Chaînes de caractères en PHP

1. Syntaxe

Il y a plusieurs façon d'écrire les chaînes de caractères en PHP. Les deux principales sont de les entourer avec soit des guillemets simples, soit des guillemets doubles. Dans tous les cas, les chaînes de caractères peuvent se coller les unes derrière les autres (concaténation) avec l'opérateur point (`.`).

a. Les guillemets simples

Le moyen le plus simple de spécifier une chaîne de caractères est d'utiliser les guillemets simples : `'` . Pour spécifier un guillemet simple littéral, vous devez l'échapper avec un anti-slash (`\`). Si un anti-slash doit apparaître dans votre chaîne ou bien en fin de chaîne, il faudra le doubler (`\\`). Notez que si vous essayez d'échapper n'importe quel autre caractère, l'anti-slash sera conservé ! Il n'y a pas besoin d'échapper d'autres caractères que le guillemet lui-même.

Les variables ne sont pas interprétées dans une chaîne comportant des guillemets simples, il faut donc, pour afficher une variable dans une chaîne avec des guillemets simples, arrêter la chaînes, concaténer la variable, puis reprendre la chaîne.

Exemple

```php
<?php
echo 'Ceci est une chaîne simple';
echo 'Vous pouvez inclure des nouvelles
lignes dans une chaîne,
comme ceci.';
echo 'Arnold a coutume de dire : "I\'ll be back"';
// affiche : "I'll be back"
echo 'Êtes-vous sûr de vouloir effacer le dossier C:\\*.*?';
// affiche : Êtes-vous sûr de vouloir effacer le dossier C:\*.*?
echo 'Êtes-vous sûr de vouloir effacer le dossier C:\*.*?';
// affiche : Êtes-vous sûr de vouloir effacer le dossier C:\*.*?
echo 'Je suis en train de mettre une nouvelle ligne comme ceci : \n';
// affiche : Je suis en train de mettre une nouvelle ligne comme ceci : \n
echo 'Les variables ne seront pas $affichees $ici';
// affiche : Les variable ne seront pas $affichees $ici
$affichees = 'afficheeeeeeeeeeeees';
echo 'En revanche elles seront '.$affichees.' ici';
// affiche : En revanche elles seront afficheeeeeeeeeeeees ici
?>
```

b. Les guillemets doubles

Si la chaîne est entourée de guillemets doubles (`"`), PHP va comprendre certaines séquences de caractères :

Séquence	Valeur
`\n`	Nouvelle ligne (linefeed , LF ou 0x0A (10) en ASCII)
`\r`	Retour à la ligne (carriage return , CR ou 0x0D (13) en ASCII)
`\t`	Tabulation horizontale (HT ou 0x09 (9) en ASCII)
`\\`	Anti-slash
`\$`	Caractère $
`\"`	Guillemets doubles
`\[0-7]{1,3}`	Une séquence de caractères qui permet de rechercher un nombre en notation octale.
`\x[0-9A-Fa-f]{1,2}`	Une séquence de caractères qui permet de rechercher un nombre en notation hexadécimale.

Si vous essayez d'utiliser l'anti-slash sur n'importe quelle autre séquence, l'anti-slash sera affiché dans votre chaîne.

Le plus important pour les chaînes à guillemets doubles est le fait que les variables qui s'y trouvent seront remplacées par leur valeur. Il y a pour cela deux types de syntaxe, une simple et une complexe . La syntaxe simple est la plus courante et la plus pratique : elle fournit un moyen d'utiliser les variables, que ce soient des chaînes, des tableaux ou des membres d'objets. Dès qu'un signe dollar `$` est rencontré, l'analyseur PHP va lire autant de caractères qu'il peut pour former un nom de variable valide. Entourez le nom de la variable avec des accolades pour indiquer explicitement son nom.

Exemple

```
<?php
$boisson = 'vin';
echo "Du $boisson, du pain et du fromage!";
 // Correct, car "," n'est pas autorisé dans les noms de variables
echo 'Il a goûté plusieurs ' . $boissons;
 // Pas correct, car 's' peut faire partie d'un nom de variable, et
 // PHP recherchera alors $boissons
echo "Il a goûté plusieurs ${boisson}s";
 // Correct
?>
```

De la même façon, vous pouvez utiliser un élément de tableau ou un membre d'objet. Pour les éléments de tableau, le crochet fermant '`]` ' marquera la fin du nom de la variable. Pour les membres d'objets, les mêmes règles que ci-dessus s'appliquent. Cependant, il n'existe pas d'astuce comme cela pour les variables simples.

Exemple

```
<?php
// Ces exemples sont spécifiques à l'utilisation de tableaux dans une chaîne.
error_reporting(E_ALL);// Affichons toutes les erreurs
$fruits = array('fraise' => 'rouge', 'banane' => 'jaune');
// Fonctionne mais notez que cela fonctionne pas comme
// si cela était hors d'une chaîne
echo "Une banane est $fruits[banane].";
// Fonctionne
```

```php
echo "Une banane est {$fruits['banane']}.";
// Fonctionne mais PHP cherche une constante appelée banane
// tel que décrit ci-dessous
echo "Une banane est {$fruits[banane]}.";
// Ne fonctionne pas, il manque les accolades. Cela donne une erreur d'analyse
echo "Une banane est  $fruits['banane'].";
// Fonctionne
echo "Une banane est " . $fruits['banane'] . ".";
// Fonctionne
echo "Ce carré a un coté de $square->width mètres de large.";
// Ne fonctionne pas. Pour une solution, voyez la syntaxe complexe.
echo "Ce carré a un coté de $square->width00 centimètres.";
?>
```

La syntaxe est dite "complexe" car elle permet l'utilisation d'expressions complexes et non pas parce qu'elle serait obscure.

En fait, vous pouvez inclure n'importe quelle valeur qui est dans l'espace de nom avec cette syntaxe. Il suffit d'écrire une expression exactement comme si elle était hors de la chaîne, puis de l'entourer d'accolades `{}` . Puisque vous ne pouvez pas échapper les accolades, cette syntaxe ne commence qu'à partir du signe dollar, qui suit immédiatement l'accolade ouvrante. Vous pouvez utiliser "`{\$`" ou "`\{$`" pour obtenir un "`{$`" littéral.

Exemple

```php
<?php
error_reporting(E_ALL);// Affichons toutes les erreurs
$super = 'fantastique';
// Ne fonctionne pas. Affiche : Ceci est { fantastique}
echo "Ceci est { $super}";
// Fonctionne. Affiche Ceci est fantastique
echo "This is {$super}";
echo "This is ${super}";
// Fonctionne
echo "Ce carré a un coté de {$square->width}00 centimètres.";
// Fonctionne
echo "Ceci fonctionne : {$arr[4][3]}";
// Ceci est faux pour la même raison que $foo[bar] est faux
// hors d'une chaîne. En d'autres terme, cela va fonctionne
// car PHP recherche d'abord une constante appelée foo, mais
// il générera une note E_NOTICE (undefined constant).
echo "Ceci est faux : {$arr[foo][3]}";
// Fonctionne. Lorsque vous utilisez un tableau multidimensionnel dans
// une chaîne, n'oubliez jamais les accolades.
echo "Ceci fonctionne : {$arr['foo'][3]}";
// Fonctionne
echo "Ceci fonctionne : " . $arr['foo'][3];
echo "Vous pouvez même écrire {$obj->values[3]->name}";
echo "Ceci est une valeur de variable variable : {${$name}}";
?>
```

2. Utilisation pour écrire du HTML

Dans l'écriture du langage HTML, les attributs qui sont définis dans les balises utilisent des guillemets doubles.

Exemple :
```
<a href="http://www.lirmm.fr/~chateau/DUphp">Site du cours</a>
```

Lorsque l'on doit réaliser un affichage en langage php sur un navigateur, il faut produire du code HTML, et donc écrire de telles balises. Pour éviter la confusion des genres, il est alors recommandé d'utiliser des guillemets simples autour d'une chaîne en HTML.
En revanche, puisque les guillemets simples ne permettent pas l'interprétation des variables, il faut alors utiliser la technique de l'interruption de la chaîne, puis reprise éventuelle de la chaîne.

Exemple :
```
<?php
  $site = "Site du cours";
  echo '<a href="http://www.lirmm.fr/~chateau/DUphp">'.$site.'</a>';
?>
```

Lorsque la variable détermine elle-même une valeur d'un attribut, il ne faut pas s'emmêler les pinceaux!

Exemple :
```
<?php
   $gras = "font-weight:bold";
   echo 'Je veux &eacute;crire en <span style="'.$gras.'"> Gras </span>';
?>
```

3. Utilisation pour écrire des requêtes SQL

Pour les langages de requête de type SQL, comme MySQL, ce sont les guillemets simples qui sont utilisés.

Exemple
```
INSERT INTO personnes (nom,prenom) VALUES ('DUPONT','Jean')
```

Dans ce cas, il est recommandé d'utiliser des guillemets doubles pour entourer une chaîne qui représente une requête MySQL. Il n'y a alors pas besoin d'interrompre la chaîne, on peut utiliser les syntaxes autorisées pour indiquer des noms de variable comme précisé au-dessus.

Exemple :
```
<?php
  $nom = "DUPONT";
  $prenom = "Jean";
  $requete = "INSERT INTO personnes (nom,prenom) VALUES ('$nom','$prenom')";
?>
```

4. Quelques fonctions pratiques

a. Les fonctions de convertions et d'encodage

Nous avons vu au début du cours que les encodages sont parfois problématiques, selon que l'on dispose de données de type unicode, western-iso ou autres variantes.
De plus nous avons à gérer la conversion des caractères accentués en mode texte "normal" vers les caractères accentués du HTML.

convert_uuencode traduit toutes les chaînes en caractères imprimables, pour assurer leur transmission sur Internet. Les données au format uuencode sont environ 35 % plus grandes que les originales.

Exemple

```php
<?php
$some_string = "test\ntexte texte\r\n";
echo convert_uuencode($some_string);
?>
```

convert_uudecode décode une chaîne au format uuencode.

Exemple

```php
<?php
/* Pouvez-vous imaginer ce que cela va afficher ? :) */
echo convert_uudecode("+22!L;W9E(%!(4\"$`\n`");
?>
```

md5 calcule le MD5 de la chaîne de caractères en utilisant l'algorithme RSA Data Security, Inc. MD5 Message-Digest Algorithm , et retourne le résultat. Le résultat est un nombre de 32 caractères hexadécimaux.

Exemple

```php
<?php
$str = 'pomme';
if (md5($str) === 'ede0f9c3a1d2093e3f48fcafd3c70915') {
    echo "Voulez-vous une golden ou une spartan?";
    exit;
}
?>
```

nl2br retourne string après avoir inséré `'<br />'` devant toutes les nouvelles lignes.

Exemple

```php
<?php
echo nl2br("foo n'est pas\n bar");
?>
```

htmlentities traduit tous les caractères qui ont des équivalents en entités HTML.
Comme la fonction htmlspecialchars, cette fonction prend un deuxième argument optionnel, qui indique comment doivent être traités les guillemets doubles et simples. Vous pouvez

utiliser l'une des constantes suivantes la valeur par défaut étant `ENT_COMPAT` :

Nom	Description
`ENT_COMPAT`	Convertit les guillemets doubles, et ignore les guillemets simples
`ENT_QUOTES`	Convertit les guillemets doubles et les guillemets simples
`ENT_NOQUOTES`	Ignore les guillemets doubles et les guillemets simples

Comme htmlspecialchars, cette fonction prend un troisième argument optionnel qui définit le jeu de caractères utilisé durant la conversion. Actuellement, le jeu de caractères ISO-8859-1 (Europe occidentale, latin1) est utilisé par défaut. On peut par exemple spécifier le jeu de caractère utf8 avec la constante `UTF-8`

Exemple

```
<?php
$str = 'Un \'apostrophe\' en <strong>gras</strong>';
// Affiche : Un 'apostrophe' en &lt;strong&gt;gras&lt;/strong&gt;
echo htmlentities($str);
// Affiche : Un &#039;apostrophe&#039; en &lt;strong&gt;gras&lt;/strong&gt;
echo htmlentities($str, ENT_QUOTES);
?>
```

html_entity_decode convertit toutes les entités HTML en caractères normaux

Exemple

```
<?php
$orig = 'J\'ai "sorti" le <strong>chien</strong> tout à l\'heure';
$a = htmlentities($orig);
$b = html_entity_decode($a);
echo $a; // J'ai &quot;sorti&quot; le &lt;strong&gt;chien&lt;/strong&gt; tout
&amp;agrave; l'heure
echo $b; // J'ai "sorti" le <strong>chien</strong> tout à l'heure
?>
```

b. Les fonctions réglant la casse

strtolower retourne la chaîne passée en paramètre, après avoir converti tous les caractères alphabétiques en minuscules. Notez que la notion d'"alphabétique" est déterminée par la configuration de localisation. Cela signifie que pour la configuration par défaut "C", les caractères tels que les voyelles accentuées (comme é, è ou à) ne seront pas convertis.

Exemple

```
<?php
$str = "Marie A un Petit Agneau, et l'aime TRès fORt.";
$str = strtolower($str);
echo $str; // marie a un petit agneau, et l'aime très fort.
?>
```

strtoupper retourne la chaîne passée en paramètre, après avoir converti tous les caractères alphabétiques en majuscules.

Exemple

```
<?php
```

```php
$str = "Marie A un Petit Agneau, et l'aime fORt.";
$str = strtoupper($str);
echo $str; // MARIE A UN PETIT AGNEAU, ET L'AIME FORT.
// Note : Très aurait été converti en TRèS
?>
```

ucfirst retourne la chaîne après avoir remplacé le premier caractère par sa majuscule, si le premier caractère est alphabétique.

Exemple

```php
<?php
$foo = 'bonjour tout le monde!';
$foo = ucfirst($foo);             // Bonjour tout le monde!
$bar = 'BONJOUR TOUT LE MONDE!';
$bar = ucfirst($bar);             // BONJOUR TOUT LE MONDE!
$bar = ucfirst(strtolower($bar)); // Bonjour tout le monde!
?>
```

c. Les fonctions de découpage et de rabotage

trim retourne la chaîne passée en paramètre, après avoir supprimé les caractères invisibles en début et fin de chaîne. Si le second paramètre charlist est omis, trim supprimera les caractères suivants :

- `" "` (ASCII 32 (0x20)), un espace ordinaire.
- `"\t"` (ASCII 9 (0x09)), une tabulation.
- `"\n"` (ASCII 10 (0x0A)), une nouvelle ligne (line feed).
- `"\r"` (ASCII 13 (0x0D)), un retour chariot (carriage return).
- `"\0"` (ASCII 0 (0x00)), le caractère NUL .
- `"\x0B"` (ASCII 11 (0x0B)), une tabulation verticale.

Exemple

```php
<?php
$text = "\t\tVoici quelques mots :) ...  ";
echo trim($text);         // "Voici quelques mots :) ..."
echo trim($text, " \t."); // "Voici quelques mots :)"
// supprime tous les caractères de contrôle ASCII au début de la chaîne de caractères.
// (de 0 à 31 inclus)
$clean = trim($binary, "\x00..\x1F");
?>
```

Suppression de caractères dans un tableau

```php
<?php
function trim_value(&$value){
    $value = trim($value);
}
$fruit = array('pomme','banane ', ' canneberge ');
var_dump($fruit);
array_walk($fruit, 'trim_value');
var_dump($fruit);
?>
```

ltrim retourne la chaîne, après avoir supprimé les caractères invisibles de début de chaîne.

rtrim retourne la chaîne, après avoir supprimé les caractères invisibles de fin de chaîne.

substr(string, start, length) retourne le segment de `string` défini par `start` et `length`.

Si `start` est positif, la chaîne retournée commencera au caractère numéro `start` , dans la chaîne `string` . Le premier caractère est numéroté zéro.

Exemple

```php
<?php
echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd
echo substr('abcdef', 0, 8);  // abcdef
echo substr('abcdef', -1, 1); // f
// Accéder à un simple caractère dans une chaîne
// peut également être réalisé en utilisant des accolades
$string = 'abcdef';
echo $string{0};                 // a
echo $string{3};                 // d
echo $string{strlen($string)-1}; // f
?>
```

Si `start` est négatif, la chaîne retournée commencera au caractère numéro `start` à compter de la fin de la chaîne `string` .

Exemple

```php
<?php
$rest = substr("abcdef", -1);    // retourne "f"
$rest = substr("abcdef", -2);    // retourne "ef"
$rest = substr("abcdef", -3, 1); // retourne "d"
?>
```

Si `length` est fourni et est positif, la chaîne retournée contiendra au plus `length` caractères, en commençant à partir du caractère `start` (en fonction de la taille de la chaîne `string`). Si `string` est plus petite que `start`, substr retournera FALSE .

Si `length` est fourni et négatif, alors le même nombre de caractères sera omis, en partant de la fin de la chaîne `string` . Si `start` représente une position hors de la chaîne, une chaîne vide sera retournée.

Exemple

```php
<?php
$rest = substr("abcdef", 0, -1);  // retourne "abcde"
$rest = substr("abcdef", 2, -1);  // retourne "cde"
$rest = substr("abcdef", 4, -4);  // retourne ""
$rest = substr("abcdef", -3, -1); // retourne "de"
?>
```

wordwrap(str,width,break,cut) retourne la chaîne `str` , après avoir inséré `break` tous les `width` caractères.
Par défaut, wordwrap va automatiquement insérer une nouvelle ligne en utilisant `\n` tous les 75 caractères, si `width` ou `break` ne sont pas fournis.

Exemple

```
<?php
$text = "Portez ce vieux whisky au juge blond qui fume.";
$newtext = wordwrap( $text, 20 );
echo "$newtext\n";
?>
```

Si le paramètre `cut` est mis à 1, la césure de la chaîne sera toujours à la taille `width` . Si vous avez un mot qui est plus long que la taille de césure, il sera coupé en morceaux. (Voir le second exemple.)

Exemple

```
<?php
$text = "Un mot très très looooooooooooooooong.";
$newtext = wordwrap($text, 8, "\n", 1);
echo "$newtext\n";
?>
```

d. Les fonctions de protection des caractères spéciaux

addslashes retourne la chaîne, après avoir échappé tous les caractères qui doivent l'être, pour être utilisée dans une requête de base de données. Ces caractères sont les guillemets simples (`'`), guillemets doubles (`"`), anti-slash (`\`) et `NUL` (le caractère NULL).

Un exemple d'utilisation de cette fonction est lorsque vous entrez des données dans une base de données. Par exemple, pour insérer le nom O'reilly dans la base, vous aurez besoin de le protéger. La plupart des bases de données le font avec \ ce qui donnerait O\'reilly . Cela ne servirait qu'à insérer le nom dans la base de données, le \ ajouté ne sera pas inséré. Si la directive PHP `magic_quotes_sybase` est activée, le guillemet simple `'` sera protégé par un autre `'` .

La directive PHP `magic_quotes_gpc` est à `on` par défaut, et elle appelle addslashes sur toutes les données `GET`, `POST` et `COOKIE`. N'utilisez pas addslashes sur des données déjà protégées avec magic_quotes_gpc sinon vous doublerez les protections. La fonction magic_quotes_gpc est utile pour vérifier ce paramètre.

Exemple

```
<?php
$str = "Votre nom est-il O'reilly ?";
echo addslashes($str); // Votre nom est-il O\'reilly ?
?>
```

stripslashes retourne une chaîne dont les anti-slash on été supprimés. `\'` devient `'` , etc. Les

doubles anti-slash sont réduits à un seul anti-slash. C'est la fonction inverse de addslashes.

Si `magic_quotes_sybase` est activée, aucun anti-slash n'est supprimé mais deux apostrophes sont remplacées par une seule à la place.

Un exemple d'utilisation de stripslashes est lorsque la directive PHP `magic_quotes_gpc` est à `on` (valeur par défaut) et que vous insérez des données dans une base de données qui requiert la protection des valeurs. Par exemple, si vous affichez simplement et directement des données provenant d'un formulaire HTML.

Exemple

```
<?php
$str = "Avez-vous l\'oreille dure?";
// Affiche : Avez-vous l'oreille dure?
echo stripslashes($str);
?>
```

La fonction stripslashes n'est pas récursive. Si vous voulez appliquer cette fonction à un tableau multi-dimensionnel, vous devez utiliser une fonction récursive.

Exemple

```
<?php
function stripslashes_deep($value){
    if (is_array($value))
       $value = array_map('stripslashes_deep', $value);
    else
       $value = stripslashes($value);
    return $value;
}
// Exemple
$array = array("f\\'oo", "b\\'ar", array("fo\\'o", "b\\'ar"));
$array = stripslashes_deep($array);
// Affichage
print_r($array);
?>
```

get_magic_quotes_gpc retourne la configuration actuelle de l'option `magic_quotes_gpc` (0 pour l'option désactivée, 1 pour l'option activée).

Si la directive `magic_quotes_sybase` est activée, elle remplacera complètement `magic_quotes_gpc`. Ce qui fait que même si get_magic_quotes retourne TRUE les guillemets doubles, les anti-slashes ou les caractères NULL ne seront pas protégés. Seul les guillemets simples le seront. Dans ce cas, ils ressembleront à **''** .

Exemple

```
<?php
echo get_magic_quotes_gpc();         // 1
echo $_POST['lastname'];             // O\'reilly
echo addslashes($_POST['lastname']); // O\\\'reilly
```

```php
if (!get_magic_quotes_gpc()) {
    $lastname = addslashes($_POST['lastname']);
} else {
    $lastname = $_POST['lastname'];
}
echo $lastname; // O\'reilly
$sql = "INSERT INTO lastnames (lastname) VALUES ('$lastname')";
?>
```

quotemeta retourne la chaîne passée en paramètre après avoir introduit un anti-slash (\ ) devant tous les caractères suivants : `. \ + * ? [ ^ ] ( $ )`

strip_tags retourne la chaîne passée en paramètre après avoir supprimé toutes les balises PHP et HTML du code. Elle génère des alertes si les balises sont incomplètes ou erronées.

Le deuxième paramètre, optionnel, permet d'indiquer les balises qui doivent être conservées.

Attention

Comme strip_tags ne valide actuellement pas le HTML, les balises partielles ou rompues peuvent conduire à la suppression de plus de textes/données que désiré.

Attention

strip_tags ne modifie pas les attributs des balises que vous autorisez via le paramètre allowable_tags , y compris les attributs `style` et `onmouseover`, que des utilisateurs mal intentionnés peuvent utiliser.

Exemple

```php
<?php
$text = '<p>Test paragraphe.</p> Autre texte';
echo strip_tags($text);
echo "\n";
// Permet la balise <p>
echo strip_tags($text, '<p>');
?>
```

e. Les fonctions de comptage et de répétition

strlen retourne la taille de la chaîne.

Exemple

```php
<?php
$str = 'abcdef';
echo strlen($str); // 6
$str = ' ab cd ';
echo strlen($str); // 7
?>
```

count_chars compte le nombre d'occurrences de tous les octets présents dans la chaîne string et retourne différentes statistiques. Le deuxième paramètre, optionnel, vaut par défaut

0. Suivant sa valeur, on obtient les informations suivantes :
- 0 - un tableau avec l'octet en index, et la fréquence correspondante pour chaque octet.
- 1 - identique à 0 mais seules les fréquences supérieures à zéro sont listées.
- 2 - identique à 0 mais seules les fréquences nulles sont listées.
- 3 - une chaîne contenant tous les octets utilisés est retournée.
- 4 - une chaîne contenant tous les octets non utilisés est retournée.

Exemple

```php
<?php
$data = "Deux D et un F.";
foreach (count_chars($data, 1) as $i => $val) {
    echo "Il y avait $val occurrence(s) de \"" , chr($i) , "\" dans la phrase.\n";
}
?>
```

str_repeat(input, mult) retourne la chaîne `input` répétée `mult` fois. `mult` doit être positif ou nul. Si `mult` vaut 0, la fonction retourne la chaîne vide.

Exemple

```php
<?php
  echo str_repeat("-=", 10);
?>
```

f. Les fonctions de gestion d'url

parse_str analyse la chaîne de caractères comme s'il s'agissait d'une requête HTTP, passée via l'URL. Toutes les variables qu'elle y repère sont alors créées, avec leurs valeurs respectives. Si le second paramètre est fourni, les variables y seront stockées, sous forme d'index de tableau. Pour accéder à l'URL appelante `QUERY_STRING` , vous devez utiliser la variable `$_SERVER['QUERY_STRING']`.

Exemple

```php
<?php
$str = "first=value&arr[]=foo+bar&arr[]=baz";
parse_str($str);
echo $first;  // value
echo $arr[0]; // foo bar
echo $arr[1]; // baz
parse_str($str, $output);
echo $output['first'];  // value
echo $output['arr'][0]; // foo bar
echo $output['arr'][1]; // baz
?>
```

urldecode décode une chaîne encodée comme une URL.

Exemple

```php
<?php
$a = explode('&', $QUERY_STRING);
$i = 0;
while ($i < count($a)) {
```

```
    $b = split('=', $a[$i]);
    echo 'La valeur du paramètre ', htmlspecialchars(urldecode($b[0])),
         ' est ', htmlspecialchars(urldecode($b[1])), "<br />\n";
    $i++;
}
?>
```

urlencode encode une chaîne en URL dont lescaractères non alpha-numériques (hormis -_.) sont remplacés par des séquences commençant par un caractère pourcentage (%), suivi de deux chiffres hexadécimaux. Les espaces sont remplacés par des signes plus (+). Ce codage est celui qui est utilisé pour poster des informations dans les formulaires HTML. Le type MIME est *application/x-www-form-urlencoded*.

Exemple

```
<?php
echo '<a href="mycgi?foo=', urlencode($userinput), '">';
$query_string = 'foo=' . urlencode($foo) . '&bar=' . urlencode($bar);
echo '<a href="mycgi?' . htmlentities($query_string) . '">';
?>
```

g. Les fonctions de comparaison

strcmp(str1,str2) retourne < 0 si `str1` est inférieure à `str2` ; > 0 si `str1` est supérieure à `str2` , et 0 si les deux chaînes sont égales. Notez que cette comparaison est sensible à la casse.

strcasecmp(str1,str2) fait comme strcmp, mais n'est pas sensible à la casse

Exemple

```
<?php
$var1 = "Hello";
$var2 = "hello";
if (strcasecmp($var1, $var2) == 0) {
    echo "$var1 est égale à $var2 (comparaison insensible à la casse)";
}
?>
```