include_path`.
On force l'autoload à parcourir les chemins indiqués.

```
set_include_path(
    PROJET . 'classes' . PATH_SEPARATOR .
    PROJET . 'outils' . PATH_SEPARATOR .
    get_include_path()
);
echo get_include_path().'\n';
```

Benoît Roche / @rocheb83

51



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

On peut indiquer à **Spl_autoload** de rechercher plusieurs extensions grâce à la fonction **spl_autoload_extensions**:

Cette fonction définit ou retourne les extensions à rechercher pour `spl_autoload`

`string spl_autoload_extensions(extension1,extension2,...)`



Pas d'espaces de part et d'autre de la virgule

```
define('PROJET', $_SERVER['DOCUMENT_ROOT'] .
spl_autoload_extensions(".class.php,.php");
```

[Démonstration](#) [code](#) [code2](#)

Benoît Roche / @rocheb83

52



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

```
<?php
spl_autoload_extensions(".class.php,.php,.inc.php");
set_include_path(
    PROJET . 'classes' . PATH_SEPARATOR .
    PROJET . 'utils' . PATH_SEPARATOR .
    get_include_path()
);
echo get_include_path()."<br />";
echo spl_autoload_extensions()."<br />";
~

<?php
// définition de la constante fournissant le répertoire racine de l'application
define('PROJET', __DIR__ . DIRECTORY_SEPARATOR);
include 'spl_include_path.php';
//on active la recherche de classes
spl_autoload_register();
try {
    $suneVoiture = new Voiture('Ferrari', '458 Italia', 605);
    echo "classe chargée le modèle de la voiture = " . $suneVoiture->get_model() . "<br />";
    $unInformaticien = new personnelInformaticien('Dupont', 'Jean', 1000);
    echo "classe chargée Nom = " . $unInformaticien->afficheNomPrenom() . "<br />";
} catch (Exception $e) {
    echo $e->getMessage();
}
```

Fichier `spl_include_path.php`

Benoît Roche / @rocheb83

[Démonstration](#) [code](#) [code2](#)

53



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Risque de collision de fonctions `__autoload`

... Dans ce cas, il y a erreur... PHP ne gère pas la surcharge de fonctions `__autoload`

Utilisation de la fonction `spl_autoload_register()` : elle permet de gérer une file de fonctions exécutées les unes après les autres jusqu'à ce que la dépendance soit résolue.

Exemple :

```
spl_autoload_register('fantastic_framework_autoload');
spl_autoload_register('helperslib_autoload');
spl_autoload_register('__autoload');
```

Benoît Roche / @rocheb83

54



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Dans chaque fonction, le test de l'existence du fichier avant l'inclusion est obligatoire, sinon PHP affichera une erreur (causant l'arrêt du script avec require).



Benoît Roche / @rocheb83

55



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

La normes PSR-0

De plus en plus de développements intègrent des frameworks, chacun implémentant son propre chargement de classes. (autoloader)

Afin de standardiser ces auto chargements et afin d'inculquer des règles de best practices, un mouvement "officiel" de convention est né en 2009, il s'agit de "**PSR-0**", "**PSR-1**", ...

Site :
[PHP Framework Interop Group\(http://www.php-fig.org/\)](http://www.php-fig.org/)

Benoît Roche / @rocheb83

56



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

La normes PSR-0



Dépréciée, PSR-4
recommandée

Quelques règles simples de **PSR-0** :

- ✓ L'espace de noms doit commencer par le nom du vendeur (ou créateur): \Nom du vendeur\Espace de noms;
- ✓ Il peut naturellement y avoir des sous-espaces de noms;
- ✓ **Chaque niveau de l'espace de noms correspond à un répertoire**
- ✓ Dans le nom de la classe (mais PAS dans le nom de l'espace de noms), chaque "_" correspond à un sous-répertoire;
- ✓ Chaque fichier se termine par ".php".

Benoît Roche / @rocheb83

57



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Exemples :

Namespace : \Doctrine\Common\IsolatedClassLoader

classe : /chemin/vers/projet/lib/vendor/Doctrine/Common/IsolatedClassLoader.php

Sous-tiret dans les Espaces de Noms et Noms de Classes

Namespace : \espace de noms\package\Class_Name

Classe : /chemin/vers/projet/lib/vendor/espace de noms/package/Class/Name.php

Benoît Roche / @rocheb83

58



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

En suivant les exemples précédents, on permet de mettre en place des fonctions standards d’auto chargement de classes :

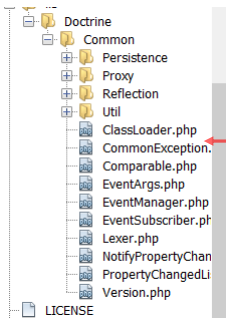
```
<?php

function autoload($className)
{
    $className = trim($className, '\\');
    $fileName = '';
    $namespace = '';
    if ($lastNSPos = strrpos($className, '\\')) {
        $namespace = substr($className, 0, $lastNSPos);
        $className = substr($className, $lastNSPos + 1);
        $fileName = str_replace('\\', DIRECTORY_SEPARATOR, $namespace) . DIRECTORY_SEPARATOR;
    }
    $fileName .= str_replace('_', DIRECTORY_SEPARATOR, $className) . '.php';

    require $fileName;
}
```

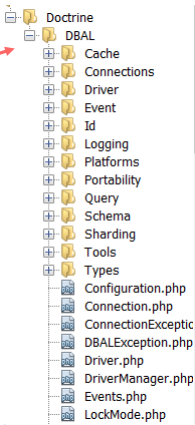


PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES



```
namespace Doctrine\ORM;

use Exception;
use Doctrine\Common\EventManager;
use Doctrine\Common\Persistence\ObjectManager;
use Doctrine\DBAL\Connection;
use Doctrine\DBAL\LockMode;
```



Un exemple : le framework MVC Symfony2

On voit clairement le lien entre les espaces de noms et les emplacements des fichiers



PHP OO AVANCÉ : AUTOCHARGEMENT DES CLASSES

Auto chargement des classes

Il existe d'autres normes concernant l'écriture du code :

PSR-1 : La norme de codage de base

<http://www.php-fig.org/psr/psr-1>

PSR-2 : Guide pour le style d'écriture de code

<http://www.php-fig.org/psr/psr-2>

PSR-3 : Interface Logger

<http://www.php-fig.org/psr/psr-3>

PSR-4 : Auto-chargement des classes (complément PSR-0)

<http://www.php-fig.org/psr/psr-4>



le framework MVC Symfony2 intègre les normes
PSR-0, PSR-1, PSR-2, PSR-4

Benoît Roche / @rocheb83

61



CLONAGE DES OBJETS

Benoît Roche / @rocheb83

62



PHP OO AVANCÉ : CLONAGE DES OBJETS

Il existe deux manières de cloner un objet :

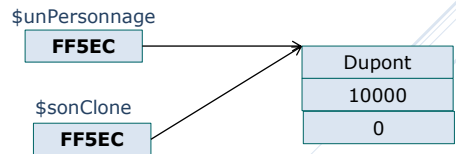
- En affectant un objet à un autre objet
- En utilisant le mot clé clone

En affectant un objet à un autre objet

le compilateur se contente juste de recopier la référence de l'objet source dans l'objet clone....

Ainsi les deux variables objet pointent sur le même objet en mémoire, la modification de l'un entraîne la modification de l'autre

```
$unPersonnage = new employe("Dupont", 10000);  
$unPersonnage->SetTablo(0,0);  
$sonClone = $unPersonnage;
```



Benoît Roche / @rocheb83

63



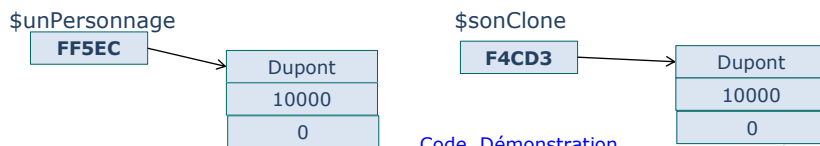
PHP OO AVANCÉ : CLONAGE DES OBJETS

En utilisant le mot clé clone

`$unobjetclone = clone $objet`

le compilateur crée une nouvelle référence qui pointe sur une autre zone de mémoire. Les données membres de l'objet source sont copiées dans l'objet cible

```
$unPersonnage = new Personnage("Dupont", 10000);  
$unPersonnage->SetTablo(0,0);  
$sonClone = clone $unPersonnage;
```



[Code Démonstration](#)

Benoît Roche / @rocheb83

64



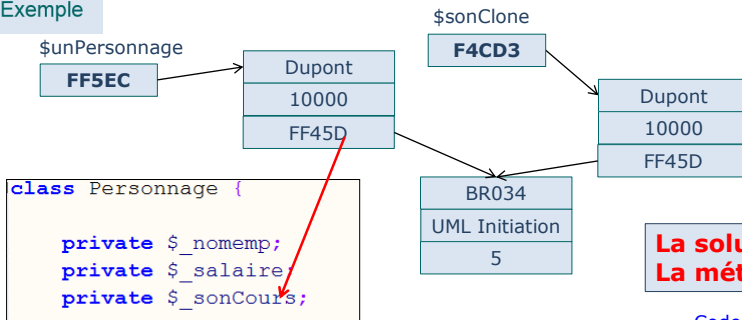
PHP OO AVANCÉ : CLONAGE DES OBJETS



Cette méthode de clonage peut toutefois avoir des effets pervers si la propriété contient une référence....
En effet la référence n'est pas modifiée par le clonage...



Exemple



**La solution
La méthode magique __clone()**

[Code Démonstration](#)

Benoît Roche / @rocheb83

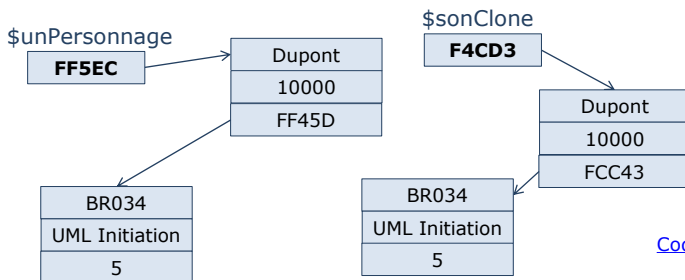
65



PHP OO AVANCÉ : CLONAGE DES OBJETS

La méthode magique **__clone** intercepte le clonage d'un objet et permet de créer une nouvelle référence d'un objet membre

```
public function __clone() {  
    // $this->_sonCours = new cours($this->getSonCours()->getCodeCours(), $this->getSonCours()->getLibelleCours());  
    $this->_sonCours = clone $this->_sonCours;  
}
```



Les 2 méthodes marchent, mais
... attention aux références dans
le deuxième cas !

[Code Démonstration](#)

Benoît Roche / @rocheb83

66



SÉRIALISATION DES OBJETS

Benoît Roche / @rocheb83

67



PHP OO AVANCÉ : SÉRIALISATION DES OBJETS

La sérialisation d'objet permet de linéariser n'importe quelle valeur pouvant être stockée.

Elle concerne

- les variables classiques : entiers, chaînes de caractères
- **Les tableaux**
- **Les objets**

On sérialise dans les cas suivants :

- Stockage des objets en base de données "en l'état"
- Passer des variables d'une page à l'autre (ex les variables de session)

Benoît Roche / @rocheb83

68



PHP OO AVANCÉ : SÉRIALISATION DES OBJETS

Pour sérialiser une variable, on utilise la fonction

serialize(\$v)

Pour récupérer une variable sérialisée, on utilise la fonction

unserialize(\$v)

Qui va remettre la variable en l'état.

Benoît Roche / @rocheb83

69



PHP OO AVANCÉ : SÉRIALISATION DES OBJETS



```
$entier = 7;
$chaine = "Hello world";
$tableau = array('nom'=>'Dupont', 'test'=>'bizarre');
$objEmploye = new employe(3, "Dupont", "Jean");
//
$serializeEntier = serialize($entier);
$serializeChaine = serialize($chaine);
$serializeTableau = serialize($tableau);
$serializeObjet = serialize($objEmploye);
//
echo "Entier : " . $entier . "      sérialisé : " . $serializeEntier . "<br><br>";
echo "Chaine : " . $chaine . "      sérialisé : " . $serializeChaine . "<br><br>";
echo "Tableau : " . $tableau . "      sérialisé : " . $serializeTableau . "<br><br>";
echo "Objet      sérialisé : " . $serializeObjet . "<br><br>";
```

Entier : 7 sérialisé : i:7;

Chaine : Hello world sérialisé : s:11:"Hello world";

Tableau : Array sérialisé : a:2:{s:3:"nom";s:6:"Dupont";s:4:"test";s:7:"bizarre"};

Objet sérialisé : O:17:"serialize\Employee":3:{s:26:"serialize\Employee_numemp";i:3;s:26:"serialize\Employee_nomemp";s:6:"Dupont";s:10:"_prenomemp";s:4:"Jean"};

Benoît Roche / @rocheb83

70



PHP OO AVANCÉ : SÉRIALISATION DES OBJETS

```
$entier = 7;
$chaine = "Hello world";
$tableau = array('nom' => 'Dupont', 'test' => 'bizarre');
$objEmploye = new Employe(3, "Dupont", "Jean");
//
$serializeEntier = serialize($entier);
$serializeChaine = serialize($chaine);
$serializeTableau = serialize($tableau);
$serializeObjet = serialize($objEmploye);
//

$v1 = unserialize($serializeEntier);
$v2 = unserialize($serializeChaine);
$v3 = unserialize($serializeTableau);
$v4 = unserialize($serializeObjet);
```

```
echo "Entier sérialisé : " . $serializeEntier . "désérialisé : " . $v1 . "<br><br>";
echo "Chaine sérialisé : " . $serializeChaine . "désérialisé : " . $v2 . "<br><br>";
echo "Tableau sérialisé : " . $serializeTableau . "désérialisé : " . $v3 . "<br><br>";
echo "Objet sérialisé : ..... désérialisé : " . $v4->getNomemp() . "<br><br>";
```

Entier sérialisé : i:7;désérialisé : 7

Chaine sérialisé : s:11:"Hello world";désérialisé : Hello world

Tableau sérialisé : a:2:{s:3:"nom";s:6:"Dupont";s:4:"test";s:7:"bizarre";}désérialisé : Array

Objet sérialisé : désérialisé : Dupont



[Code Démonstration](#)

Benoît Roche / @rocheb83

71



PHP OO AVANCÉ : SÉRIALISATION DES OBJETS

Il est nécessaire d'inclure la définition de la classe avant la désérialisation d'un objet de cette classe.

Si ce n'est pas fait, Php tentera d'appeler la fonction définie par le paramètre unserialize_callback_func

Défini soit :

- dans le fichier php.ini (ex : unserialize_callback_func : fonctionCallBack)
- Par la fonction ini_set ex : ini_set('unserialize_callback_func', 'fonctionCallBack');

Libre au développeur de définir ou non une telle fonction.



la fonction de rappel spécifiée est appelée lorsque la fonction [unserialize\(\)](#) tente d'utiliser une classe non définie.

Benoît Roche / @rocheb83

72



PHP OO AVANCÉ : SÉRIALISATION DES OBJETS

Les objets mis en variables de session sont désérialisés **AU MOMENT DE L'APPEL** à la fonction `session_start()`;

On ne peut passer en variable de session un objet de la classe PDO

On peut simuler la mise en variable de session d'objets de la classe PDO, en utilisant les fonctions magiques `__sleep` et `__wakeup`



LES MÉTHODES MAGIQUES `__SLEEP` ET `__WAKEUP`



PHP OO AVANCÉ : __SLEEP ET __WAKEUP

Ces 2 méthodes magiques permettent d'effectuer des traitements lors de la sérialisation d'un objet.

- **__sleep** : effectue un traitement au moment de la sérialisation de l'objet
- **__wakeup** : effectue un traitement lors de la désérialisation de l'objet



Ainsi, il est possible par exemple de changer l'état d'un objet, fermer / ouvrir une ressource (bdd, réseau, etc.), ou encore effectuer un log en fichier de l'opération, lors de ces actions de serialisation / deserialisation.

Benoît Roche / @rocheb83

75



PHP OO AVANCÉ : __SLEEP ET __WAKEUP

Un exemple concret : on va stocker un objet de la classe PDO dans une classe. Le constructeur de la classe va effectuer la connexion:

```
class Bdd {  
  
    private $_dbmysql;  
    private $_str;  
    private $_user;  
    private $_pass;  
    public function __construct($str, $login, $pwd) {...6 lines }  
    public function connexion() {  
        try {  
            $options[PDO::ATTR_ERRMODE] = PDO::ERRMODE_EXCEPTION;  
            $this->_dbmysql = new PDO($this->_str, $this->_user, $this->  
        } catch (Exception $e) {  
            throw new Exception("Erreur à la connexion \n" . $e->getMes.  
        }  
    }  
    public function __sleep() {  
        return array('_str', '_user', '_pass');  
    }  
    public function __wakeup() {  
        $this->connexion();  
    }  
}
```

On va stocker dans un tableau les variables connect_str, connect_user et connect_pass au moment de la sérialisation.

On va refaire une connexion à la BD au moment de la désérialisation, avec les variables sauvegardées lors de la sérialisation

[Code appelant](#) [codeappele](#) [classe](#)
[Démonstration](#)

Benoît Roche / @rocheb83

76



Fin du cours, merci

Questions/Réponses

Benoît Roche
@rocheb83