Chapitre 4. PHP

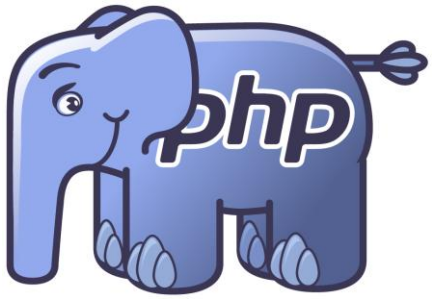
Prof. Amina ADADI

2ème Année Génie Logiciel

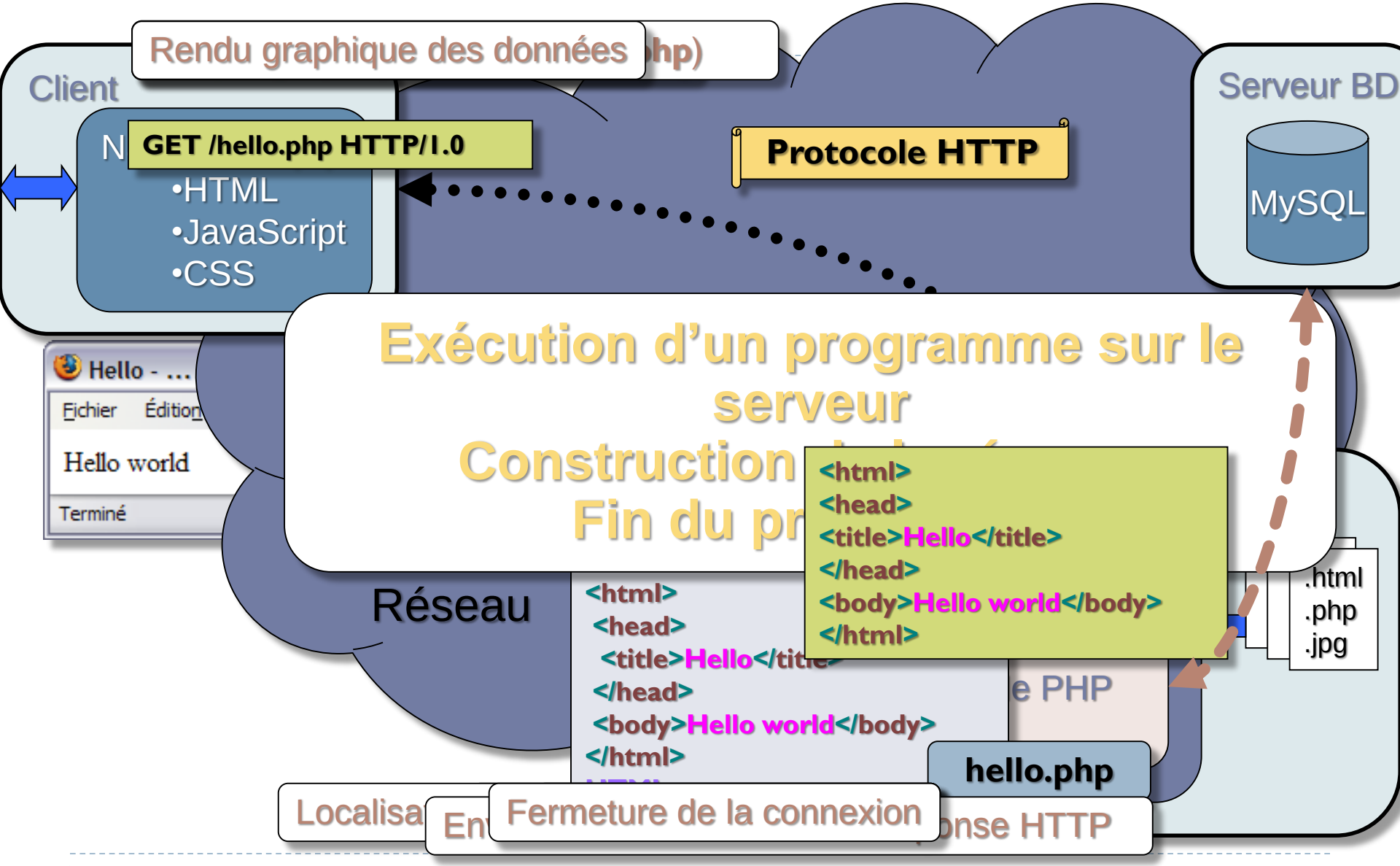
Année universitaire 2019-2020

Le langage PHP

- Langage de script (interprété) . Principalement utilisé côté serveur
- Langage impératif avec programmation orientée objet possible
- Créé en 1994-1995 par Rasmus Lerdorf
- Acronyme initial : **P**ersonal **H**ome **P**age
- Acronyme récursif : **P**HP: **H**ypertext **P**reprocessor
- Langage multi plate-forme (UNIX / Windows…)
- Open Source
- Versions actuelles : 7.3.x et 7.4.x



Fonctionnement de PHP en mode serveur



Syntaxe de PHP

- La syntaxe de PHP est celle de la famille « C » (C, C++, Java, …)
- Chaque instruction se termine par « ; »
- Commentaires:

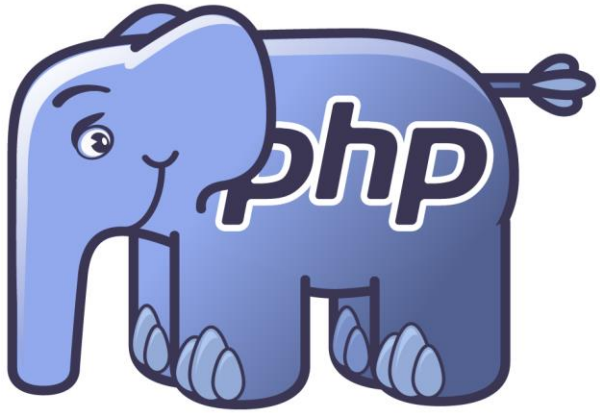
```
/* jusqu'au prochain */
// jusqu'à la fin de la ligne
# jusqu'à la fin de la ligne
```

- Délimitation du code PHP dans le fichier .php

```
<?php Code PHP ?>
```

PHP Rappel des bases:

Variables & Opérateurs
Structure de contrôle et boucle
Tableaux
Formulaires
Fonctions



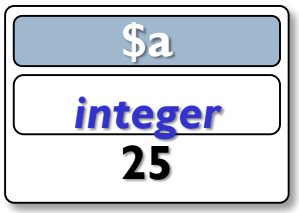
Les variables

- Tout identificateur commence par « **$** »
- Les affectations sont réalisées grâce à « **=** »
- Le typage est dynamique. Les types sont :
  - Numérique entier : **12** ou réel : **1.54**
  - Chaîne: **"Hello"** ou **'Bonjour'**
  - Booléen: **true**, **false**
  - Tableau: $tab[**2**]=**12**
  - Objet
  - Ressource
  - **NULL**
  - Callable
- Les variables ne sont pas explicitement déclarées: La « déclaration » d'une variable correspond à sa première affectation: **$test = 12**

Les variables

- Le cast :`$variable = (nom_du_type) valeur:`

```
$a = (integer) "25" ; =>
```



- Définition d'une constante:

```
define(« nom_constante",  "valeur");
```

- **Chaine de caractère:**

guillemets doubles -> interprétation,

guillemets simples -> pas interprétation

Guillemets doubles

```
$a="chaîne" ;
$b="voici une $a";
```

voici une chaîne

Guillemets simples

```
$a='chaîne' ;
$b='voici une $a';
```

voici une $a

Les variables prédéfinies

$GLOBALS : Contient le nom et la valeur de toutes les variables globales du script. Les noms des variables sont les clés de ce tableau.

$_COOKIE: Contient le nom et la valeur des cookies enregistrés sur le poste client. Les noms des cookies sont les clés de ce tableau .

$_ENV: Contient le nom et la valeur des variables d'environnement qui sont changeantes selon les serveurs.

$_FILES :Contient le nom des fichiers téléchargés à partir du poste client.

$_GET: Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode GET. Les noms des champs du formulaire sont les clés de ce tableau.

$_POST: Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode POST. Les noms des champs du formulaire sont les clés de ce tableau.

$_REQUEST: Contient l'ensemble des variables superglobales $_GET, $_POST, $_COOKIE et $_FILES.

$_SESSION : Contient l'ensemble des noms des variables de session et leurs valeurs.

$_SERVER: Contient les informations liées au serveur Web, tel le contenu des en-têtes HTTP ou le nom du script en cours d'exécution. Retenons les variables suivantes :

$_SERVER["HTTP_ACCEPT_LANGUAGE"]: qui contient le code de langue du navigateur client.

$_SERVER["HTTP_COOKIE"]: qui contient le nom et la valeur des cookies lus sur le poste client.

$_SERVER["HTTP_HOST"]: qui donne le nom de domaine.

$_SERVER["SERVER_ADDR"]:qui indique l'adresse IP du serveur.

$_SERVER["PHP_SELF"]: qui contient le nom du script en cours. Nous l'utiliserons souvent dans les formulaires.

$_SERVER['REQUEST_METHOD'] :détermines est ce que la requête utilise la méthode POST ou GET.

La commande echo

- **basiquement**, permet d'envoyer du texte au navigateur du client (« écrire » la page au format HTML résultant de l'interprétation de PHP)
  - `echo "Bonjour" ;`
  - `$nom="Marcel" ; echo "Bonjour $nom" ;`
- **Exactement**, permet d'envoyer des octets au navigateur du client
  - Contenu HTML, XML, CSS, JavaScript, …
  - Données d'une image
  - Contenu d'un ficher PDF, Flash, etc.
- **Précisément**, remplit le corps de la réponse HTTP (la charge utile) et engendre la production et l'envoi de la réponse HTTP complète.

Les opérateurs: Concaténation de chaînes

- Permet d'assembler plusieurs chaînes
- Réalisé grâce à l'opérateur point : « . »

```
"Bonjour " .  "HAMED"
```

→ vaut `"Bonjour AHMED"`

```
$nb = 6*2 ;
"Acheter " . $nb . " oeufs"
```

→ vaut `"Acheter 12 oeufs"`

Les opérateurs arithmétiques

`$a + $b`	Somme
`$a - $b`	Différence
`$a * $b`	Multiplication
`$a / $b`	Division
`$a % $b`	Modulo (Reste de la division entière)

Les opérateurs de comparaison

$a == $b	Vrai si égalité entre les valeurs de $a et $b
$a != $b	Vrai si différence entre les valeurs de $a et $b
$a < $b	Vrai si $a inférieur à $b
$a > $b	Vrai si $a supérieur à $b
$a <= $b	Vrai si $a inférieur ou égal à $b
$a >= $b	Vrai si $a supérieur ou égal à $b
$a === $b	Vrai si $a et $b identiques (valeur et type)
$a !== $b	Vrai si $a et $b différents (valeur ou type)

Les opérateurs logiques

[Expr1] and **[Expr2]**	Vrai si **[Expr1]** et **[Expr2]** sont vraies
[Expr1] && **[Expr2]**	idem
[Expr1] or **[Expr2]**	Vrai si **[Expr1]** ou **[Expr2]** sont vraies
[Expr1] || **[Expr2]**	idem
[Expr1] xor **[Expr2]**	Vrai si **[Expr1]** ou **[Expr2]** sont vraies mais pas les deux
! **[Expr1]**	Vrai si **[Expr1]** est non vraie

Structure de contrôle if… else

```php
<?php
$prix=55;
if($prix>100)
{
echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 10 % </b>
<br />";
echo "Le prix net est de ",$prix*0.90;
}
else
{
echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 5 %</b><br />";
echo "<h3>Le prix net est de ",$prix*0.95,"</h3>";
}
?>
```

Ternaire

```php
$action = empty($_POST['action']) ? 'default' : $_POST['action'];
```

Structure de contrôle Switch cases

```php
<?php
$dept=75;
switch($dept)
{
//Premier cas
case 75:
case "Capitale":
echo "Paris";
break;
//Deuxième cas
case 78:
echo "Hauts de Seine";
break;
//Troisième cas
case 93:
case "Stade de france":
echo "Seine Saint Denis";
break;
//la suite des départements...
//Cas par défaut
default:
echo "Département inconnu en Ile de France";
break;
}
?>
```

Les boucles

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

for

While
Do While

```php
<?php
$n=1;
while($n%7!=0 )
{
$n = rand(1,100);
echo $n,"  /";
}

/**********************/
<?php
do
{
$n = rand(1,100);
echo $n,"  / ";
}
while($n%7!=0);
?>
```

Tableaux « classiques » : indexés

- Création / initialisation:

```php
$tab1 = array(12, "fraise", 2.5) ;

$tab1_bis = [ 12, "fraise", 2.5 ] ;

$tab2[] = 12 ;
$tab2[] = "fraise" ;
$tab2[] = 2.5 ;

$tab3[0] = 12 ;
$tab3[1] = "fraise" ;
$tab3[2] = 2.5 ;
```

Clé	Valeur
0	12
1	"fraise"
2	2.5

Tableaux associatifs : syntaxe

```php
$tab5['un'] = 12 ;
$tab5['trois'] = "fraise" ;
$tab5["deux"] = 2.5 ;
$tab5[42] = "el5" ;
```

Clé	Valeur
"un"	12
"trois"	"fraise"
"deux"	2.5
42	"el5"

```php
$tab6 = array(  'un'   => 12,
                'trois' => "fraise",
                "deux"  => 2.5,
                   42   => "el5") ;

$tab7 = [ 'un'    => 12, 'trois' => "fraise",
          "deux" => 2.5, 42     => "el5" ]  ;
```

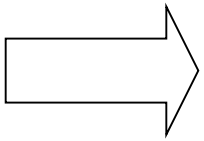
Parcours de tableau: `foreach`

Des tableaux associatifs ne peuvent PAS être parcourus grâce aux indices des cases contenant les éléments…

```
foreach ($tableau as $element)
{
    /* Bloc d'instructions répété pour chaque élément de $tableau */
    /* Chaque élément de $tableau est accessible grâce à $element */
}
```

Parcours de tableau : `foreach`

```
…
$tab4[0] = 12 ;
$tab4[6] =
  "fraise" ;
$tab4[2] = 2.5 ;
$tab4[5] = "el5" ;
foreach($tab4 as
  $v)
{
   echo "<p>Val:
  $v\n";
}
…
```



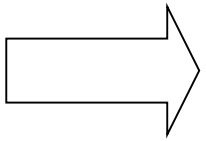
```
…
<p>Val: 12\n
<p>Val: fraise\n
<p>Val: 2.5\n
<p>Val: el5\n
…
```

Parcours de tableau: **foreach**

```
foreach($tableau as $cle => $element)
{
  /* Bloc d'instructions répété pour
     chaque élément de $tableau */
  /* Chaque élément de $tableau est
     accessible grâce à $element */
  /* La clé d'accès à chaque élément est
     donnée par $cle */
}
```

Parcours de tableau: **foreach**

```
<?php
<!doctype html>
<html >
  <head><title>foreach
  clé</title>
  </head>
<body>
HTML;
$tab6 = array('un'      =>
  12,
                'trois'   =>
  "fraise",
                "deux"    =>
  2.5,
                42        =>
  "el5") ;
foreach ($tab6 as $cle =>
  $val)
{
  $html .= "<p>tab[$cle]:
  $val\n" ;
}
echo $html .
  "</body>\n</html>" ;
```



```
…
<p>tab[un]:12\n
<p>tab[trois]:fraise\
n
<p>tab[deux]:2.5\n
<p>tab[42]:el5\n
…
```

Opérations tableau

Création d'un tableau à partir d'une chaine de caractère

```php
<?php
$chaine="La cigale et la fourmi";
$tabmot = explode(" ",$chaine);
print_r($tabmot);
?>
```

Extraire une partie d'un tableau

```
$sous_tab = array_slice(array $tab,int ind, int nb);
```

le tableau $sous_tab contient nb éléments du tableau initial extrait en commençant à l'indice ind.

Ajouter et enlever des éléments

```
int array_push($tab, valeur1, valeur2,…, valeurN)
```

ajoute en une seule opération les N éléments passés en paramètres à la fin du tableau désigné par la variable $tab. La fonction retourne également le nouveau nombre d'éléments du tableau modifié. **array_unshift** ajoute les éléments au début du tableau. **array_pop ()** dépile et retourne le dernier élément du tableau. array_shift quant à elle défile.

Opérations sur les tableaux

Fusionner des tableaux

```
$sous_tab = array_slice(array $tab,int ind, int nb);$tab = array_merge($tab1,$tab2,…,$tabN)
```

Cette fonction retourne dans $tab l'ensemble des éléments présents dans les tableaux $tab1, $tab2, …, $tabN. Les tableaux passés en paramètres sont tous sauvegardés tels qu'ils étaient avant l'appel de la fonction.

Intersection et différence de deux tableaux

```
array array_intersect($tab1,$tab2)
```

Cette fonction retourne un tableau contenant tous les éléments communs aux tableaux $tab1 et $tab2.

```
array_diff($tab1,$tab2)
```

retourne un tableau contenant les éléments présents dans le premier paramètre mais pas dans le second

Opérations sur les tableaux

Plusieurs autres opérations:

- array sort($tab)
- array array_reverse($tab)
- void asort(array $tab)
- array_count_values(array $tab)
- array array_filter(array $tab,string nom_fonction)
- array array_flip(array $tab)…
- array **array_keys** (array *input*)
- array **array_values** (array *input*)
- string **implode** (string *glue*, array *pieces*)
- bool **in_array** (mixed *needle*, array *haystack* [, bool *strict*])

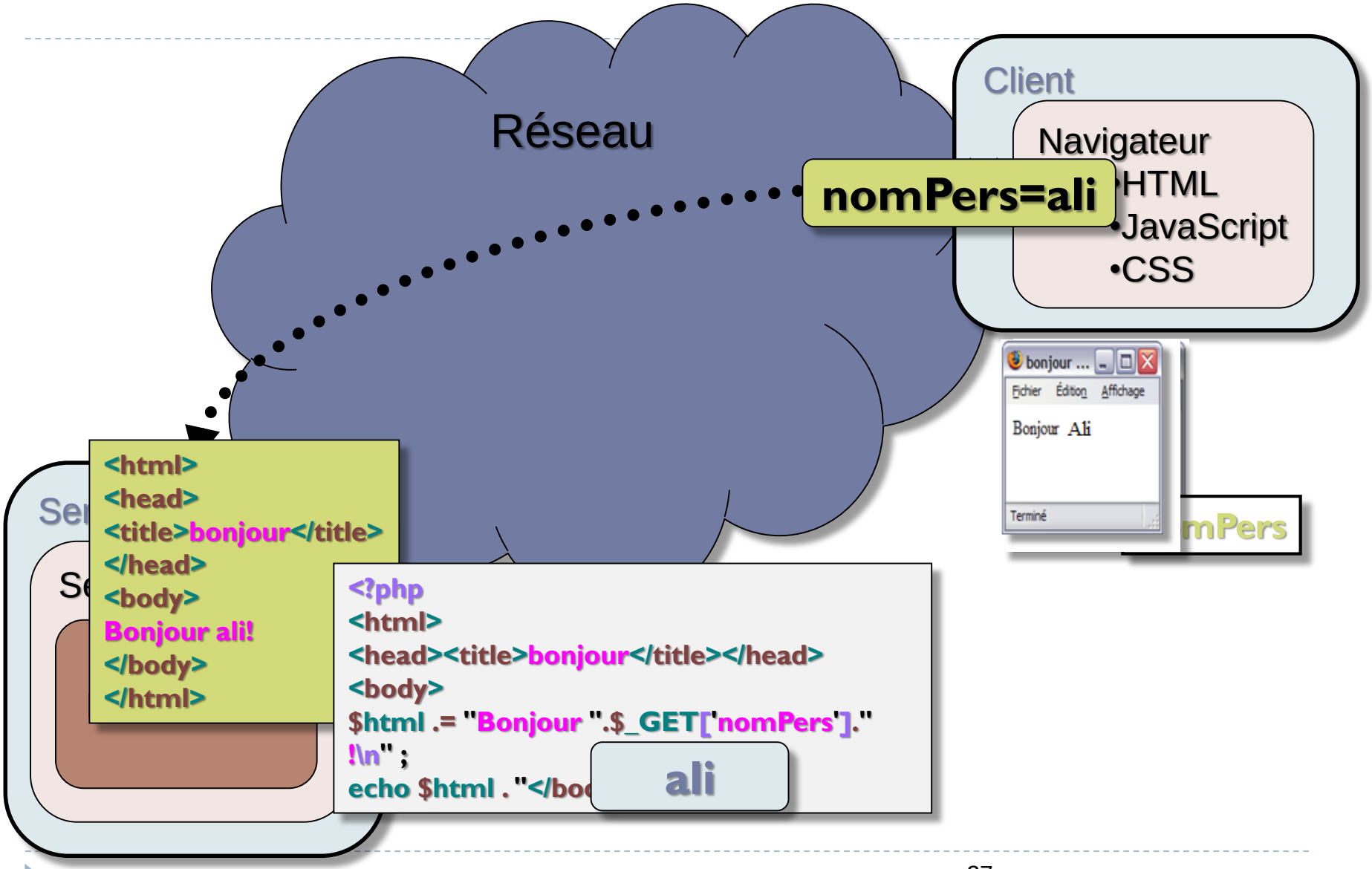
Traitement des données de formulaires

- PHP permet de **traiter les données** saisies grâce à un formulaire HTML si le champ **ACTION** du formulaire désigne une page PHP du serveur.

- Après récupération par le serveur Web, les données sont contenues dans l'une des variables superglobales de type tableau associatif **$_GET** ou **$_POST** selon la méthode de la requête (ou **$_REQUEST** qui reçoit le contenu de **$_GET** et **$_POST**).

- La valeur peut être trouvée grâce à une clé **qui porte le même nom** que le champ du formulaire de la page HTML de saisie.

Traitement des données de formulaires



Exemple – Formulaire HTML

```
<!doctype html>
<html ">
 <head>
   <title>formulaire</title>
 </head>
 <body>
   <form action="valide1.php"
  method="get">
      Nom: <input type="text"
  name="nomPers">
      <input type="submit"
  value="Valider">
   </form>
 </body>
</html>
```

Exemple – Traitement en PHP

```php
<?php
<!doctype html>
<html>
 <head><title>bonjour</title></he
 <body>
if (isset($_GET['nomPers']))
      if (!empty($_GET['nomPers'])
          $html .= "Bonjour " .
  $_GET['nomPers'] . "\n" ;
       else
            $html .= "Aucune valeur saisie\n";
   else
       $html .= "Utilisation incorrecte\n" ;
echo $html . "</body>\n</html>" ;
```

$_GET['nomPers'] est-il défini ?

$_GET['nomPers'] est-il vide ?

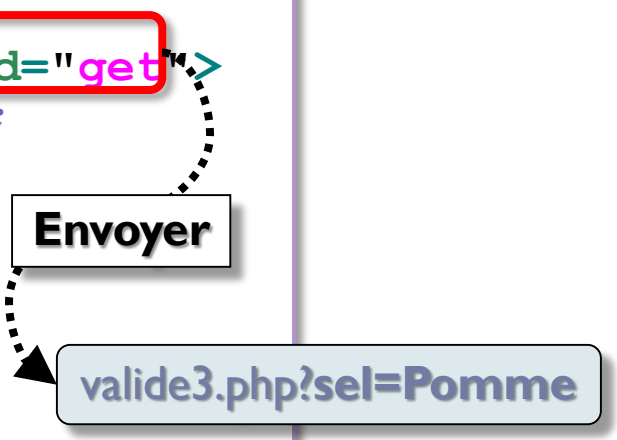
Récupération des valeurs multiples

Formulaires contenant des champs « SELECT »

Récupération des valeurs multiples

```
<!doctype html>
<html lang="fr">
<head>
  <title>Formulaire de saisie des
  fruits</title>
</head>
<body>
  <form action="valide3.php" method="get">
      Choisissez des fruits: 
      <select name="sel">
          <option>Fraise
          <option>Pomme
          <option>Poire
          <option>Banane
          <option>Cerise
      </select>
      <input type="submit" value="envoyer">
  </form>
</body>
</html>
```

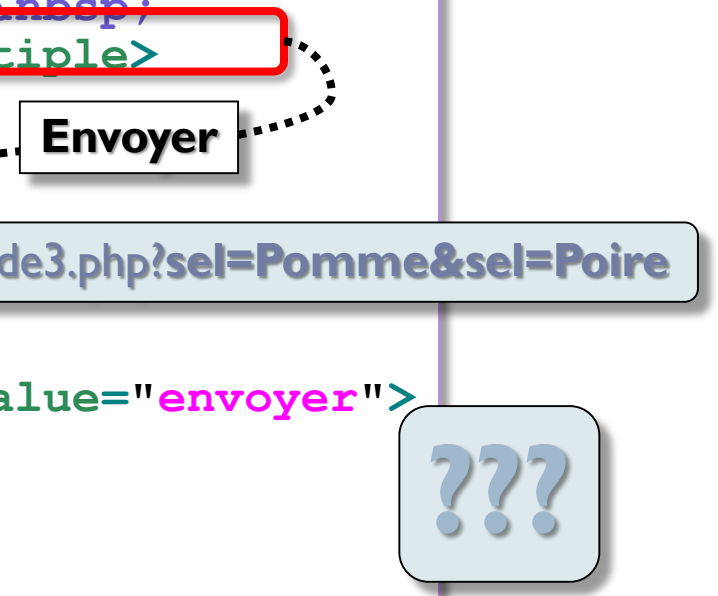
« SELECT unique»



Récupération des valeurs multiples

```
<!doctype html>
<html lang="fr">
<head>
  <title>Formulaire de saisie des
  fruits</title>
</head>
<body>
  <form action="valide3.php" method="get">
      Choisissez des fruits: 
      <select name="sel" multiple>
          <option>Fraise
          <option>Pomme
          <option>Poire
          <option>Banane
          <option>Cerise
      </select>
      <input type="submit" value="envoyer">
  </form>
</body>
</html>
```

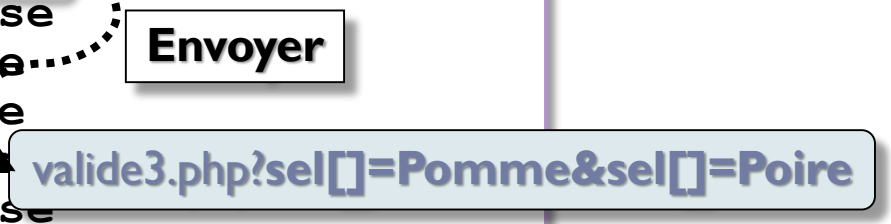
« SELECT multiple»



Récupération des valeurs multiples

```
<!doctype html>
<html lang="fr">
<head>
  <title>Formulaire de saisie des
  fruits</title>
</head>
<body>
  <form action="valide3.php" method="get">
      Choisissez des fruits: 
      <select name="sel[]" multiple>
          <option>Fraise
          <option>Pomme
          <option>Poire
          <option>Banane
          <option>Cerise
      </select>
      <input type="submit" value="envoyer">
  </form>
</body>
</html>
```

SELECT multiple

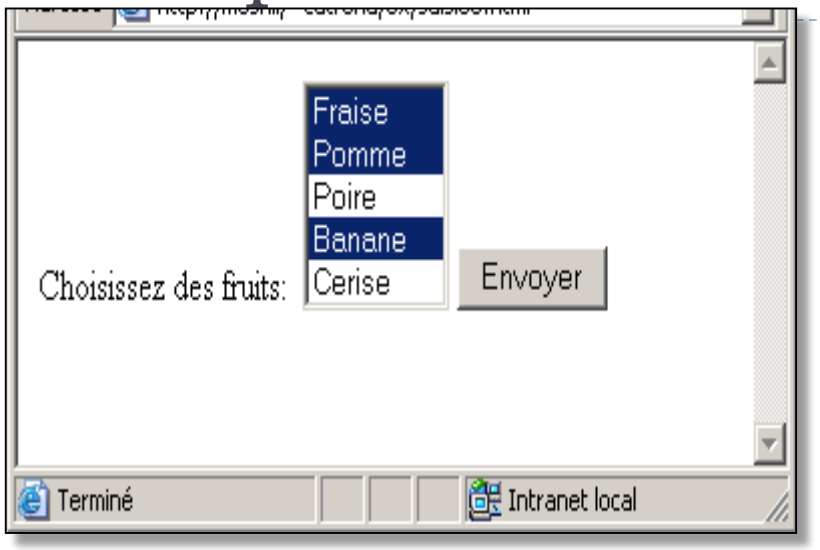


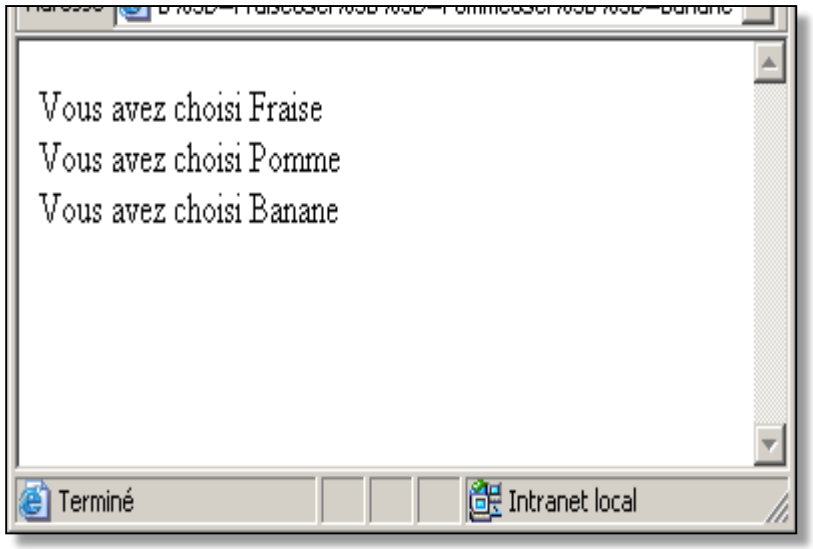
Récupération des valeurs multiples

```php
<?php
<!doctype html>
<html>
<head>
  <title>Liste de fruits</title>
</head>
<body>
  if (isset($_GET['sel']) && is_array($_GET['sel']))
  { /* La variable $_GET['sel'] est définie et elle
  est un tableau */
      foreach($_GET['sel'] as $fruit)
          $html .= "<p>Vous avez choisi $fruit\n" ;
  }
  else
      $html .= "<p>Vous n'avez pas choisi de
  fruit\n" ;
$html .= "</body>\n</html>" ;
echo $html ;
```

$_GET['sel'] est un tableau

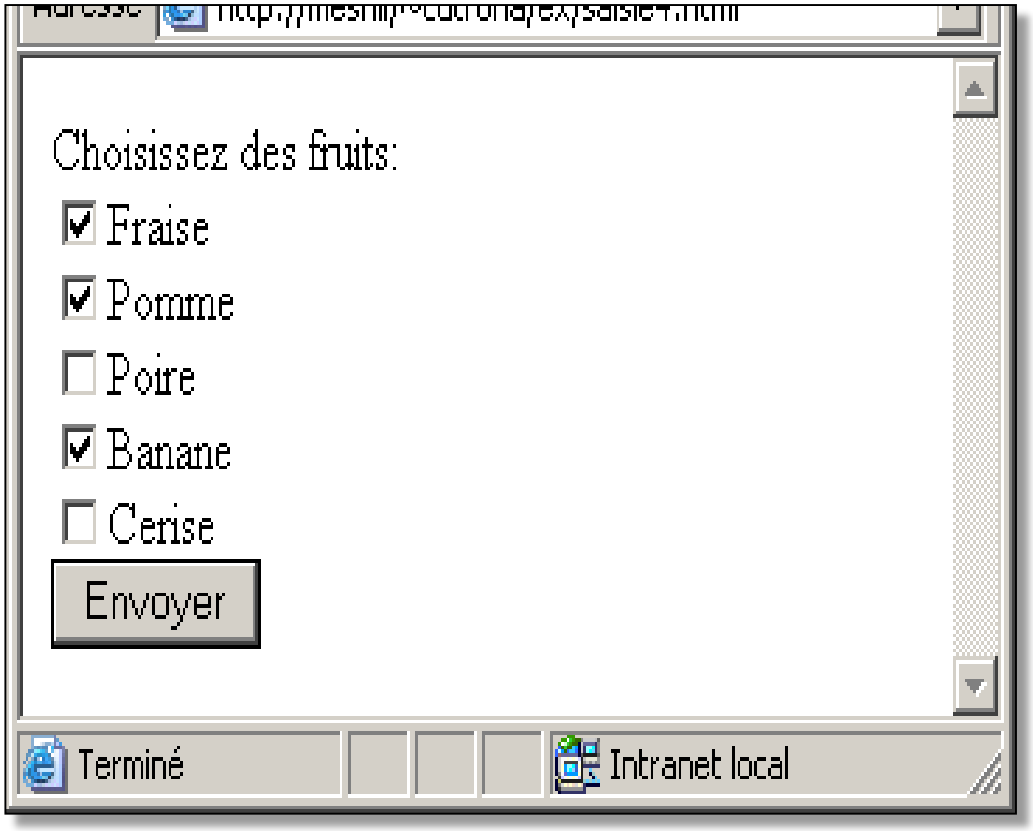
Récupération des valeurs multiples





Récupération des valeurs multiples

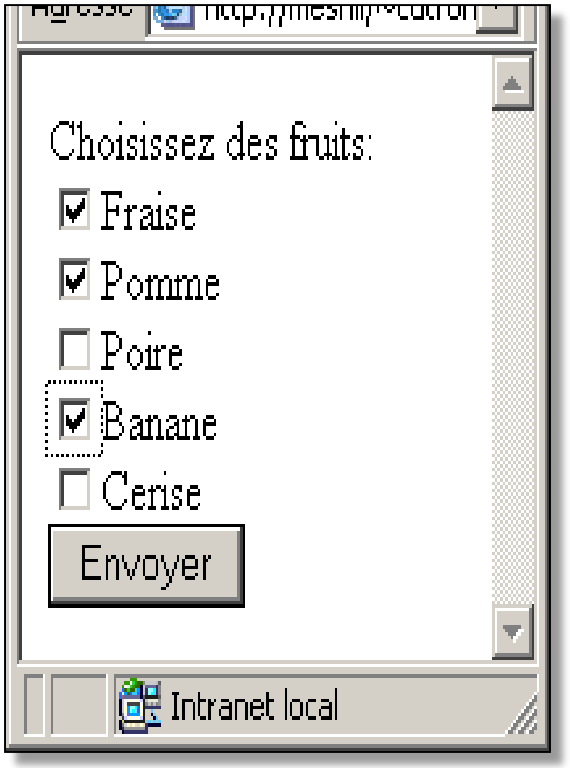
Formulaires contenant des champs « CHECKBOX »

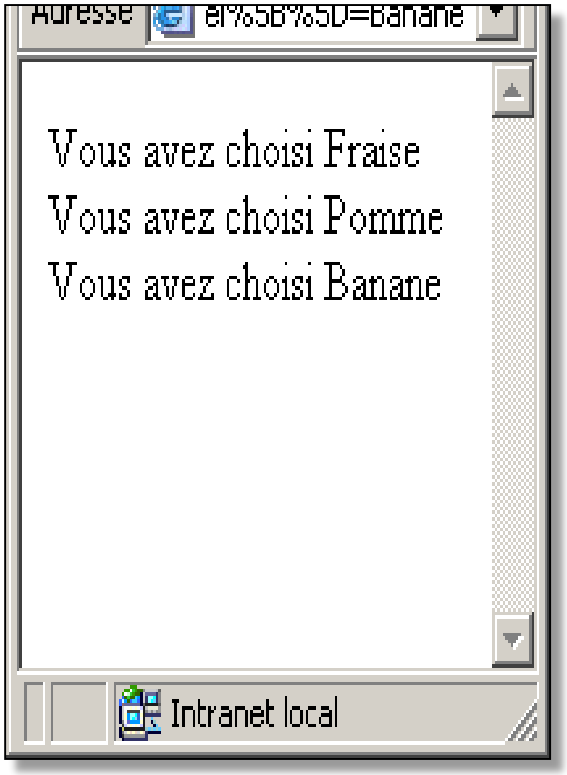


Récupération des valeurs multiples

```
<!doctype html>
<html >
 <head>
  <title>Formulaire de saisie des fruits</title>
 </head>
 <body>
  <form name="formu" action="valide3.php" method="get">
   Choisissez des fruits :
   <label><input type="checkbox" name="sel[]"
  value="Fraise">Fraise</label>
   <label><input type="checkbox" name="sel[]" value="Pomme"
  >Pomme</label>
   <label><input type="checkbox" name="sel[]" value="Poire"
  >Poire</label>
   <label><input type="checkbox" name="sel[]"
  value="Banane">Banane</label>
   <label><input type="checkbox" name="sel[]"
  value="Cerise">Cerise</label>
   <input type="submit" value="Envoyer">
  </form>
 </body>
</html>
```

Récupération des valeurs multiples





Résultat

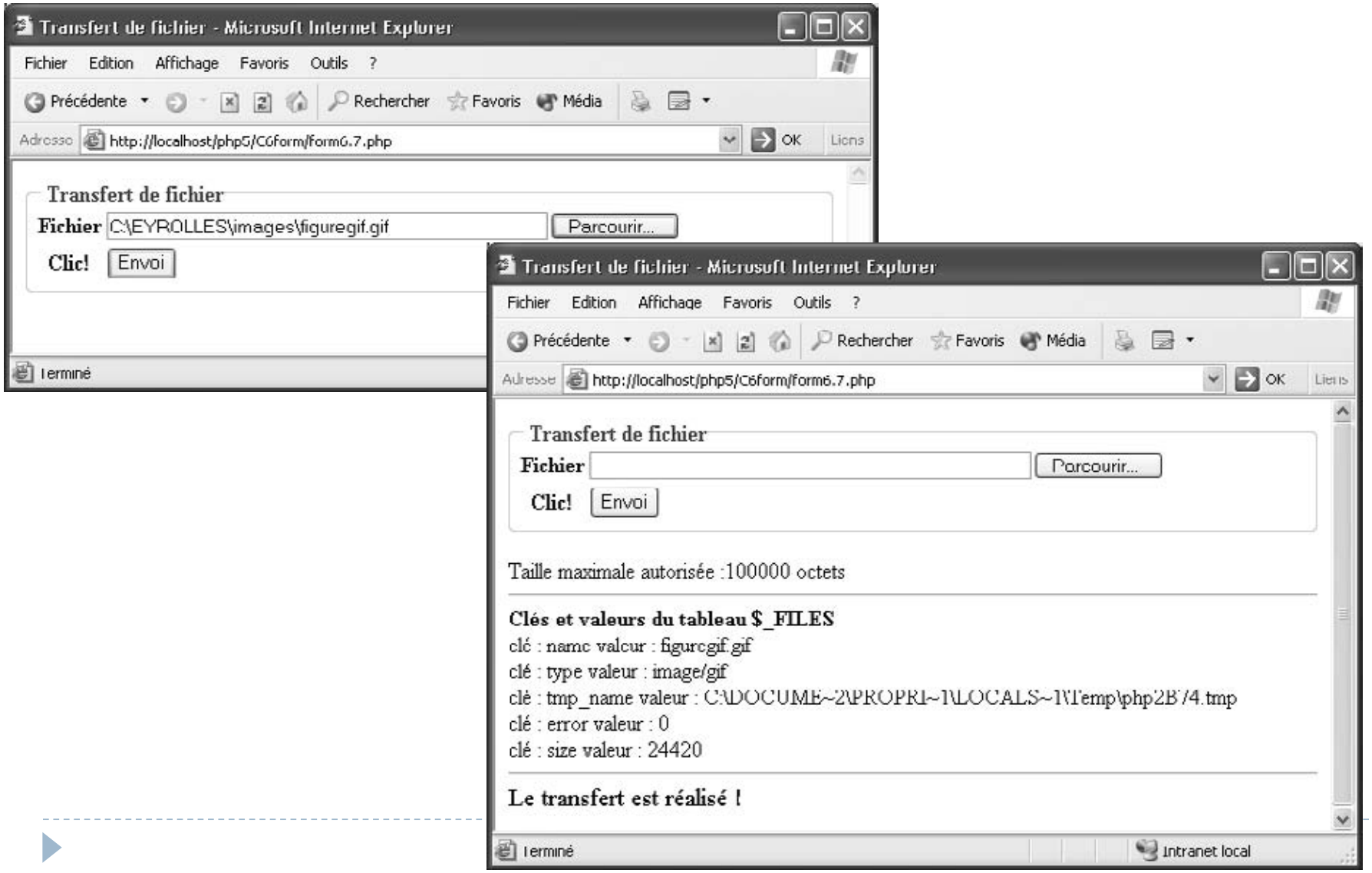
Transfert de fichier vers le serveur

```
<body>
<form action="form6.php" method="post" >
<fieldset>
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
<legend><b>Transfert de fichier</b></legend>
<table>
<tbody>
<tr>
<th>Fichier</th>
<td> <input type="file" name="fich" accept="image/gif" size="50"/></td>
</tr>
<tr>
<th>Clic!</th>
<td> <input type="submit" value="Envoi" /></td>
</tr>
</tbody>
</table>
</fieldset>
</form>
```

Transfert de fichier vers le serveur

```php
<?php
if(isset($_FILES['fich']))
    {
echo "Taille maximale autorisée :",$_POST["MAX_FILE_SIZE"],
" octets<hr / >";
echo "<b>Clés et valeurs du tableau \$_FILES </b><br />";
foreach($_FILES["fich"] as $cle => $valeur)
{
echo "clé : $cle valeur : $valeur <br />";
}
//Enregistrement et renommage du fichier
$result=move_uploaded_file($_FILES["fich"]["tmp_name"],"imagephp.gif");
if($result==TRUE)
{echo "<hr /><big>Le transfert est réalisé !</big>";}
else
{echo "<hr /> Erreur de transfert n°",$_FILES["fich"]["error"];}
}
?>
```

Transfert de fichier vers le serveur



Gérer les boutons d'envoi multiples

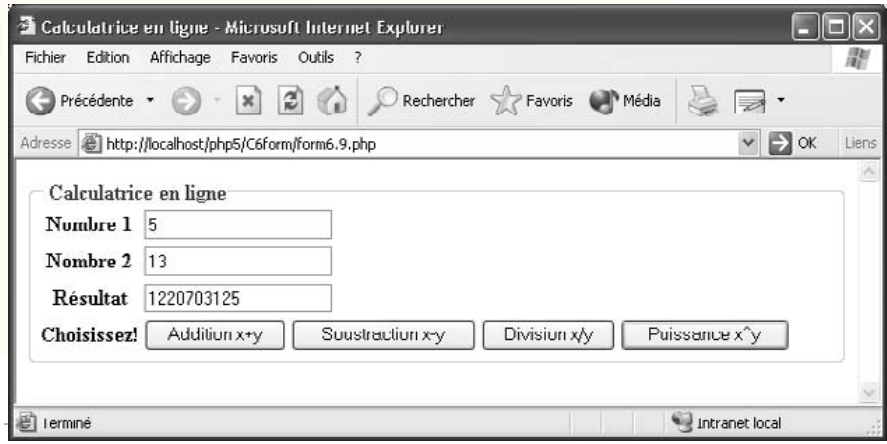
L'utilisation de plusieurs boutons submit dans un même formulaire permet de déclencher des actions différentes en fonction du bouton activé par l'utilisateur. Il est nécessaire pour cela que les boutons aient le même nom et que la sélection de l'action se fasse en fonction de la valeur associée à chaque bouton *via l'attribut value.*

Gérer les boutons d'envoi multiples

```
<tbody>
<tr>
<th>Nombre X</th>
<td> <input type="text" name="nb1" value="<?php if(isset($_POST["nb1"]))
echo $_POST['nb1'];else echo'' ?>"/>
</td>
</tr>
<tr>
<th>Nombre Y</th>
<td> <input type="text" name="nb2" value="<?php if(isset($_POST["nb2"]))
echo $_POST['nb2'];else echo'' ?>"/>
</td>
</tr>
<tr>
<th>Résultat </th>
<td> <input type="text" name="result" value="<?php if(isset($resultat))
echo $resultat;else echo''?>"/>
</td>
</tr>
<tr>
<th>Choisissez!</th>
<td>
<input type="submit" name="calcul" value="Addition x+y" />
<input type="submit" name="calcul" value="Soustraction x-y" />
<input type="submit" name="calcul" value="Division x/y" />
<input type="submit" name="calcul" value="Puissance x^y" />
</td>
</tr>
</tbody>
```

Gérer les boutons d'envoi multiples

```php
<?php
if(isset($_POST["calcul"])&&isset($_POST["nb1"])&&isset($_POST["nb2"]))
{
switch($_POST["calcul"])
{
case "Addition x+y":
$resultat= $_POST["nb1"]+$_POST["nb2"];
break;
case "Soustraction x-y":
$resultat= $_POST["nb1"]-$_POST["nb2"];
break;
case "Division x/y":
$resultat= $_POST["nb1"]/$_POST["nb2"];
break;
case "Puissance x^y":
$resultat= pow($_POST["nb1"],$_POST["nb2"]
break;
}
}
else
{
echo "<h3>Entrez deux nombres : </h3>";
}
?>
```



Exercice 1

Créez un formulaire demandant la saisie d'un prix HT et d'un taux de TVA. Le script affiche le montant de la TVA et le prix TTC dans deux zones de texte créées dynamiquement.

Fonctions utilisateur

- Description d'une fonctionnalité dépendant éventuellement de paramètres et retournant éventuellement un résultat
- Définition

```
function moyenne($a,$b)
{
    return ($a+$b)/2. ;
}
```

- Utilisation

```
$resultat = moyenne(2,4) ;
echo $resultat ; // vaut 3
```

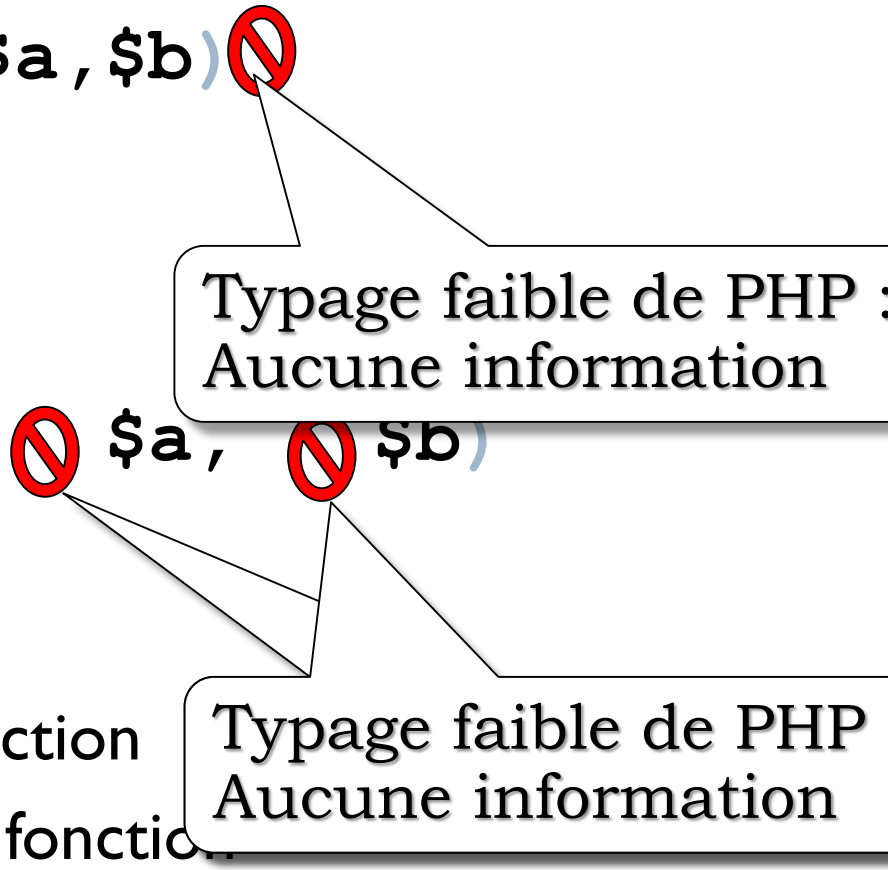
Fonctions utilisateur PHP ≤ 4

- Valeur de retour

```
function moyenne($a,$b)
{   …   }
```

- Arguments

```
function moyenne( $a,  $b)
{   …   }
```



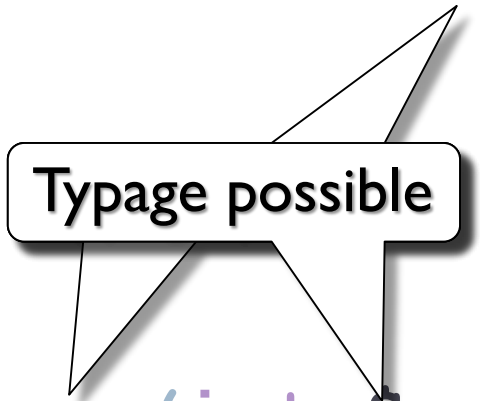
- Variables définies dans une fonction

➔ Visibles et accessibles dans la fonction

Fonctions utilisateur PHP ≥ 5

- Valeur de retour

```
function moyenne($a, $b) : int
{   …   }
```



- Arguments

```
function moyenne(int $a, int $b)
{   …   }
```

- Fonctionnement identique pour les méthodes
- Possible de typer paramètre et valeur de retour

Typage des paramètres / retours PHP ≥ 5

Type	Description	Version PHP minimum
Nom de la Classe/interface	Le paramètre doit être une instanceof de la classe ou interface donnée.	PHP 5.0.0
self	Le paramètre doit être une instanceof de la même classe qui a défini la méthode. Ceci ne peut être utilisé que sur les méthodes des classes et des instances.	PHP 5.0.0
array	Le paramètre doit être un array.	PHP 5.1.0
callable	Le paramètre doit être un callable valide.	PHP 5.4.0
bool	Le paramètre doit être un boolean.	PHP 7.0.0
float	Le paramètre doit être un nombre flottant (float).	PHP 7.0.0
int	Le paramètre doit être un integer.	PHP 7.0.0
string	Le paramètre doit être une string.	PHP 7.0.0
iterable	Le paramètre doit être soit un array ou une instanceof Traversable.	PHP 7.1.0
object	Le paramètre doit être un object.	PHP 7.2.0

Mode de passage des arguments (types natifs)

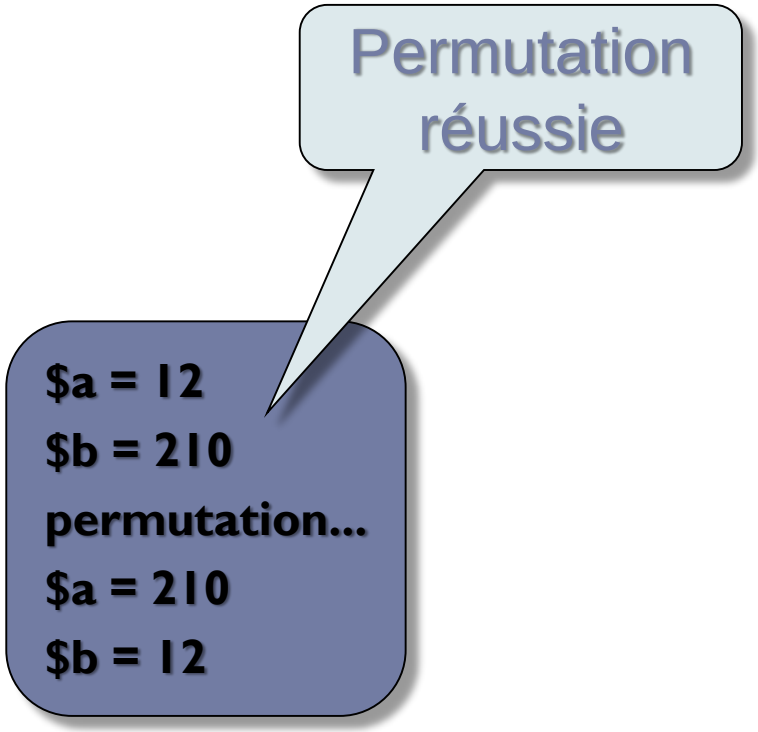
```php
<?php
function permutation($x, $y) {
   echo "permutation..." ;
   $t = $x ;
   $x = $y ;
   $y = $t ;
}
$a = 12 ;
$b = 210 ;
echo "\$a = $a" ;
echo "\$b = $b" ;
permutation($a, $b) ;
echo "\$a = $a" ;
echo "\$b = $b" ;
```

Permutation impossible :
Passage des arguments des fonctions par valeur

```
$a = 12
$b = 210
permutation...
$a = 12
$b = 210
```

Mode de passage des arguments (types natifs)

```php
<?php
function permutation(&$x, &$y) {
   echo "permutation..." ;
   $t = $x ;
   $x = $y ;
   $y = $t ;
}
$a = 12 ;
$b = 210 ;
echo "\$a = $a" ;
echo "\$b = $b" ;
permutation($a, $b) ;
echo "\$a = $a" ;
echo "\$b = $b" ;
```



Arguments par défaut des fonctions

- Valeur par défaut d'un argument s'il n'a pas été défini lors de l'appel de la fonction

```
function bonjour($nom="inconnu")
{
  echo "Bonjour cher $nom" ;
}
```

- Utilisation

```
bonjour() ;
```

Bonjour cher inconnu

```
bonjour("Ali") ;
```

Bonjour cher Ali

Définition de fonctions fréquemment utilisées

- Certaines fonctions sont utilisées dans plusieurs scripts PHP
- Comment faire pour ne pas les définir dans chacune des pages ?
- Utilisation de :
  - `include("fichier") ;`
  - `require("fichier") ;`
  - `include_once("fichier") ;`
  - `require_once("fichier") ;`

Permet d'inclure le contenu de ***fichier*** dans le script courant

require erreur bloquante, **include** erreur non bloquante

…_once pour les inclusions uniques

include et require

Fichier mafonction.php

```
<?
function mafonction($arg)
{
  …
}
```

Problème avec include :
- produit un **warning**
- le script continue

Problème avec require :
- produit un **fatal error**
- le script s'arrête

Fichier utilisation1.php

```
…
require("mafonction.php")
mafonction(true) ;
…
```

Fichier utilisation2.php

```
…
include("mafonction.php")
…
$var=false ;
mafonction($var) ;
…
```

Fichier utilisation3.php

```
…
require("mafonction.php")
…
```

Gestion des erreurs

- Dans certains cas, il n'est ni possible ni utile de poursuivre l'exécution du code PHP (variables non définies, valeurs erronées, échec de connexion, …)

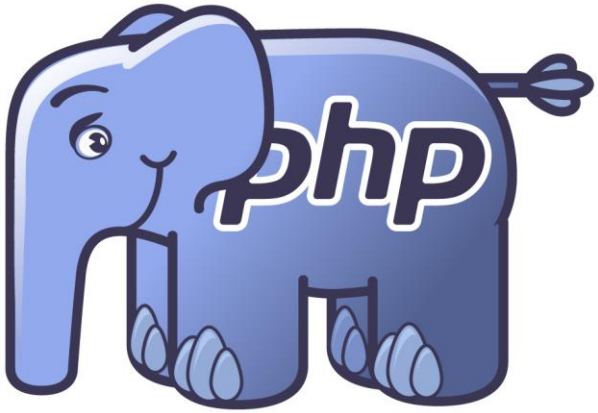
- Arrêt brutal de l'exécution du code:
  - `die(message)`
  - `exit(message)`

  Envoie *message* au navigateur et termine l'exécution du script courant

Exercice 2

Écrivez une fonction qui retourne la somme de la série de terme général *un = x^{2n} + 1/n!. Les paramètres de la fonction sont n pour le* nombre d'itérations et *d pour le nombre de décimales affichées* pour le résultat.

Rappels du POO



Développement objet

- Définition de briques logicielles représentant un concept, une idée ou une entité ainsi que leurs interactions : les objets
- Un objet est une structure de données comprenant également les fonctionnalités de traitement des données
- L'objet est vu au travers de ses spécifications
- Les concepts associés sont :
  - Encapsulation
  - Héritage
  - Polymorphisme

Classe

- Une classe définit un modèle, un moule, à partir duquel tous les objets de la classe seront créés
- La classe décrit les données internes ainsi que les fonctionnalités des objets
- La classe est une vision « inerte », une recette de cuisine, visant à décrire la structure et le comportement des objets qui seront créés
- La construction d'un objet à partir de la classe génératrice s'appelle instanciation
- Les objets, entités « vivantes » en mémoire, sont des instances de la classe

Instanciation

- La classe est une description « inerte »

- Les objets doivent être instanciés à partir de la classe génératrice pour exister et devenir fonctionnels

- Exemple : la classe Animal
  $bambi = new Animal() ;
  $scrat = new Animal() ;
  $melman = new Animal() ;

Encapsulation

- Procédé consistant à rassembler les données et les traitements au sein des objets
- L'implémentation interne des objets est cachée
- Les objets sont vus à travers leurs spécifications
- Les données internes et les fonctionnalités possèdent un niveau de visibilité et peuvent éventuellement être masquées :
  - Public
  - Privé
  - Protégé

Encapsulation

- Les données internes des objets sont appelées attributs (ou propriétés voire champs)
- Les fonctionnalités des objets sont appelées méthodes
- Méthodes habituelles :
  - Constructeur / destructeur
  - Accesseurs / modificateurs (getters / setters)
- Référence à l'objet courant dans la description de la classe : $this

Visibilité

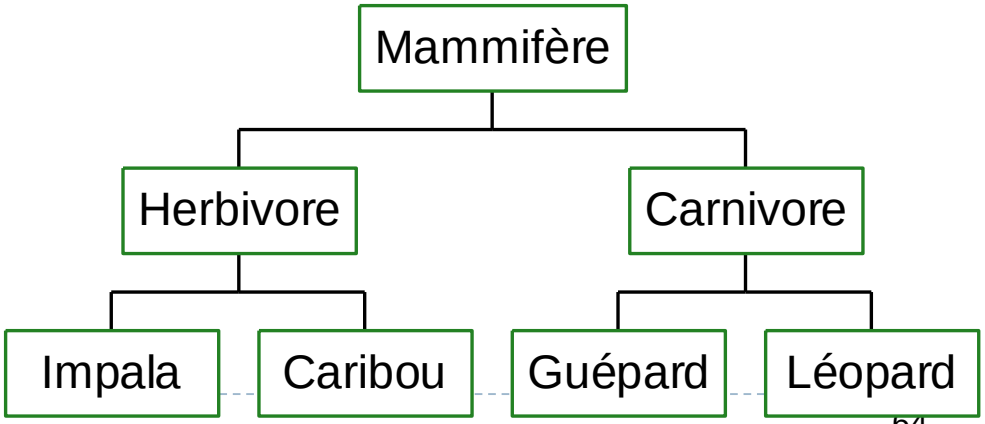
- Publique :
  Les données internes et les méthodes sont accessibles par tous

- Protégé :
  Les données internes et les méthodes sont accessibles aux objets dérivés

- Privé :
  Les données internes et les méthodes ne sont accessibles qu'aux objets de la classe

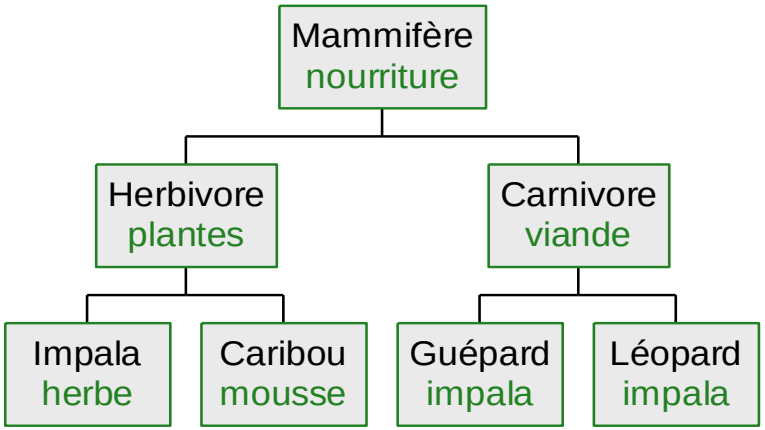
Héritage ou dérivation ou extension

- Création de nouvelles classes à partir du modèle d'une classe existante
- La nouvelle classe possède tous les attributs et méthodes de la classe mère
- La nouvelle classe peut proposer de nouveaux attributs et de nouvelles méthodes ou spécialiser des méthodes de la classe mère



Polymorphisme

- Choix dynamique de la méthode qui correspond au type réel de l'objet
- Méthode **mange()**

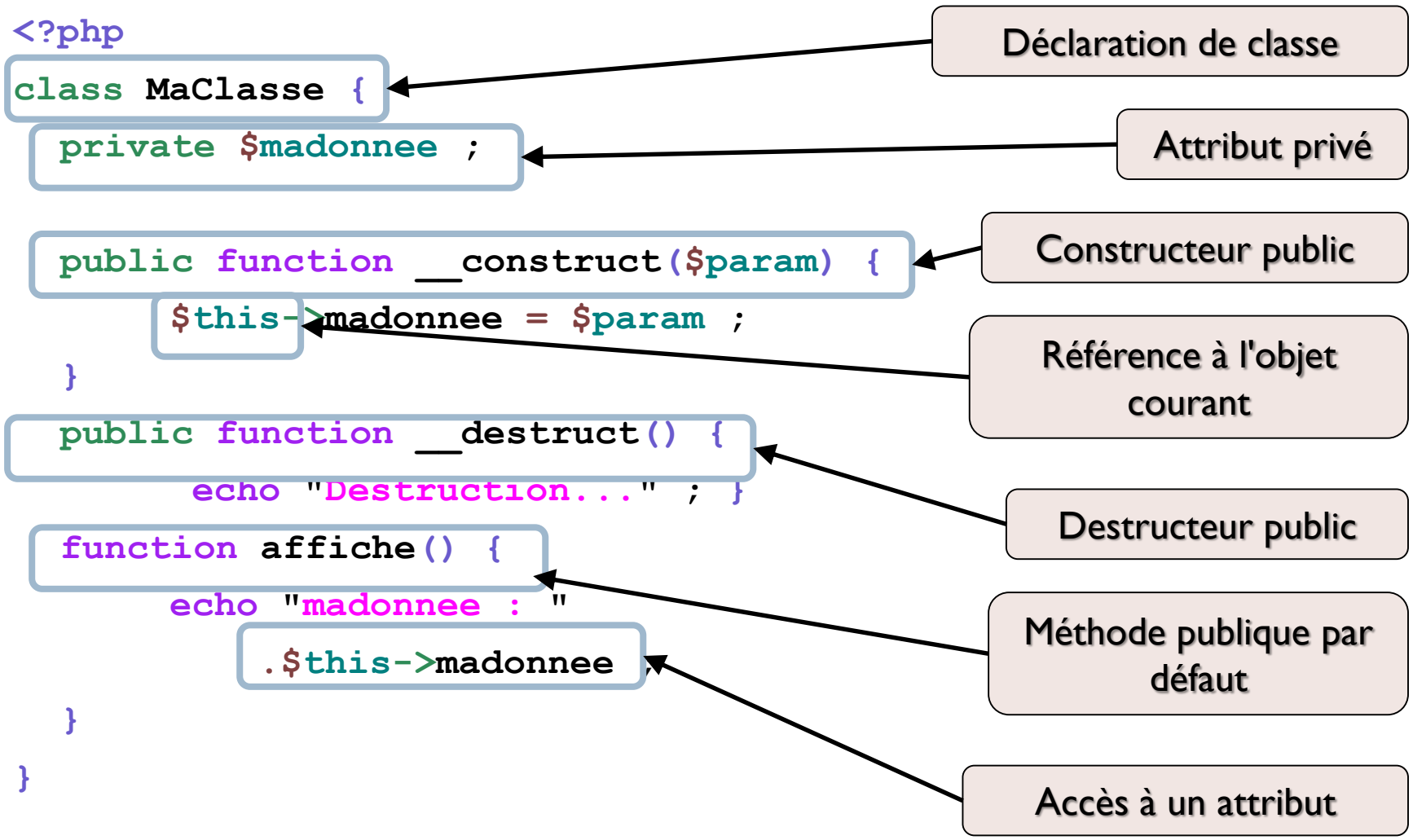



```
function nourriture(Mammifere $m) {
        return $m->mange() ;
   }
$i = new Impala() ; nourriture($i) ;
$c = new Carnivore() ; nourriture($c) ;
```

herbe

viande

Définition d'une classe



Utilisation d'une classe

```php
<?php
require "maclasse.class.php" ;

// Nouvel objet
$o = new MaClasse(12) ;
// Utilisation d'une méthode
maclasse contient 12
```

Inclusion de la définition de la classe

Création d'un objet

Méthode affiche de l'objet $o

```php
class MaClasse {
      private $madonnee ;
...
```

L'attribut est privé

```php
    function __destruct() {
   echo "Destruction..." ; }
```

Destruction de l'objet $o

Valeur par défaut des attributs

```php
<?php
class MaClasse {
  private $madonnee = "Défaut" ;
  public function affecte($val) {
        $this->madonnee = $val ; }
  public function affiche() {
        echo "madonnee : ".$this->madonnee ; }
}
$o = new MaClasse()
madonnee : Défaut
$o->affecte("Nouvelle")
madonnee : Nouvelle
$o->affiche();
```

Attribut avec valeur par défaut

Nouvel objet

Affichage

Affectation

Affichage

Attributs et méthodes de classe

- Mot clé `static`
- Attributs et méthodes utilisables sans instance de la classe (=attributs et méthode de classe)
- Attributs NE peuvent PAS être accédés depuis une instance (~~`$objet->attribut`~~)
- Attributs partagés par toutes les instances de la classe
- Méthodes peuvent être accédés depuis une instance (`$objet->methode()` mais c'est mal !)
- Dans les méthodes, `$this` n'est pas disponible

Attributs statiques

```
class MaClasse {
  private static $n = 0 ;
  public function __construct() {
        echo ++MaClasse::$n
           ." instance(s)" ; }
  public function __destruct() {
        echo "destruction" ; self::$n-- ; }
}
```

Attribut privé statique : ne peut être accédé que par des méthodes de la classe

Accès à l'attribut statique

```
1 instance(s)
2 instance(s)
destruction
2 instance(s)
3 instance(s)
```

Fatal error: Cannot access private property MaClasse::$n in **dummy.php** on line **37**

Méthodes statiques

```
class MaClasse {
   private static $n = 0 ;
   public function __construct() {
         echo ++MaClasse::$n." instance(s)\n" ; }
   public function __destruct() {
         MaClasse::$n-- ; }
   public static function f($i) {
         echo "Dans f() : ".($i*$i) ; }
}
```

Méthode publique statique

```
1 instance(s)
Dans f() : 4
Dans f() : 9
```

Appel à partir d'une instance
Toléré

Appel sans instance

Constantes de classe

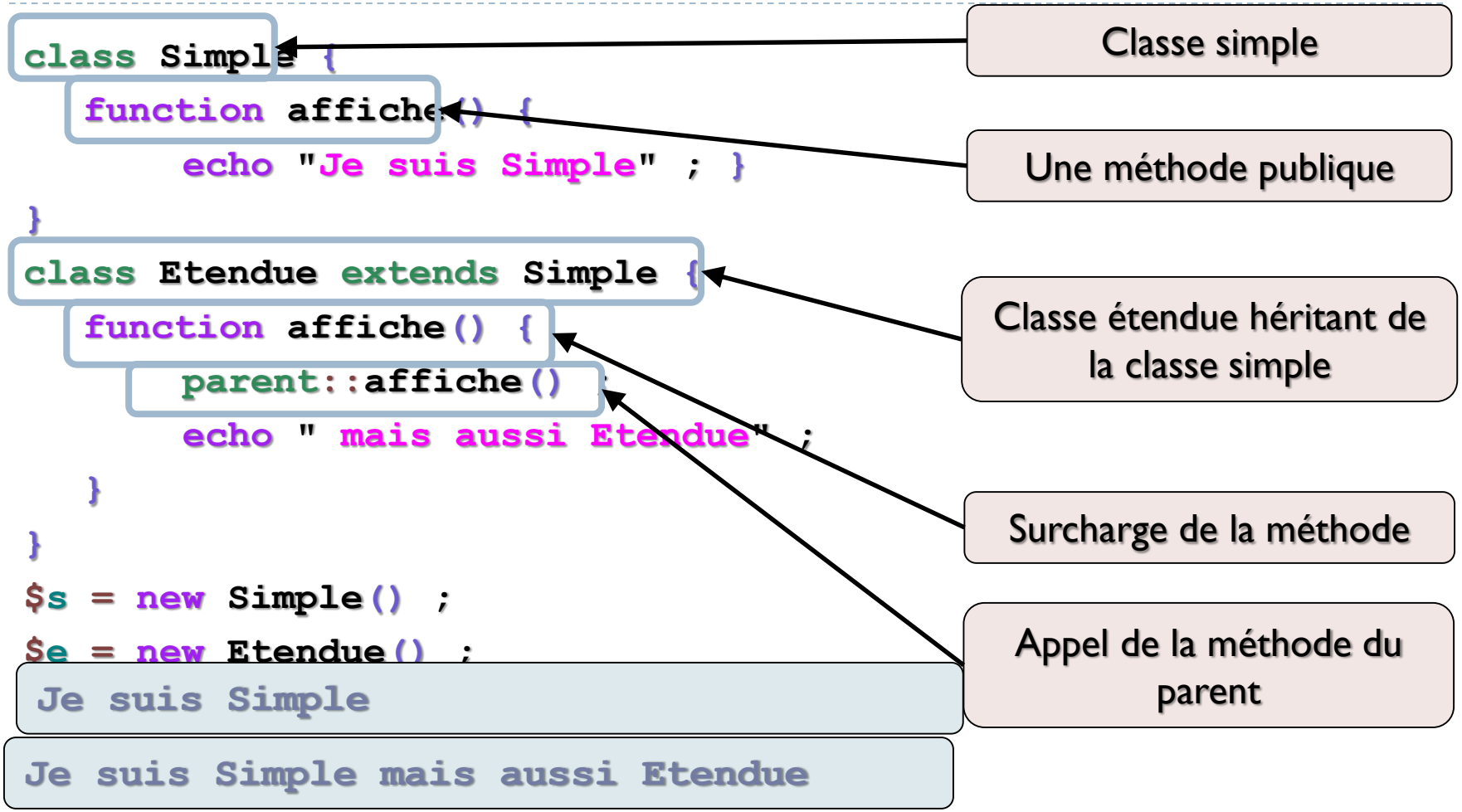
```
class MaClasse {
   const constante = "Valeur" ;
   public function montre() {
         echo self::constante ;
   }
}

$c = new MaClasse() ;
$c->montre() ;
echo MaClasse::constante ;
```

Constante publique de classe

Accès à la constante de classe depuis la classe

Accès à la constante de classe à l'extérieur de la classe

Héritage

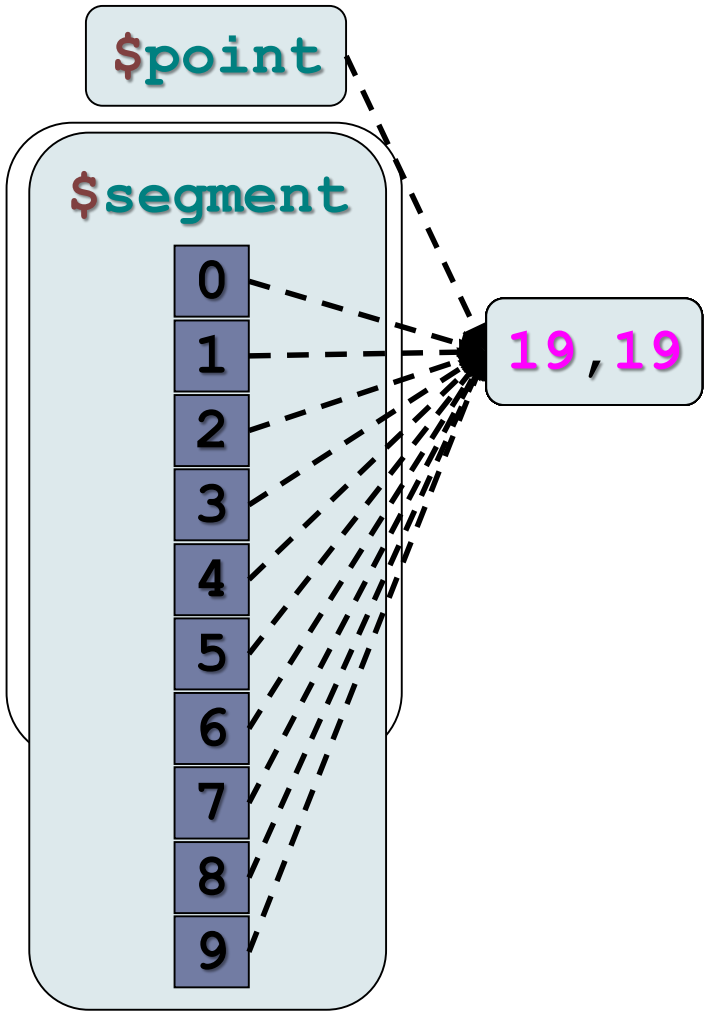

class Simple {
function affiche() {
echo "Je suis Simple" ; }
}
class Etendue extends Simple {
function affiche() {
parent::affiche()
echo " mais aussi Etendue" ;
}
}
$s = new Simple() ;
$e = new Etendue() ;
Je suis Simple
Je suis Simple mais aussi Etendue
Classe simple
Une méthode publique
Classe étendue héritant de la classe simple
Surcharge de la méthode
Appel de la méthode du parent

Assignation d'objets

```
class Point {
  private $_x ;
  private $_y ;

  public function __construct($x=0, $y=0) {
       $this->_x = $x ;
       $this->_y = $y ; }

  public function set($x, $y) {
       $this->_x = $x ;
       $this->_y = $y ; }

  public function toString() {
       return "({$this->_x}, {$this->_y})" ; }
}
```

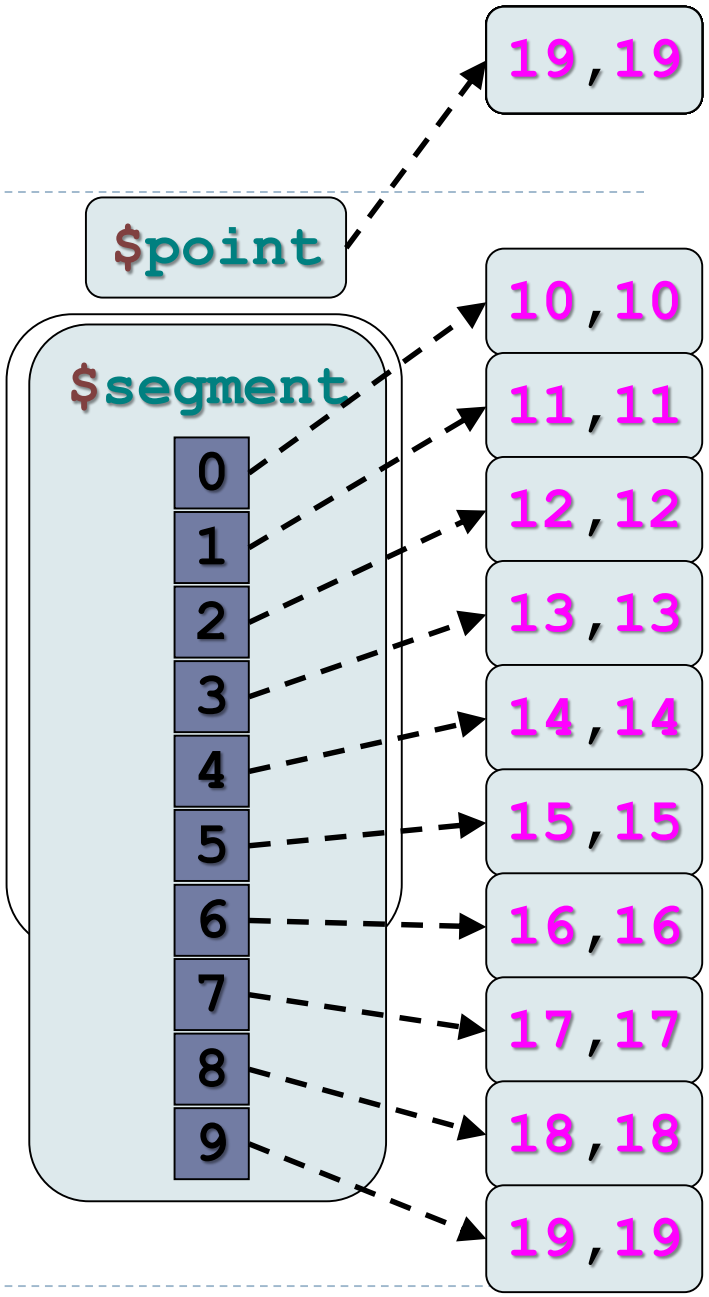
Assignation d'objets

```php
$segment = array() ;
$point = new Point() ;
for ($i=10; $i<20; $i++)
{
  $point->set($i, $i) ;
  $segment[] = $point ;
}

foreach ($segment as $k => $p)
  echo "$k: {$p->toString()}\n" ;
```



Clonage d'objets

```php
$segment = array() ;
$point = new Point() ;
for ($i=10; $i<20; $i++)
{
   $point->set($i, $i) ;
   $segment[] = clone $point ;
}

foreach ($segment as $k => $p)
   echo "$k: {$p->toString()}\n" ;
```



Objets comme arguments de fonctions

```
function origine($p) {
  $p->set(0, 0) ; }

$point = new Point(10, 10) ;

echo "avant: {$point->toString()}\n" ;
origine($point) ;
echo "apres: {$point->toString()}\n" ;
```

```
avant: (10, 10)
apres: (0, 0)
```

Passage de l'objet **Point** par référence

Classe Abstraite

- Pour créer une classe abstraite, faites précéder le mot-clé class du mot-clé abstract, comme ceci :

```
abstract class nomclasse
{
//Définition de la classe
}
```

- Pour créer une méthode abstraite, faites également précéder le modificateur d'accès du mot-clé abstract, selon le modèle suivant :

```
abstract public function nomfonction() ;
```

- Dans la classe qui dérive d'une classe abstraite, vous devez définir les modificateurs d'accessibilité des méthodes avec une visibilité égale ou plus large que celle de la méthode abstraite. Une classe abstraite définie, par exemple, protected, est implémentée dans les classes dérivées comme protected ou public.

Interface

- La structure d'une interface doit respecter la forme suivante :

```
interface nom_interface
{
public function fonction1($var) ;
public function fonction2($var) ;
}
```

- Pour implémenter une interface dans une classe, il faut faire suivre le nom de la classe du mot-clé **implements** à la place de **extends** puis des noms des interfaces séparés par des virgules.

Classe Final

- Pour interdire la redéfinition d'une méthode d'une classe parente dans ses classes dérivées, faits précéder le mot-clé function du mot-clé final.

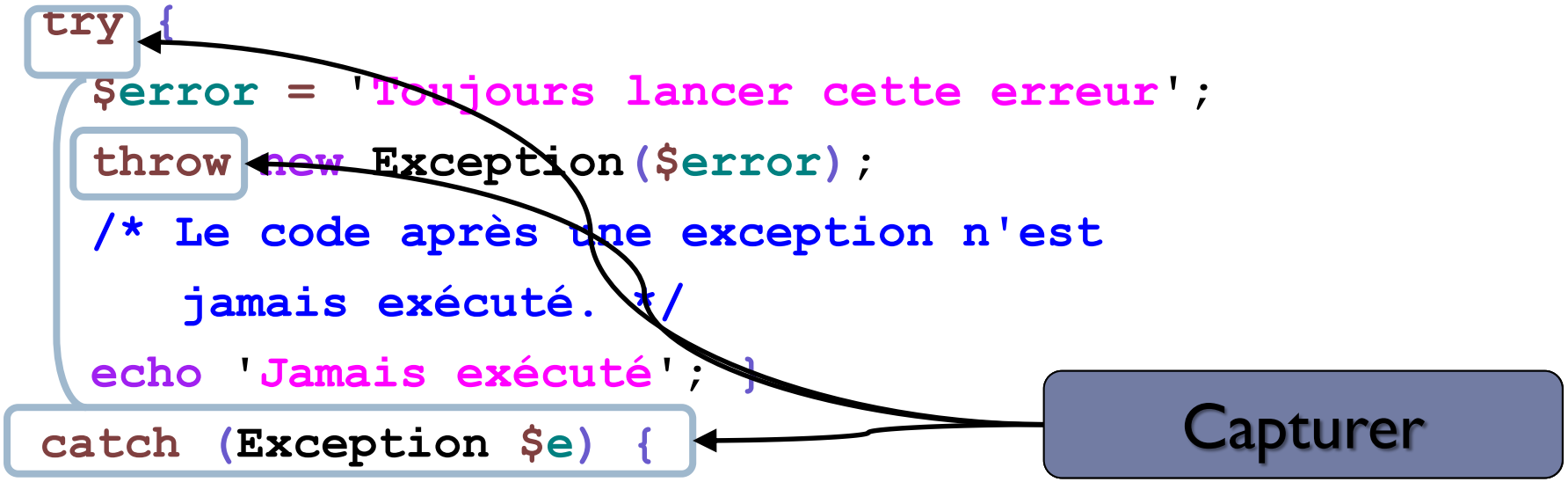
```
class triangle
{
private $x ;
private $y ;
private $z ;
function __construct($x,$y,$z)
{
$this->x=$x ;
$this->y=$y ;
$this->z=$z ;
}
final function trianglerect()
{
if(($this->x*$this->x + $this->y*$this->y) ==($this->z*$this->z))
return "Le triangle est rectangle";
else echo "Triangle non rectangle";
}
}
```

Gestion des erreurs : exceptions

- Gestion des exception identiques à C++/Java
- Exception peut être :
  - Jetée : `throw`
  - Essayée : `try`
  - Capturée : `catch`
- Exception jetée : code après `throw` non exécuté
- Capture : 1 ou plusieurs blocs (selon type)
- Exception non capturée : erreur fatale

Utilisation des exceptions

```
try {
  $error = 'Toujours lancer cette erreur';
  throw new Exception($error);
  /* Le code après une exception n'est
      jamais exécuté. */
  echo 'Jamais exécuté'; }
catch (Exception $e) {
  echo  "Capture Exception: "
        .$e->getMessage()." \n"; }
// Poursuite de l'exécution
echo 'Bonjour le monde';
```



Classe **Exception** PHP 5

```php
<?php
class Exception {
  protected $message = ''; // message de l'exception
  protected $code = 0;     // code de l'exception
  protected $file;         // fichier source de l'exception
  protected $line;         // ligne de la source de l'exception

      function __construct(string $message=NULL, int code=0);

final function getMessage();        // message de l'exception
final function getCode();           // code de l'exception
final function getFile();           // nom du fichier source
final function getLine();           // ligne du fichier source
final function getTrace();          // un tableau de
  backtrace()
final function getTraceAsString(); // chaîne de trace
      function __toString();         // chaîne pour l'affichage
}
```

Exercice 3

A. Créez une classe nommée **form** représentant un formulaire HTML.

1. Le constructeur doit créer le code d'en-tête du formulaire en utilisant les éléments <form> et <fieldset>.
2. Une méthode **settext**() doit permettre d'ajouter une zone de texte.
3. Une méthode **setsubmit**() doit permettre d'ajouter un bouton d'envoi. Les paramètres de ces méthodes doivent correspondre aux attributs des éléments HTML correspondants.
4. La méthode **getform**() doit retourner tout le code HTML de création du formulaire.
5. Créez des objets form, et ajoutez-y deux zones de texte et un bouton d'envoi. Testez l'affichage obtenu.

Le fichier contenant la définition de la classe form (form.php) est indépendant ce qui permet son inclusion dans d'autres scripts en vue de l'utilisation de la classe ou de son extension.

Exercice 3 Suite

B. Créez une sous-classe nommée form2 en dérivant la classe **form**. Cette nouvelle classe doit permettre de créer des formulaires ayant en plus des boutons radio et des cases à cocher. Elle doit donc avoir les méthodes supplémentaires qui correspondent à ces créations. Créez des objets, et testez le résultat.

C. Créez un objet à partir de la classe form2 , puis créez-en un clone. Modifiez certaines caractéristiques de cet objet, et affichez les deux formulaires obtenus.

REFERENCES

- PHP7 cours et exercice, Jean Engels, Groupe Eyrolles, 2017, ISBN : 978-2-212-67360-9.

- Cours PHP, Jérôme CUTRONA, Université de Reims Champagne Ardenne.