

# Développement Web coté serveur

Introduction au langage PHP

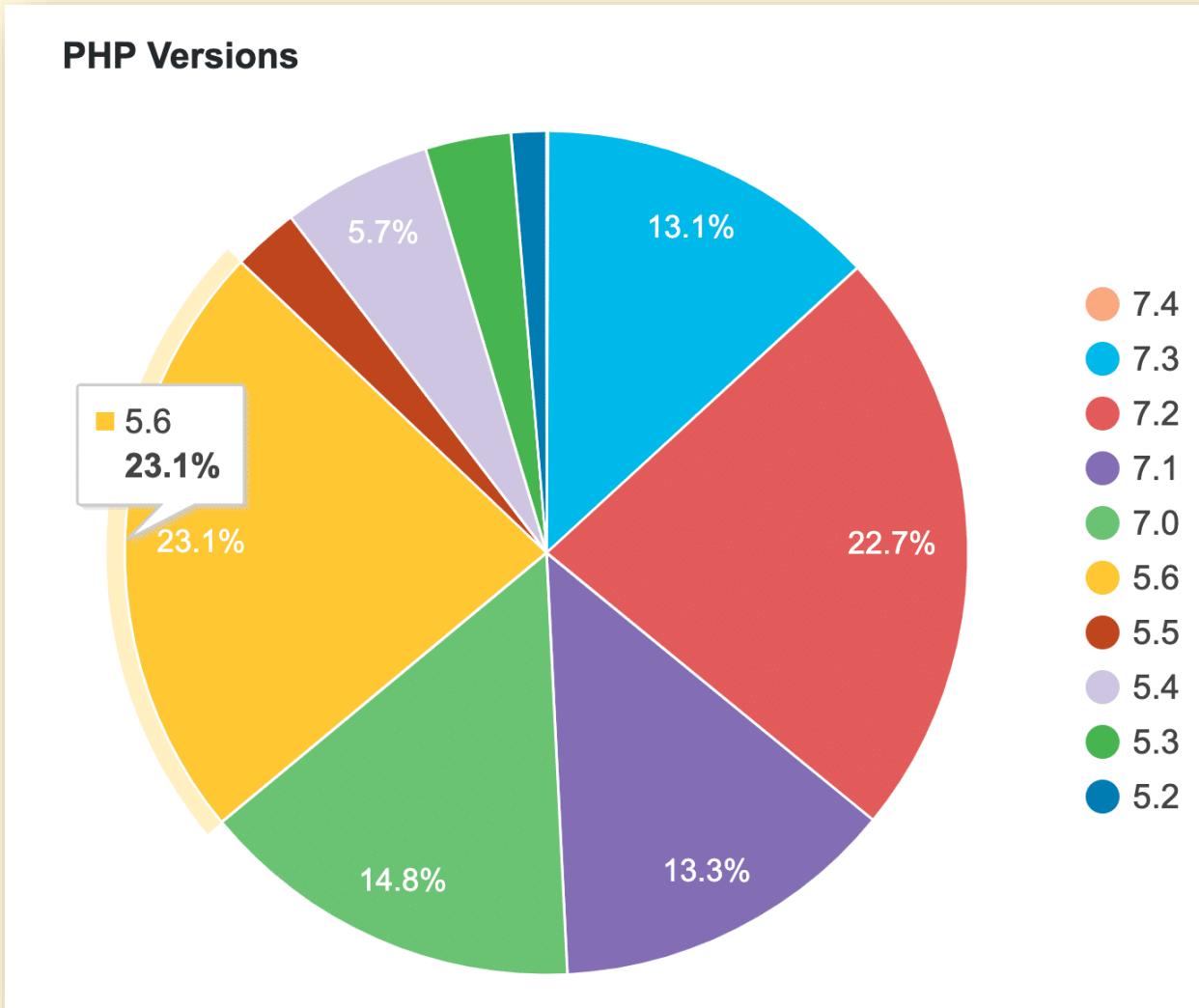
2020

# PHP: Langage de script pour le Web

---

- Qu'est-ce que PHP ?
  - Langage de **script**. Utilisé **coté serveur**
  - Acronyme récursif : **PHP: Hypertext Preprocessor**
  - Crée en 1994-1995 par Rasmus Lerdorf
  - Extension utilisée sur certains serveurs Web (78.9%)
  - Langage multi plate-forme (UNIX / Windows...)
  - Open Source
  - Versions actuelles (*source [w3techs.com](https://w3techs.com)*) :
    - PHP4 (0,3% en sept 2020)
    - PHP5 (43,6% en sept 2020)
    - PHP7 (56.1% en sept 2020)
    - PHP8 (en cours des tests)

# PHP: Langage de script pour le Web



% Utilisation des versions PHP en 2020

# Utilité et utilisation de PHP

---

- Crédit : Utilité et utilisation de PHP

# Principales fonctionnalités de PHP

---

- Manipulation de chaînes et tableaux
- Calendrier / dates / heures
- Fonctions mathématiques
- Accès au système de fichiers (create, open, read, write, delete)
- Manipulation d'images
- HTTP / FTP / IMAP
- Bases de données (Oracle, MySQL, ...)
- XML, JSON
- ...

# Fonctionnement de PHP

Rendu graphique des données le fichier  
= réponse HTTP

Réseau

GET /hello.php HTTP/1.0

Client

récepteur

- HTML
- JavaScript
- CSS

Protocole HTTP

Exécution d'un programme sur  
Construction de la réponse

Server

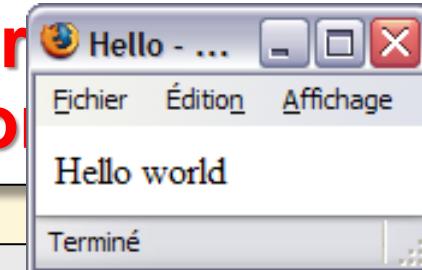
Server

```
<html>
<head>
<title>Hello</title>
</head>
<body>Hello world</body>
</html>
```

Module PHP

MySQL

```
<?php
echo <<<HTML
<html>
<head>
<title>Hello</title>
</head>
<body>Hello world</body>
</html>
HTML;
```



hello.php

# Fonctionnement de PHP

---

Client ↔ Serveur

1. Connexion TCP sur le serveur (port 80)
2. Requête HTTP du client (*mon\_fichier.php*)
3. Localisation de la ressource
4. Exécution du code PHP
5. Envoi du résultat de l'exécution au client  
= réponse HTTP
6. Fermeture de la connexion
7. Rendu graphique des données (HTML, image, ...)

# Installation de PHP

---

Pour installer PHP, on peut utiliser les outils :



WampServer



# Programme en PHP

Un script PHP est exécuté sur le serveur et le résultat HTML brut est renvoyé au navigateur.

L'extension de fichier par défaut pour les fichiers PHP est **.php**

Un fichier PHP contient normalement des balises HTML et du code de script PHP.

Un script PHP commence par **<?php** et se termine par **?>**

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

My first PHP page

Hello World!

# Eléments de syntaxe PHP

---

- La syntaxe de PHP ressemble à celle de famille "C" (C, C++, Java, ...)
- Chaque instruction se termine par ;
- les noms de variables sont sensibles à la casse!
- Commentaires:

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

# Les variables et les types de données

---

- Tout identificateur commence par "\$"
- Les affectations sont réalisées grâce à "="
  - Numérique entier: **12** ou réel: **1.54**
  - Chaîne: "Hello" ou 'Bonjour'
  - Booléen: **true**, **false**
  - Tableau: **\$tab[2]=12**
  - Objet (PHP4, PHP5)
  - Ressource
  - **NULL**
- Le **type** d'une variable est **dynamique** et est déterminé par la valeur qui lui est affectée

# Typage faible. Exemple

---

// Pas de déclaration de variable

\$test = 1.5 ; // Réel

\$test = 12 ; // Entier

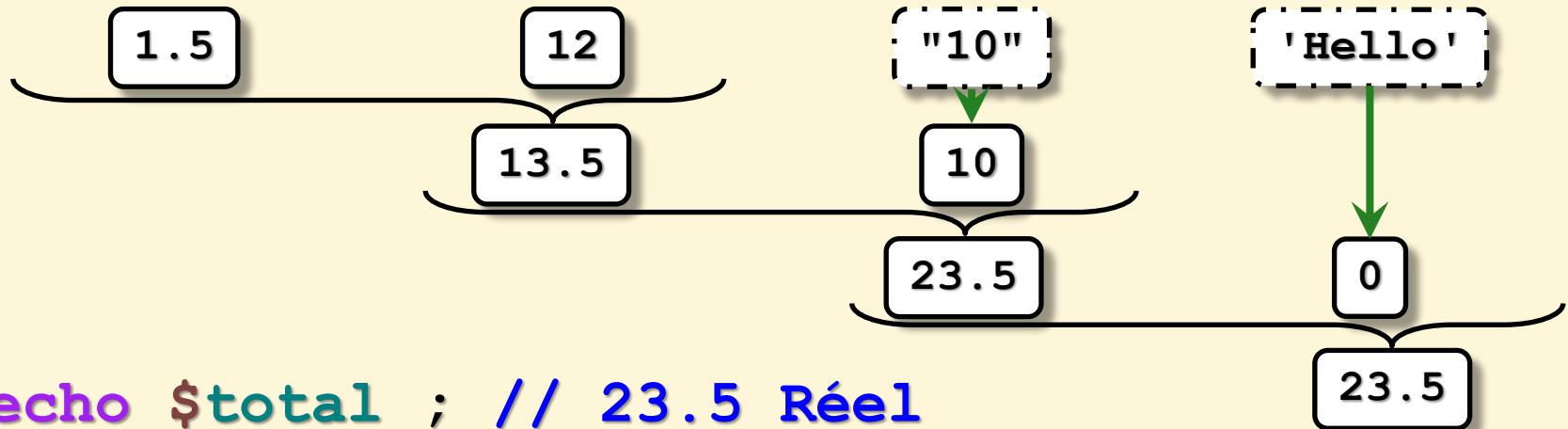
\$test = array() ; // Tableau

\$test = "10" ; // Chaîne

echo \$test ; // 10

# Typage automatique. Exemple

```
$nombre1 = 1.5 ;           // Réel  
$nombre2 = 12 ;            // Entier  
$chaine1 = "10" ;          // Chaîne  
$chaine2 = 'Hello' ;        // Chaîne  
  
$total =  
    $nombre1 + $nombre2 + $chaine1 + $chaine2 ;
```



```
echo $total ; // 23.5 Réel
```

# Les chaînes de caractères

## Substitution de variables dans les chaînes

### ■ Guillemets simples

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

chaîne

voici une \$a

### ■ Guillemets doubles

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

chaîne

voici une chaîne

### ■ Syntaxe *HereDoc*

- `$a="chaîne" ;`
- `$b=<<<MARQUE_DE_FIN`  
`voici une $a`  
`sur deux lignes ;-)`  
`MARQUE_DE_FIN;`

chaîne

voici une chaîne  
sur deux lignes ;-)

# Concaténation de chaînes

---

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

"**Bonjour** " . "**Mohamed**"

→ vaut "**Bonjour Mohamed**"

**\$nb** = 6\*2 ;

"**Acheter** " . **\$nb** . " **jeux**"

→ vaut "**Acheter 12 jeux**"

# La commande echo

---

- Permet d'envoyer du texte au navigateur du client (« écrire » la page au format HTML résultant de l'interprétation de PHP)
  - `echo "Bonjour" ;`
  - `$nom="Mohamed" ; echo "Bonjour $nom" ;`
- Plus généralement, permet d'envoyer des octets au navigateur du client
  - Ficher HTML, XML, CSS, JavaScript, ...
  - Données d'une image
  - Contenu d'un fichier PDF, Flash, etc.

# Hello world !

Interprétation du code PHP sur le serveur  
et transmission du résultat au client

```
<?ph  
$debut = <<<HTML  
  
<html>  
  <head>  
    <title>hello</title>  
  </head>  
  <body>\n  
HTML;  
$corps = "Hello world!\n";  
$fin   = <<<HTML  
  </body>  
</html>  
HTML;  
/* Envoi au client */  
echo $debut.$corps.$fin ;
```

```
<html>  
  <head>  
    <title>hello</title>  
  </head>  
  <body>  
Hello world!  
  </body>  
</html>
```

Impossible de voir le code PHP depuis le navigateur !!

# 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 d'incrémentation

---

<b>\$a++</b>	Retourne la valeur de <b>\$a</b> puis augmente la valeur de <b>\$a</b> de 1
<b>+\$a</b>	Augmente la valeur de <b>\$a</b> de 1 puis retourne la nouvelle valeur de <b>\$a</b>
<b>\$a--</b>	Retourne la valeur de <b>\$a</b> puis diminue la valeur de <b>\$a</b> de 1
<b>--\$a</b>	Diminue la valeur de <b>\$a</b> de 1 puis retourne la nouvelle valeur de <b>\$a</b>

# Les opérateurs de comparaison

---

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

# Les opérateurs logiques

---

[Expr1] <b>and</b> [Expr2]	Vrai si [Expr1] et [Expr2] sont vraies
[Expr1] <b>&amp;&amp;</b> [Expr2]	idem
[Expr1] <b>or</b> [Expr2]	Vrai si [Expr1] ou [Expr2] sont vraies
[Expr1] <b>  </b> [Expr2]	idem
[Expr1] <b>xor</b> [Expr2]	Vrai si [Expr1] ou [Expr2] sont vraies mais pas les deux
! [Expr1]	Vrai si [Expr1] est non vraie

# Les opérateurs sur bits

---

<code>\$a &amp; \$b</code>	<b>ET binaire</b>
<code>\$a   \$b</code>	<b>OU binaire</b>
<code>\$a ^ \$b</code>	<b>XOR binaire</b>
<code>~ \$a</code>	<b>Inversion bit à bit</b>
<code>\$a &lt;&lt; \$b</code>	<b>\$a décalé à gauche de \$b rangs</b>
<code>\$a &gt;&gt; \$b</code>	<b>\$a décalé à droite de \$b rangs</b>

# Précédence des opérateurs

Opérateurs
<b>new</b>
<b>[</b>
<b>++ --</b>
<b>! ~ - (int) (float) (string) (array) (object) @</b>
<b>* / %</b>
<b>+ - .</b>
<b>&lt;&lt; &gt;&gt;</b>
<b>&lt; &lt;= &gt; &gt;=</b>
<b>== != === !==</b>
<b>&amp;</b>

# Précédence des opérateurs

Opérateurs
<code>^</code>
<code> </code>
<code>&amp;&amp;</code>
<code>  </code>
<code>? :</code>
<code>= += -= *= /= *= ^=</code>
<code>&lt;&lt;= &gt;&gt;=</code>
<code>and</code>
<code>xor</code>
<code>or</code>

En cas de doute,  
utilisez les parenthèses ;-)

# Structure de contrôle Si...Alors...Sinon...

---

```
if (condition)
{
    /* Bloc d'instructions exécuté si la
    condition est vraie */

}

[else
{
    /* Bloc d'instructions exécuté si la
    condition est fausse */

}]
```

# Structure de contrôle Tant que... faire...

---

**while** (**condition**)

```
{  
    /* Bloc d'instructions répété tant que la  
    condition est vraie */  
}
```

---

**do** {

```
    /* Bloc d'instructions exécuté une fois  
    puis répété tant que la condition est  
    vraie */
```

```
} while (condition) ;
```

# Structure de contrôle Tant que... faire...

```
for(avant; condition; fin_chaque_itération)
{ /* Bloc d'instructions répété tant que la
condition est vraie */
}
```

Équivalent à:

```
avant ;
while (condition)
{ /* Bloc d'instructions répété tant que la
condition est vraie */
    fin_chaque_itération ;
}
```

# Structure de contrôle switch...

---

```
switch (val)
{
    case v1:
        instructions exécutées si val==v1
    case v2:
        instructions exécutées si val==v2
                    ou si val==v1
    ...
    default:
        instructions dans tous les cas
}
```

# L'instruction break

---

Permet de sortir d'une structure de contrôle

```
switch (val)
{
    case v1:
        instructions exécutées si val==v1
        break ; /* On sort du switch si val==v1 */
    case v2:
        instructions exécutées si val==v2
        ou si val==v1
        break ; /* On sort du switch si val==v2 */
    ...
    default:
        instructions exécutées dans tous les cas
        si val!=v1 et val!=v2
}
```

# Les tableaux

- Crédit / initialisation:

```
$tab1=array(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

# Les tableaux « à trous »

- Les éléments du tableaux ne sont pas forcement d'indices consécutifs :

```
$tab4[0] = 12 ;  
$tab4[1] = "fraise" ;  
$tab4[2] = 2.5 ;  
$tab4[5] = "e15" ;
```

Clé	Valeur
0	12
1	"fraise"
2	2.5
3	
4	
5	"e15"

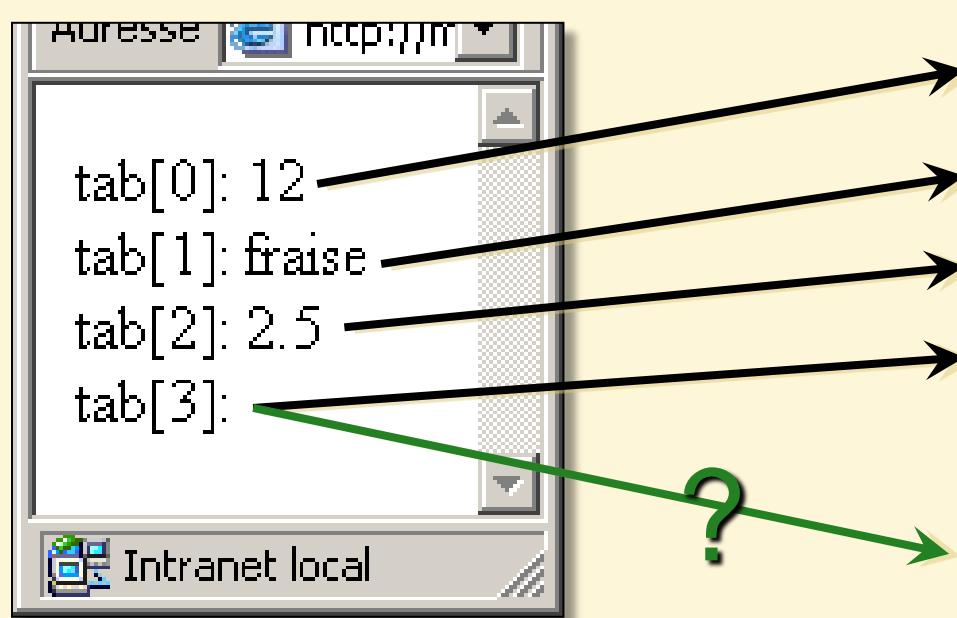
- Comment parcourir de tels tableaux ?

# Les tableaux « à trous » (suite)

Parcours classique :

```
for ($i=0; $i < sizeof($tab4); $i++)
{ echo "tab[$i]: "
    . $tab4[$i] . "<BR>\n";
}
```

4



Clé	Valeur
0	12
1	"fraise"
2	2.5
3	
4	
5	"el5"

# Structure de contrôle Pour chaque...

---

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

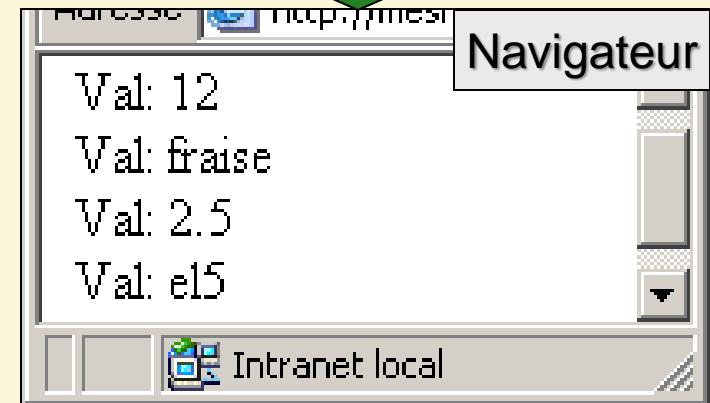
# Parcours de tableau : foreach

PHP

```
...
$tab4[0] = 12 ;
$tab4[1] = "fraise" ;
$tab4[2] = 2.5 ;
$tab4[5] = "e15" ;
foreach($tab4 as $v)
{
    echo "Val: $v<br>\n";
}
...
```

HTML

```
...
Val:12<br>
Val:fraise<br>
Val:2.5<br>
Val:e15<br>
...
```



# Tableaux associatifs

---

- Tableaux dont l'accès aux éléments n'est plus réalisé grâce à un index (0,1,...) mais grâce à une clé de type entier ou chaîne.
- Exemples de clés:

`$tab['un'] = 12 ;`

`$tab[205] = "bonjour" ;`

`$tab["la valeur"] = 3.0 ;`

- Création

```
$tab = array(cle1 => val1,  
             cle2 => val2,  
             ...);
```

# Tableaux associatifs - Exemples

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

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

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

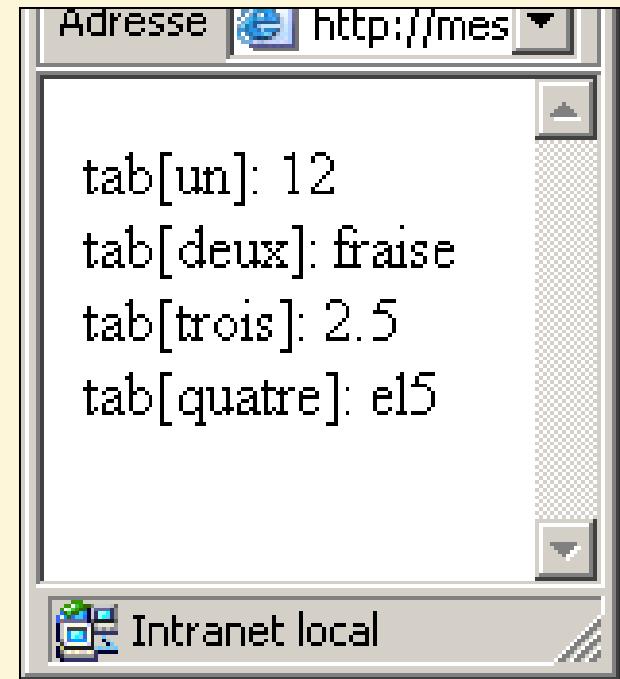
# Structure de contrôle Pour chaque...

---

```
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

```
<?php
$html = <<<HTML
<html>
  <head><title>foreach clé</title>
  </head>
<body>
HTML;
$tab6 = array('un'      => 12,
              'deux'    => "fraise",
              "trois"   => 2.5,
              "quatre"  => "el15") ;
foreach ($tab6 as $cle => $val)
{
  $html .= "tab[$cle]: $val<br>\n" ;
}
echo $html . "</body>\n</html>" ;
?>
```



# Exemple de génération de code HTML

```
<?php
$html = <<<HTML
<!DOCTYPE html">
<html>
<head>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
    <title>Boucle</title>
</head>
<body>
HTML;
for ($i=1; $i<20; $i++) {
    $html .= "Le serveur compte... "
        . $i . "<br>\n" ;
}
$html .= <<<HTML
</body>
</html>
HTML;
echo $html ;
```

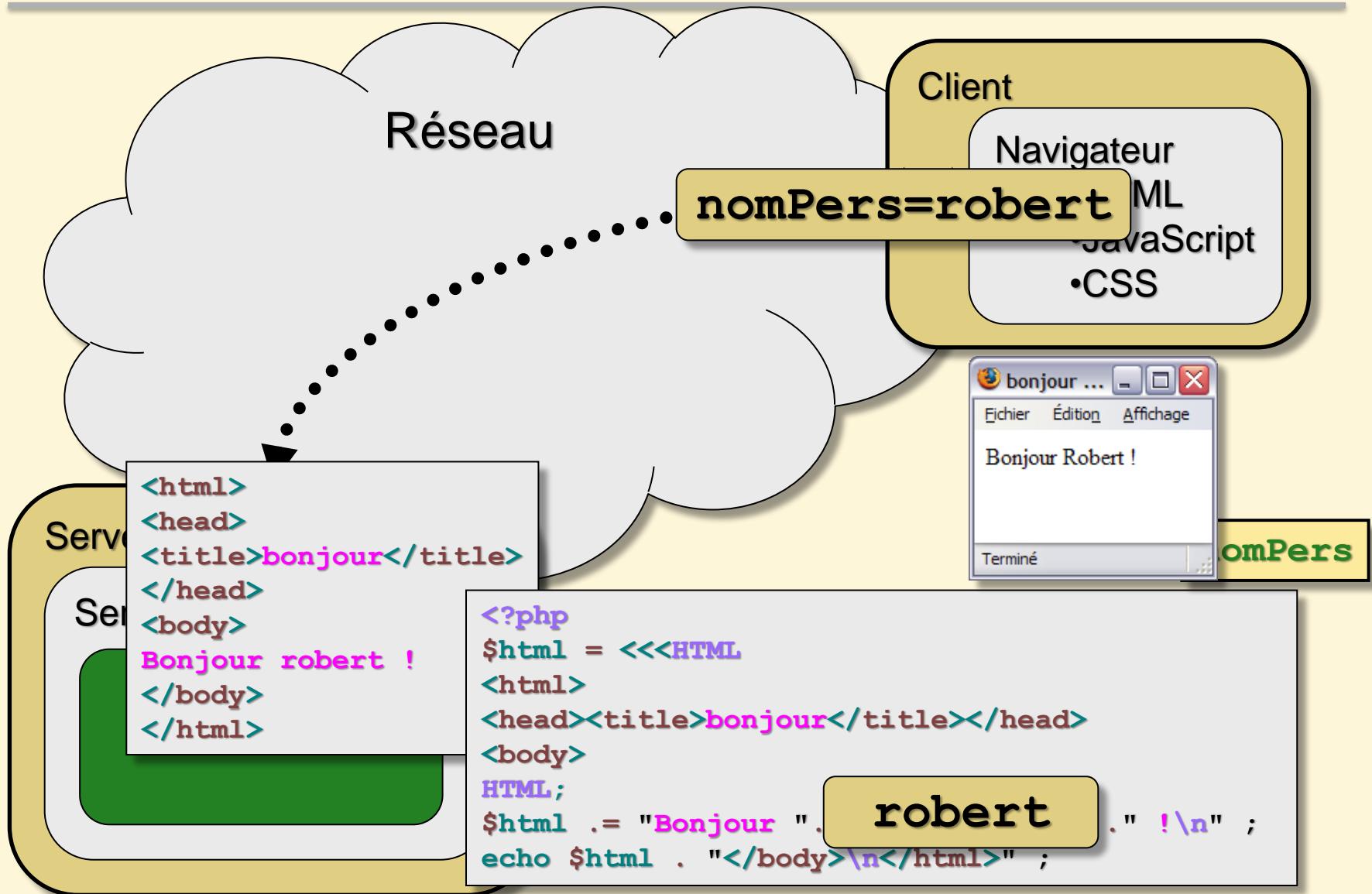
Le serveur compte... 1  
Le serveur compte... 2  
Le serveur compte... 3  
Le serveur compte... 4  
Le serveur compte... 5  
Le serveur compte... 6  
Le serveur compte... 7  
Le serveur compte... 8  
Le serveur compte... 9  
Le serveur compte... 10  
Le serveur compte... 11  
Le serveur compte... 12  
Le serveur compte... 13  
Le serveur compte... 14  
Le serveur compte... 15  
Le serveur compte... 16  
Le serveur compte... 17  
Le serveur compte... 18  
Le serveur compte... 19

# 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** (ou **\$\_REQUEST**).
- La valeur peut être trouvée grâce à une clé **qui porte le même nom** que le champs du formulaire de la page HTML de saisie.

# Traitement des données de formulaires



# Exemple – Formulaire HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>formulaire</title>
</head>
<body>
    <form action="valide1.php" method="get">
        Nom: <input type="text" name="nomPers">
        <input type="submit" value="Envoyer">
    </form>
</body>
</html>
```

# Exemple – Traitement en PHP

```
<?php
$html = <<<HTML
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Validation</title>
</head>
<body>
HTML;
if (isset($_GET['nomPers'])) {
    if (!empty($_GET['nomPers'])) {
        $html .= "Vous avez saisi \""
            .$_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 ?

# Formulaires contenant des champs « SELECT »

saisies.html

Choisissez des fruits:

Fraise  
Pomme  
Poire  
Banane  
Cerise

Envoyer

Intranet local

# Formulaires contenant des champs « SELECT unique »

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<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>
```

A diagram illustrating the form submission process:

- An arrow points from the "Envoyer" button to the "sel" select element.
- An arrow points from the "sel" select element to the URL "valide3.php?sel=Pomme".

# Formulaires contenant des champs « SELECT multiple »

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<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>
```

Envoyer

valide3.php?sel=Pomme&sel=Poire

???

# Formulaires contenant des champs « SELECT multiple »

```
<html>
<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>
```

Envoyer

The diagram illustrates the state of the 'sel[]' select element. A dashed arrow points from the 'sel[]' element to two yellow boxes at the bottom. The top box contains the URL 'valide3.php?sel%5B%5D=Pomme&sel%5B%5D=Poire'. The bottom box contains the URL 'valide3.php?sel[]=Pomme&sel[]=Poire'. This visualizes how the 'multiple' attribute creates a single parameter with multiple values.

# Traitement des données des champs « SELECT »

```
<?php
$html = <<<HTML
!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Liste de fruits</title>
</head>
<body>
HTML;
if (isset($_GET['sel']) && !empty($_GET['sel']))
/* La variable $_GET['sel'] est définie
et elle n'est pas vide */
foreach($_GET['sel'] as $fruit)
    $html .= "Vous avez choisi $fruit<br>\n";
}
else
    $html .= "Vous n'avez pas choisi de fruit\n";
echo $html . "</body>\n</html>" ;
```

`$_GET['sel']`  
est un tableau

# Résultat

Choisissez des fruits:

- Fraise
- Pomme
- Poire
- Banane
- Cerise

Envoyer

 Intranet local

Vous avez choisi Fraise  
Vous avez choisi Pomme  
Vous avez choisi Banane

 Intranet local

# Formulaires contenant des champs « CHECKBOX »

Choisissez des fruits:

Fraise  
 Pomme  
 Poire  
 Banane  
 Cerise

**Envoyer**

 Terminé  Intranet local

# Formulaires contenant des champs « CHECKBOX »

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Formulaire de saisie des fruits</title>
</head>
<body>
    <form name="formu" action="valide3.php" method="get">
        Choisissez des fruits :<br>
        <input type="checkbox" name="sel[]" value="Fraise">Fraise<br>
        <input type="checkbox" name="sel[]" value="Pomme">Pomme <br>
        <input type="checkbox" name="sel[]" value="Poire">Poire <br>
        <input type="checkbox" name="sel[]" value="Banane">Banane<br>
        <input type="checkbox" name="sel[]" value="Cerise">Cerise<br>
        <input type="submit" value="Envoyer">
    </form>
</body>
</html>
```

# Résultat

Choisissez des fruits:

Fraise  
 Pomme  
 Poire  
 Banane  
 Cerise

Intranet local

Vous avez choisi Fraise  
Vous avez choisi Pomme  
Vous avez choisi Banane

Intranet local

# Références

```
$a = 12 ;
```

```
$b = $a ;
```

```
$c = &$a ;
```

```
$b = "coucou" ;
```

```
$c = 84 ;
```

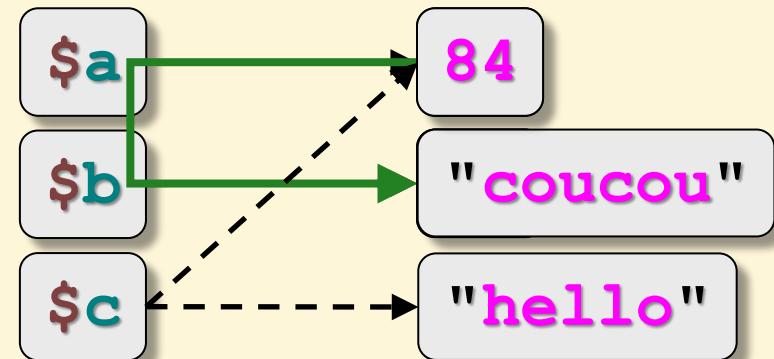
```
$a : 84
```

```
$b : coucou
```

```
$c : 84
```

```
unset($c) ;
```

```
$c = "hello" ;
```



# 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

- Valeur de retour

**function moyenne (\$a , \$b)**



```
{  
    ...  
}
```

Typage faible de PHP :  
Aucune information

- Arguments

**function moyenne ( \$a , \$b )**



```
{  
    ...  
}
```

Typage faible de PHP :  
Aucune information

# Mode de passage des arguments (types natifs)

```
<?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  
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 réussie

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

# 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("Marcel") ;
```

Bonjour cher Marcel

# 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

# include et require

Fichier mafonction.php

```
<?
function mafonction($arg)
{
    if (isset($arg))
    {
        echo ("Vrai") ;
    }
else
{
    echo ("Faux") ;
}
?>
```

Fichier utilisation1.php

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

Fichier utilisation2.php

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

Fichier utilisation3.php

```
...
require ("mafonction.php")
```

# Définition de constantes

```
<?php
```

```
define("ma_constante", "Bonjour à tous") ;
```

nom

valeur

Définition d'une constante

```
echo ma_constante ;  
?>
```

Utilisation de la constante

# 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

# Gestion des erreurs – (Mauvais) Exemple

```
<?php  
$html = <<<HTML  
<html>  
<head>  
<title>die-exit</title>  
</head>  
<body>  
HTML;  
  
if (!isset($val)) {  
    die($html."problème val");  
    /* Au delà de ce point,  
       fin du script */  
}  
$html .= <<<HTML  
Choix: $val  
</body>  
</html>
```

PHP

```
<html>  
<head>  
<title>die-exit</title>  
</head>  
<body>  
problème val
```

HTML

HTML non valide...

Navigateur

problème val

# Gestion de l'affichage des erreurs

- `int error_reporting ( [int level] )`

Ancien niveau d'erreur

Sur un serveur en production, toute erreur affichée donne des indices sur les scripts et rend le site vulnérable

`php.ini`

`display_errors boolean`

Constante
<code>E_ERROR</code>
<code>E_WARNING</code>
<code>E_PARSE</code>
<code>E_NOTICE</code>
<code>E_CORE_ERROR</code>
<code>E_CORE_WARNING</code>
<code>E_COMPILE_ERROR</code>
<code>E_COMPILE_WARNING</code>
<code>E_USER_ERROR</code>
<code>E_USER_WARNING</code>
<code>E_USER_NOTICE</code>
<code>E_ALL</code>
<code>E_STRICT</code>

Débogage

# Opérateur de contrôle d'erreur

```
$v = file("dummy.txt") Fichier absent  
or die("Problème de lecture");
```

**Warning:** file(dummy.txt): failed to open stream: No such file or directory in **dummy.php** on line **68**  
Problème de lecture

```
$v = @file("dummy.txt")  
or die("Problème de lecture");
```

Problème de lecture