PhP

©EISTI

2 mars 2010

Présentation

Présentation

- PhP est un langage de script serveur
- permet de créer dynamiquement des pages html
- interprété côté serveur par un serveur PhP (Apache/mySQL)
- génère une page HTML, épurée de tout code PhP, envoyée au client
- possède l'extension *.php*

Présentation

Buts

- générer dynamiquement du contenu HTML ou Javascript
- factorise le code HTML afin de ne pas le répéter dans chaque page
- permet d'exécuter des fonctions, d'interroger des bases de données

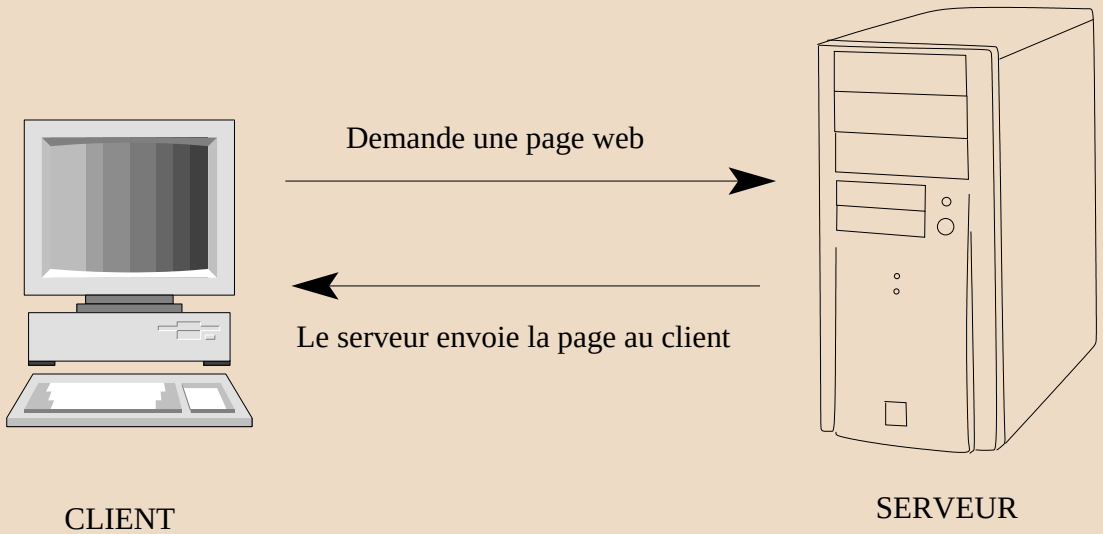
Présentation

PhP ne gère pas...

- le design de la page (HTML + CSS)
- le comportement du navigateur, l'exécution de scripts (Javascript)
- l'exécution de requêtes vers une base de données (MySQL, Oracle...)

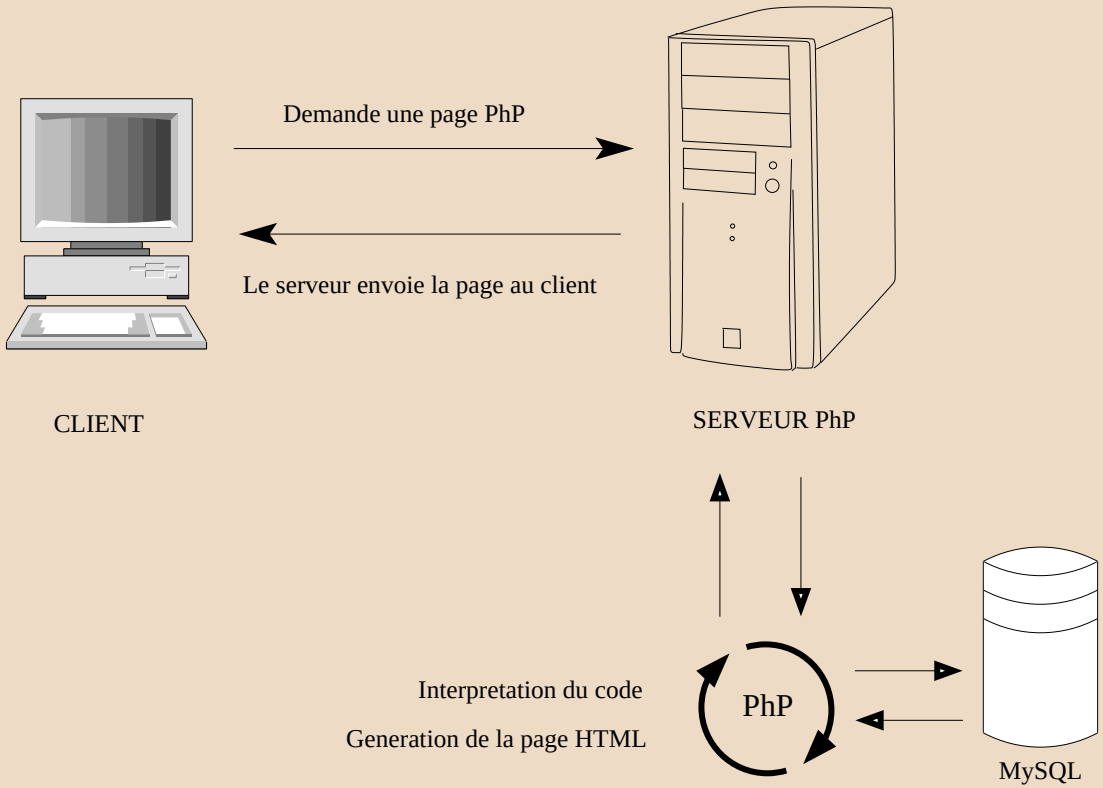
Présentation

Interrogation classique



Présentation

Interrogation PhP



Présentation

Principe

- les fichiers PhP se trouvent à un endroit particulier configurable dans le serveur web
- ce dossier est, en général, et avec Apache en particulier, repéré en local par l'url : `http://localhost/`

Caractéristiques

Caractéristiques

- orienté objet
- sensible à la casse
- inspiré de C++
- insertion par les balises <?php ... ?>

Caractéristiques

Caractéristiques

- inséré n'importe où dans le code html
- code non accessible côté client
- indépendant du système d'exploitation
- aide en ligne : `http://www.manuelphp.com/`

Exemple

Exemple

```
1  <html>
2  <head> ... </head>
3  <body>
4  <?php
5     //déclaration de variables
6     $nom = "Toto";
7     $age = 72;
8     //affichage
9     echo "L\'âge␣de␣$nom␣est␣:␣$age␣ans␣<br/>";
10    echo 'Quoi!! '.$age. ans'.$nom.' est si vieux que ça...';
11 ?>
12 </body>
13 </html>
```

Variables simples

Règles

- commentaires : //, /* ... */
- règles de nommage classiques
- préfixées par "$", ex : $password="peio"
- **les variables ne sont pas typées**
  SOYEZ TRÈS RIGOUREUX !!

```
$v1 = 'Toto' ;
$v2 = 3 ;
$v3 = 1.2 ;
$v4 = "-3.7" ;
$v5 = true ;
```

Variables simples

Constantes

- Pour définir une constante : **define()** la page en cours
- define("PI", 3.14159) ;
  echo(PI) ;

Variables simples

Quelques conseils...

- pas de marqueur explicite de variables globales/locales :
  SOYEZ RIGOUREUX ! !
  toute variable déclarée hors d'une fonction sera globale pour la page en cours
- bannissez au maximum les variables globales, utilisez des constantes

Variables simples

Opérateurs

- +, -, *, /,
- !=, ==, <=, >, ...
- !, &&, ||, ...
- ., .= : concaténation
- ===, !== : compare les valeurs ET les types des opérandes (ET l'ordre pour les tableaux)
- `instanceof` pour les types d'objets

Variables simples

Définition

- **defined(*var*)** : teste si une constante est définie
- **isset(*var*)** : teste si une variable est NULL ou non
- **empty(*var*)** : teste si une variable est à sa valeur vide ou non (0 pour un nombre, "" pour une chaîne, NULL, tableau vide, false, etc.
- **is_int(*var*)**, **is_string(*var*)**, **is_float(*var*)**, ... : teste le type de la variable

Variables simples

Tableaux

- $monTab = array() ;
- $ingredients = array('Farine', 'sucre', 'eau', 'sel', 'lait') ;
- $monTab[0] = "toto" ;

Variables simples

Associatifs

- $eleve = array() ;
  $eleve['nom'] = "toto" ;
  $eleve['prenom'] = "titi" ;
  $eleve['age'] = 12 ;
- $eleve = array('nom' => 'toto', 'prenom' => 'titi', 'age' => 12) ;
- echo $eleve['nom'] ;

Boucles et structures conditionnelles

Si ... Alors ... Sinon

```
if ($test == 1 && $i<=3) {
...
}else{ //elseif(...)
...
}
```

Boucles et structures conditionnelles

Pour

```
for ($i=0 ; $i<2 ; $i++){
...
}
```

Boucles et structures conditionnelles

Tant que

```
// exécuter une fois le traitement avant le test
do{
....
}while($a<10)
// tester avant d'exécuter une fois le traitement
while($a<10){
...
}
```

Boucles et structures conditionnelles

Selon

```
$age = 30;
switch ($age) { //variable de travail
case 20 :
echo "Salut jeune!!!";
break;
case 30 :
echo "Bonjour Monsieur";
}
```

Boucles et structures conditionnelles

Parcours de tableau

- ordinaires :

```
for($i=0 ;$i<sizeof($monTab) ;$i++){
   echo $nomTab[ $i ].'<br>' ;
}
ou bien
foreach($monTab as $contenu){
   echo $contenu.'<br />' ;
}
```

- associatifs :

```
foreach($monTab as $cle=>$valeur){
   echo $cle.'='.$valeur.'<br/>' ;
}
```

Fonctions

Fonctions

- se déclarent sous la forme :

```
function carre($val) {
    return $val * $val;
}
```

- se déclarent n'importe où (généralement au début du document)
- utilisation : *$res=carre(12);*

Fonctions

Passage de paramètres

- par copie de valeur :

```
function carre($val) {
   return $val * $val ;
}
...
$res = carre($var) ; //si var=4, res=16
```

- par référence :

```
function carreModif(&$val) {
   $val *= $val ;
}
...
carre($var) ; //si var=4 avant l'appel, après, var=16
```

où **&**$val est l'adresse mémoire de la variable $val

Fonctions

Fonctions diverses

Il existe de nombreuses fonctions PhP (`http://fr.php.net/manual/fr`)

- `date` : $maDate = date('d/m/Y') ; récupère la date dans un format spécifié
- `mail` : mail($email,$sujet,$message,$headers) ; envoi un mail
- `eval($ch)` : interprète la chaîne comme étant du code PhP
- fonctions concernant les chaines :
  - `strlen(chaine)` : longueur de chaîne
  - `str_replace(chAncienne,chNvlle,chaine)` : remplace un caractère par un autre
  - `nl2br(chaine)` : traduit les retours chariot ('\n') en HTML (
)
  - `htmlentities(chaine)` : supprime l'interprétation des balises HTML dans une chaine
  - `trim(chaine)` : supprime les espaces
  - `explode(delim, chaine)` : fragmente une chaine en tableau, par rapport à un délimiteur
  - `implode(delim, tab)` : reconstruit une chaine en tableau, par rapport à un délimiteur et un tableau

Include et require

Fonctions include et require

- `include("page.php")` et `require("page.php")` sont deux fonctions au comportement identique
- elles se remplacent par le code inclus du fichier page.php
- gestion d'erreur différente :
  - `require` génère une *erreur fatale* et ne continue pas le code,
  - `include`, génère une alerte et continue l'exécution du code
- `include_once('page.php')` et `require_once('page.php')` ont le même comportement que les deux précédentes fonctions mais le code ne sera ajouté qu'une seule fois

Exceptions

bloc try ... catch ...

```
1 function inverse($x){
2     try{
3         return 1/$x;
4     }catch(Exception $ex){
5         echo 'Exception!! : '.$ex->getMessage();
6     }
7 }
```

PhP et les formulaires

Rappels

- Un formulaire transmets des données vers une page selon 2 méthodes : get ou post
  - `get` :
  - `post` :

PhP et les formulaires

Récupération de données

- Les données arrivent au serveur sous forme de tableau
- le tableau est différent suivant le type de méthode d'envoi utilisée :
  - GET : `$_GET`
  - POST : `$_POST`
- utilisation :
  - GET :
    `http://www.monsite.fr/resultat.php?nom=ternet&prenom=alain`
    echo 'nom='.$_*GET*['*nom*'];
  - POST
    <label>Votre pseudo : <input type="text" id="pseudo" name="pseudo" id="name" value= "entrez-le ici"/></label>
    echo $_*POST*['*pseudo*'];