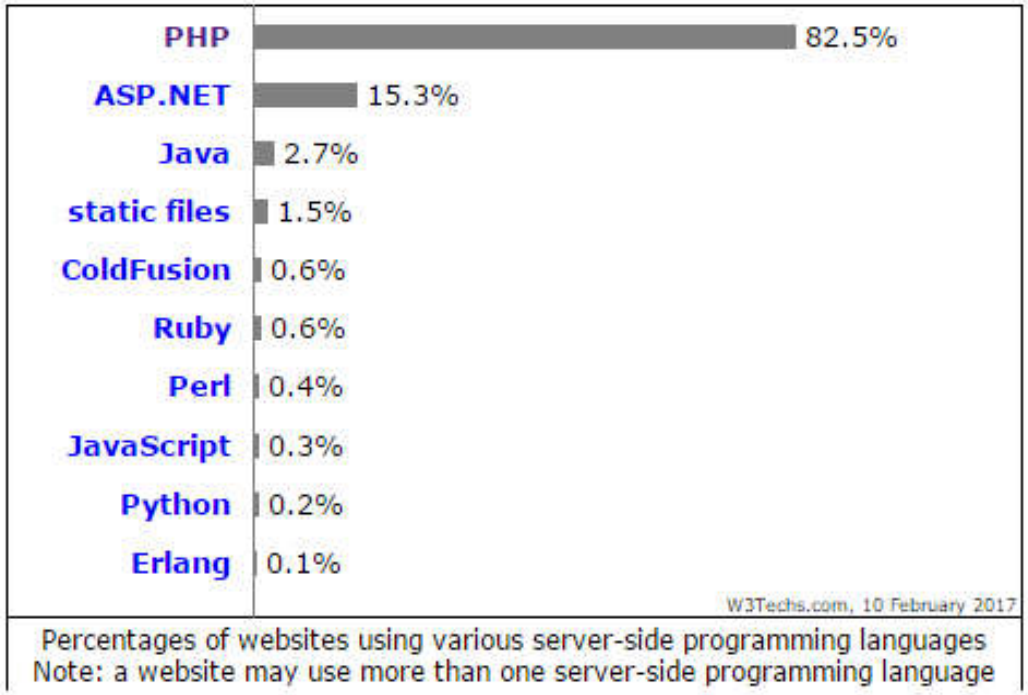
Cours **PHP5** Licence Informatique

Prof. Papa DIOP

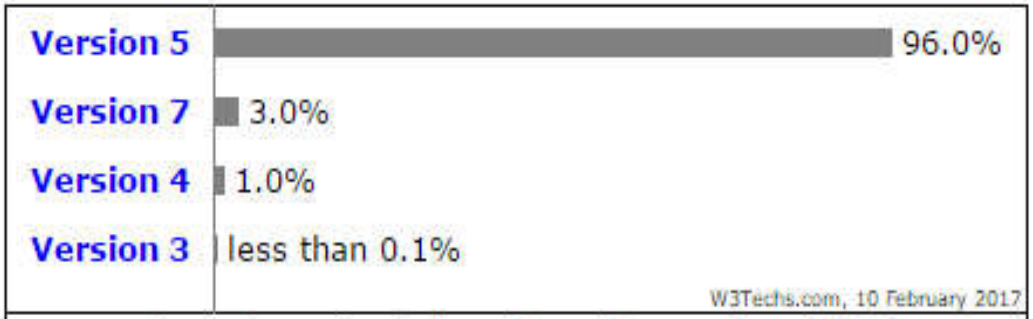
papaddiop@gmail.com

POO : Programmation orientée objet

Motivations du choix de PHP5

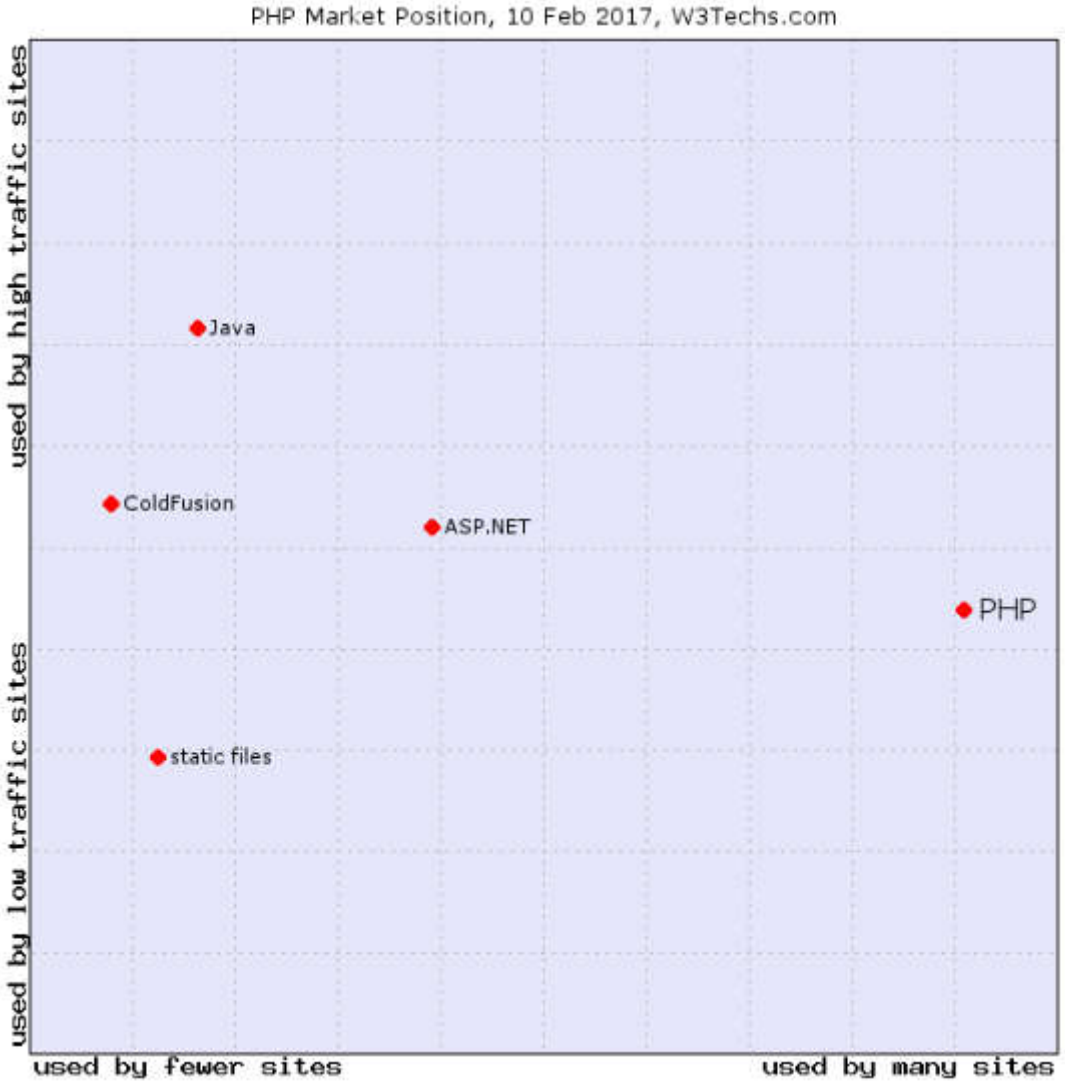


Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language



Percentages of websites using various versions of PHP

Motivations du choix de PHP5



Déclarer une classe en PHP

- **Une classe est un modèle de données**
  - famille d'objets, ou encore moule à objets
  - tous les objets d'une même classe partagent les mêmes attributs et les mêmes méthodes
- **le mot clé *class* permet de déclarer une classe d'objet.**

```
class voiture
{
   //code de la classe
}
```

déclarer des attributs (ou propriétés)

```
class voiture
{
   public $marque = "trabant";
}
```

- **public, protected, private** sont supportés
  - L'un des trois est obligatoire ou le mot clé **var** (⇔ public)
- Affectation et même déclaration facultatives!!

déclarer des méthodes (ou propriétés)

```
class voiture
{
   function freiner($force_de_freinage)
   {
   //code qui freine
   }
}
```

- **public**, **protected**, **private** **sont supportés**
- **Implicitement « public »**

déclarer des constantes

```
class voiture
{
   const ROUES_MOTRICES = 2;
}
```

- Locale à la classe
- convention classique : spécifier les constantes en majuscules dans le code pour mieux les identifier

Instanciation

```
class voiture
{
    const ROUES_MOTRICES = 2;
    public $marque;
    function freiner($force_de_freinage)
    {
    //code qui freine
    }
}
$MaVoiture = new voiture();
```

- Les parenthèses sont optionnelles si le constructeur ne nécessite pas de paramètre
- En PHP5 toute classe doit être déclarée avant d'être utilisée
  - Non obligatoire en PHP4

Accéder à un attribut

```
class voiture
{
   public $marque = "trabant";
}
$MaVoiture = new voiture();
echo $MaVoiture->marque;
// affiche trabant
```

Accéder à une méthode

```
class voiture
{
    function klaxonner()
    {
     return "tut tut!!";
    }
}
$MaVoiture = new voiture();
echo $MaVoiture->klaxonner();
// affiche "tut tut!"
```

Accèder à une constante

```
class voiture
{
    const ROUES = 2;
    public $marque;
    function freiner($force_de_freinage)
    {
     //code qui freine
    }
}
$MaVoiture = new voiture();
echo "ma voiture a ".$MaVoiture::ROUES." roues  ";
// affiche "ma voiture a 2 roues"
```

référence à l'objet en cours

```
class voiture
{
    public $vitesse = 0;
    function avance( $temps)
    {
     echo "avance de ".$temps*$this->vitesse." km en
    ".$temps." h";
    }
}

$MaVoiture = new voiture();
$MaVoiture->vitesse = 100; //la vitesse est de 100km/h
$MaVoiture->avance(2); // affiche "avance de 200 km en 2h"
```

Accès statique

- **appel direct via la classe, pas d'instanciaton**
- **utilisant de l'accesseur ::**
- **La référence à l'objet courant $this est alors interdite**

```
class voiture
{
    public $roues = 4;
    function statique()
    {
        return 4;
    }
    function dynamique()
    {
        return $this->roues;
    }
}
echo voiture::$roues; // affiche un message d'erreur
echo voiture::statique(); // affiche 4
echo voiture::dynamique(); // affiche un message d'erreur
```

Accès statique explicite

- **Via le mot clé *static***
- **Pour les attributs public private ou protected devient facultatif (public par défaut)**

```
class voiture
{
    static $roues = 4
    static function statique()
    {
      echo 4;
    }
}
echo voiture::$roues; // affiche 4
echo voiture::statique(); // affiche 4
$v = new voiture();
echo $v->roues; // affiche un message d'erreur
echo $v->statique(); //affiche 4 et est sensé afficher un message d'erreur
    de niveau E_STRICT
```

Accès à la classe courante

```
class voiture
{
    static $roues = 4;
    function static()
    {
     return self::$roues;
    }
}
echo voiture::static(); // CORRECT affiche 4
$v = new voiture();
echo $v->dynamique()."<br />"; //INCORRECT mais affiche
    4 et est sensé afficher un message d'erreur de niveau
    E_STRICT
```

Exception

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Division by zero.');
    }
    else return 1/$x;
}
try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}
```

L’héritage en POO

- permet de regrouper des parties communes dans une classe dite mère
- Toute classe dérivant de cette classe mère est appelée classe fille
- Une classe fille possède les attributs et méthodes de la classe mère ainsi que des attributs et des méthodes qui lui sont propres
- Pas d’héritage multiple en PHP5

Héritage en PHP5

```
class vehicule
{
    public $marque = "";
    function avance()
    {
      //code qui fait avancer le véhicule
    }
    function freine()
    {
      //code qui fait freiner le vehicule
    }
}
class voiture extends vehicule
{
    function klaxonne()
    {
      //code qui fait klaxonner la voiture
    }
    // la classe voiture possède un attribut marque, une méthode freine et une méthode
    avance par héritage
}
```

Redéfinition de méthode

- Les méthodes hérités peuvent être réécrites dans la classe fille

```
class voiture extends vehicule
{
    public $marque= « peugeot »;
    function klaxonne()
    {
     //code qui fait klaxonner la voiture
    }
    function avance()
    {
     //code qui fait avancer la voiture
    }
    function freine()
    {
     //code qui fait freiner la voiture
    }
}
```

Héritage strict

- **Les méthodes de la classe fille doivent avoir des prototypes compatibles avec ceux de la classe mère**
  - Les classes filles doivent pouvoir se manipuler comme la classe dont elles héritent
  - Il est possible d'ajouter des paramètres supplémentaires, à condition qu'ils soient facultatifs
  - Il est aussi possible de rendre facultatifs des paramètres en leur donnant une valeur par défaut
- **Seuls les constructeurs ne sont pas soumis à cette règles.**

Accéder à la classe parente

```
class vehicule
{
    $roues = 4;
    function affiche()
    {
      return "a ".$this->roues." roues";
    }
}
class voiture extends vehicule
{
    function affiche()
    {
      echo "cette voiture ".parent::affiche();
    }
}
$v = new voiture();
$v->affiche(); //affiche cette voiture a 4 roues
```

Contrôle d'accès

- **public:** une méthode ou attribut publique est accessible depuis toute votre application
- **private:** une méthode ou attribut privée n'est accessible que depuis l'intérieur de la classe
- **protected:** une méthode ou attribut publique est accessible depuis l'intérieur de la classe, et depuis toutes les classes dérivés

Contrôle d’accès héritage

- **Les accès aux attributs et méthodes sont redéfinissables dans les classes filles pourvu que la directive soit identique ou plus large**
  - Une méthode protégée peut être redéfinie comme protégée ou publique dans une classe fille.
  - Une méthode publique ne peut être que publique dans un classe fille.
  - Si une méthode privée est redéfinie dans une classe fille, PHP considèrera qu'il a deux méthodes de même nom simultanément dans la classe fille.
    - Si c'est une méthode de la classe mère qui y fait appel, elle accèdera à la méthode privée initiale.
    - Si c'est une méthode de la classe fille qui y fait appel, elle accèdera à la nouvelle implémentation.

Classe abstraite

- Début d'implémentation d'une classe
- Non instanciable
- Toute classe contenant au moins une méthode abstraite doit être déclarée abstraite
- Seule la signature d’une méthode abstraite est déclarée (pas d’implémentation)
- Les classes dérivées doivent implémentées toutes les méthodes abstraites
- Les classes dérivées ne sont pas obligées d’implémenter les méthodes déjà implémentées dans la classe parent, et peuvent posséder leurs propres méthodes

Classe abstraite

```
abstract class vehicule
{
   abstract function avancer();
   function tourner($sens)
   {
        echo "tourne à ".$sens."<br />";
   }
}
```

Héritage d'une classe abstraite

```
class voiture extends vehicule
{
function avancer()
    {
        echo "go!<br />";
}
function klaxonner()
    {
     echo "tut tut!<br />";
    }
}
```

interface

- API (Application Programming Interface) qui spécifie quelles méthodes et variables une classe peut implémenter, sans avoir à définir comment ces méthodes seront gérées
- Non instanciable
- Seule les signatures des méthodes d'une interface sont déclarées (pas d'implémentation)
- Toutes les méthodes de l'interface doivent être implémentées
- Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule

interface

```
interface peutAvancer
{
    public function avancer();
    public function freiner();
}
interface peutTourner
{
    public function tourneGauche();
    public function tourneDroite();
}
```

Implémentation d'une interface

```
class voiture implements peutAvancer, peutTourner
{
    public function avance()
    {
     echo "on avance";
    }
    public function freine()
    {
       echo "on freine";
    }
    public function tourneGauche()
    {
       echo "on tourne à gauche";
    }
    public function tourneDroite()
    {
       echo "on tourne à droite";
    }
    function klaxonner()
    {
      echo "tut tut!<br />";
    }
}
```

Abstract VS interface

- Aucun code n'est présent dans une interface
  - Une interface est donc une classe abstraite qui ne contiendrait que des méthodes abstraites
  - Une classe ne peut dériver que d'une classe abstraite mais peut implémenter plusieurs interfaces

Classe finale

- **Ces classes et méthodes ne pourront jamais être héritées**

```
class voiture extends vehicule
{
    final function avancer()
    {
     echo "on avance";
    }
}
final class voiture extends vehicule
{
    public function avancer()
    {
     echo "on avance";
    }
}
```

déréférencement des méthodes

```
class pneu
{
    public $marque = "michelin";
}
class voiture
{
    function Pneu()
    {
     return new pneu();
    }
}
$MaVoiture = new voiture();
echo "je roule avec des pneus ".$MaVoiture->Pneu()->$marque;
// affiche "je roule avec des pneus michelin"
```

affectation en PHP4

- par défaut les objets étaient passés par copie

```
class voiture
{
    public $marque = "trabant";
}
$MaVoiture = new voiture();
$MaVoiture2 = $MaVoiture;
$MaVoiture2->marque = 'ferrari';
echo $MaVoiture->marque; // affiche "trabant"
```

- **$MaVoiture** et **$MaVoiture2** sont des objets distincts.

Affectation par référence en PHP4

- **On pouvait toutefois forcer l'affectation par référence grâce à l'opérateur &.**

```
class voiture
{
    public $marque = "trabant";
}
$MaVoiture = new voiture();
$MaVoiture2 = &$MaVoiture;
$MaVoiture2->marque = 'ferrari';
echo $MaVoiture->marque; // affiche "ferrari"
```

- Pour qu'une fonction puisse modifier un objet, il fallait donc obligatoirement lui passe l'objet par référence :-/

Affectation en PHP5

Par défaut l'affectation est désormais faite pas référence pour les objets (l'opérateur & est maintenant implicite)

```
class voiture
{
   public $marque = "trabant";
}
$MaVoiture = new voiture();
$MaVoiture2 = $MaVoiture;
$MaVoiture2->marque = 'ferrari';
echo $MaVoiture->marque; // affiche "ferrari"
```

clonage en PHP5

- c'est le mot-clé clone qui permet d'effectuer une copie distinct d'un objet

```
class voiture
{
   public $marque = "trabant";
}
$MaVoiture = new voiture();
$MaVoiture2 = clone $MaVoiture;
$MaVoiture2->marque = 'ferrari';
echo $MaVoiture->marque; // affiche "trabant"
```

égalité

- l'égalité de deux objets (tous les attributs sont égaux) se teste par ==
- l'identité de deux objets (les deux variables référencent le même objet) se teste par ===

Appartenance d'un objet à une classe

```
class voiture{ }
$MaVoiture = new voiture();
if($MaVoiture instanceof voiture)
{
   echo "cet objet est une voiture";
}
else
{
   echo "cet objet n'est pas une voiture";
}
```

obtenir la classe d'un objet

```
class voiture{ }
$MaVoiture = new voiture();
echo "cet objet est une " . get_class($MaVoiture);
// affiche "cet objet est une voiture"
```

Obtenir la classe parente d'un objet

```
class vehicule{ }
class voiture extends vehicule{ }
$MaVoiture = new voiture();
echo get_class_parent($MaVoiture);
// affiche "vehicule"
echo get_class_parent('voiture');
// affiche "vehicule"
```

Quelques fonctions sur les objets

get_declared_classes() : retourne un tableau listant toutes les classes définies
class_exists($str) : vérifie qu'une classe dont le nom est passé en argument a été définie
get_class($obj), **get_parent_class** : retournent le nom de la classe de l'objet **$obj**
get_class_methods($str) : retourne les noms des méthodes de la classe **$str** dans un tableau
get_class_vars($str) : retourne les valeurs par défaut des attributs de la classe **$str** dans un tableau associatif
get_object_vars($obj) : retourne un tableau associatif des attributs de l'objet **$obj** les clés sont les noms des attributs et les valeurs, celles des attributs si elles existent
is_subclass_of($obj,$str) : détermine si l'objet **$obj** est une instanciation d'une sous-classe de **$str**, retourne VRAI ou FAUX
method_exists($obj,$str) : vérifie que la méthode **$str** existe pour une classe dont **$obj** est une instance, retourne VRAI ou FAUX

Réflexion

- PHP 5 introduit API de réflexion complète qui permet de faire du *reverse-engineering* sur les classes, les interfaces, les fonctions et les méthodes tout comme les extensions. L'API de réflexion permet également d'obtenir les commentaires de la documentation pour les fonctions, les classes et les méthodes.

http://fr2.php.net/manual/fr/book.reflection.php

PHP 5.3

- Extension SPL (Standard PHP Library) intégrée en natif
  - http://www.php.net/~helly/php/ext/spl/
- Les namespaces sont supportés \ comme caractère séparateur
  - http://fr.php.net/manual/fr/language.namespaces.rationale.php
  - http://jcrozier.developpez.com/tutoriels/web/php/espaces-noms--mots-cles-autoloading/

Fonctions Anonymes ou lambdas

- Comme en javascript (« à la jQuery »)
- Affectation à une variable

```
$hello = funtcion(){echo 'hello world'}
```

- Appel

```
$hello();
call_user_func($hello);
```

- Passé en argument d'une fonction

```
Function foo(Closure $func){$func();}
foo($hello);
```

http://blog.pascal-martin.fr/post/php-5.3-1-closures-et-lambdas
http://blog.pascal-martin.fr/post/php-5.3-2-closures-et-lambdas
http://www.slideshare.net/fabpot/playing-with-php-53