Les entrées/sorties 3

PLAN

OBJECTIFS

- Connaître les différentes syntaxes pour saisir des données et afficher des résultats aussi bien dans un environnement shell que web.

3.1 Le contexte d'interprétation

En mode shell, un programme PHP reçoit les informations directement du clavier ou *via* la ligne de commande, et affiche les résultats sur l'écran au format texte.

Dans un environnement web, un programme PHP effectue les traitements sur des données dont la source peut être un formulaire HTML ou une information transmis *via* l'URL.

Ce chapitre présente les différentes possibilités d'entrées (les saisies) et de sorties (les affichages) selon ces deux environnements d'exécution, shell ou web.

3.2 Dans un environnement shell

Dans cet environnement, la saisie se fait au clavier et l'affichage apparaît à l'écran au format texte.

Les entrées

La saisie au clavier

- *L'instruction fscanf()*

Le langage PHP a été conçu principalement pour un environnement web. Il n'existe pas d'instruction spécifique à la saisie au clavier. Cependant, tout comme le langage C

dont il hérite certaines syntaxes, on peut utiliser la fonction `fscanf()` de lecture de fichier appliquée au « fichier » d'entrée standard `STDIN` qui pointe sur le clavier. Les formats sont présentés dans le tableau 3.1.

Tableau 3.1 – Les formats de fscanf()

Type de données	Exemple
Entier	`fscanf(STDIN,"%d",$age) ;`
Caractère	`fscanf(STDIN,"%c",$initiale) ;`
Réel	`fscanf(STDIN,"%f",$rayon) ;`
Chaîne	`fscanf(STDIN,"%s",$nom) ;`

- *Les instructions fgetc() et fgets()*

Le langage PHP propose deux instructions spécifiques à la saisie, en mode shell, d'un caractère (chaîne d'un seul caractère) et d'une chaîne de caractères. Elles sont conçues pour lire les données à partir d'un fichier.

On peut utiliser la fonction `fgetc()` de lecture de fichier appliquée à l'entrée standard `STDIN`, pour effectuer la saisie au clavier. La syntaxe est identique pour `fgets()`. Les formats sont présentés dans le tableau 3.2.

Tableau 3.2 – Les formats de fgetc() et fgets()

Fonction	Exemple
fgetc()	`$caractere = fgetc(STDIN) ;`
fgets()	`$nom = fgets(STDIN) ;`

Remarque

La fonction `fgetc()` lit un seul caractère. Après la lecture du caractère, la mémoire tampon (buffer) du clavier contient toujours le caractère `Line Feed` (LF=validation) qui n'est pas lu, ce qui peut perturber le déroulement de la saisie suivante. La fonction `fgets()` lit une ligne complète y compris le caractère fin de ligne (LF=validation).

Les arguments de la ligne de commande *argc* et *argv[]*

Quand on exécute le programme *via* la commande `php`, il est possible de passer des arguments (des données) au lancement de cette exécution. La variable `$argc` contient alors le nombre d'arguments et le tableau `$argv[]` contient leurs valeurs. Le programme `argc_argv_shell.php` présente ces syntaxes :

Listing 3.1 – Programme argc_argv_shell.php

```
<?php
 $nb_arguments=$argc ;
 echo "-- boucle for utilisant argc --\n";
 for ($i=0;$i<$nb_arguments;$i++)
```

```
 {echo "Argument ".$i." : ".$argv[$i]."\n" ;
 }
 echo"\n";
?>
```

Voici un exemple d'exécution avec trois arguments, deux chaînes de caractères « Dupont » et « Jean », et la valeur entière 25 :

Listing 3.2 – Exécution de argc_argv_shell.php

```
$ php argc_argv_shell.php Dupont Jean 25
-- boucle for utilisant argc --
Argument 0 : argc_argv_shell.php
Argument 1 : Dupont
Argument 2 : Jean
Argument 3 : 25
```

La case 0 du tableau `argv[]` contient le nom du programme. Les autres arguments « Dupont », « Jean » et « 25 » sont rangés dans les cases 1, 2 et 3.

Les sorties

L'instruction echo

- *Syntaxe générale*

L'instruction `echo` a été utilisée dans des exemples précédents. Ce n'est pas une fonction mais un élément du langage. Voici des exemples de syntaxe :

```
echo "texte" ;
echo "texte1","texte2" ; // Liste de paramètres
echo $i ; // Contenu d'une variable
echo "Le résultat est : ".$i ; //Concatène texte et variable
echo "Le résultat est : ",$i ; // Liste de valeurs
echo "Le résultat est : ".$i."\n" ; // Avec saut de ligne
echo "Le résultat est : ",$i,"\n" ; // Avec saut de ligne
```

- *Les apostrophes et les guillemets*

Le texte affiché par `echo` peut être entre guillemets ", ou entre apostrophes '. Le programme `echo_guillemets_apostrophes.php` présente ces deux syntaxes :

Listing 3.3 – Programme echo_guillemets_apostrophes.php

```
<?php
 echo "bonjour\n";
 echo 'bonjour\n';
?>
```

Les guillemets interprètent les syntaxes particulières. La notation `\n` est reconnue comme un saut de ligne en mode shell, et il est effectué. Les apostrophes ignorent les syntaxes particulières. La notation `\n` est assimilée à deux caractères qui sont simplement affichés. Voici l'exécution de ce programme en shell :

Listing 3.4 – Exécution de echo_guillemets_apostrophes.php

```
$ php echo_guillemets_apostrophe.php
bonjour
bonjour\n$
```

- *La déspécialisation des guillemets et des apostrophes*

Les guillemets et les apostrophes sont des délimiteurs de texte. Pour les afficher, il faut les « déspécialiser » avec le caractère « \ », comme dans la syntaxe suivante :

```
echo "Afficher un guillemet \"comme ceci\".";
```

L'instruction print

L'instruction `print` diffère de `echo` sur deux points : on peut l'utiliser avec des parenthèses ; elle affiche une chaîne de caractères et non une liste d'arguments.

Voici quelques syntaxes de cette instruction :

```
print("print() avec les parenthèses\n");
print "print sans les parenthèses.\n";
print "Voici un print
écrit sur plusieurs lignes\n";
print "Afficher un guillemet \"comme ceci\".\n";
print 'Avec des apostrophes : bonjour\n';
$age=65;
print "L'âge limite est:".$age." ans\n"; // Avec concaténation
```

L'instruction *printf*

Cette instruction est similaire à celle du langage C. Seul le format « %c » change. Il affiche alors le caractère dont l'*entier* est donné en argument. Les formats sont présentés dans le tableau 3.3.

Tableau 3.3 – Les formats de printf()

Type de données	Exemple
Entier	`printf("%d",$age) ;` `printf("%c",$numlettre) ;`
Caractère	`printf("%s",$initiale) ;`
Réel	`printf("%f",$rayon) ;`
Chaîne	`printf("%s",$nom) ;`

L'instruction *fputs*

Le langage PHP propose une instruction spécifique à l'affichage des chaînes de caractères. Elle est conçue pour écrire dans un fichier. Cependant, on peut l'utiliser sur la sortie standard `STDOUT` qui correspond à l'écran. Le format est présenté dans le tableau 3.4.

Tableau 3.4 – Le format de fputs()

Fonction	Exemple
fputs()	`fputs(STDOUT, $nom) ;`

Exemples

Cette section présente la mise en œuvre de la saisie et de l'affichage en environnement shell *via* les instructions `echo, fscanf(), printf(),fgetc(), fgets(), fputs()`, pour chaque type de données. La fonction `setlocale()`gère l'affichage des dates et des décimales en français.

Pour un entier, un réel, un caractère et une chaîne de caractères

Le programme `saisie_affichage_entier_shell.php` traite les entiers.

Listing 3.5 – Programme saisie_affichage_entier_shell.php

```
<?php
// Définit les dates, et numériques au format local (français)
setlocale(LC_ALL, '') ;
echo "Entrez un entier (ex : 65) : ";
fscanf(STDIN, "%d", $number);
echo "echo:La valeur saisie par fscanf() est:".$number . "\n";
printf("printf:La valeur saisie est : %d\n",$number) ;
printf("printf:La valeur saisie est : %020d\n",$number) ;
printf("printf:Le caractère équivalent est : %c   <== format %%c en
PHP appliqué à un entier\n",$number) ;
?>
```

Le programme `saisie_affichage_caract_shell.php` traite un caractère.

Listing 3.6 – Programme saisie_affichage_caract_shell.php

```
<?php
// Définit les dates, et numériques au format local (français)
setlocale(LC_ALL, '') ;
echo "\n";
echo "Entrez un caractère (ex : Q) : ";
fscanf(STDIN, "%c", $caract);
echo "echo:Valeur saisie avec fscanf():".$caract."\n";
printf("printf:Valeur saisie:%c<== problème de %%c en PHP\n",$caract) ;
```

```
printf("printf:Valeur saisie:%s <== format %%s en PHP\n",$caract) ;
echo "Entrez un caractère (ex : Q) : ";
$caract=fgetc(STDIN);
echo "echo:Valeur saisie avec fgetc():".$caract."\n";
fputs(STDOUT,"fputs: Valeur saisie avec fgetc():".$caract . "\n");
?>
```

Le programme `saisie_affichage_reel_shell.php` traite les réels.

Listing 3.7 – Programme saisie_affichage_reel_shell.php

```
<?php
// Définit la présentation des dates, valeurs numériques
// au format local (français)
setlocale(LC_ALL, '') ;
// --- Saisie et affichage d'un réel ---
echo "\n";
echo "Entrez un réel avec le point décimal (ex : 2.8) : ";
fscanf(STDIN, "%f", $reel);
echo "echo   : La valeur saisie est : " . $reel . "\n" ;
printf("printf : La valeur saisie est : |%+10.2f|\n",$reel) ;
?>
```

Le programme `saisie_affichage_chaine_shell.php` traite les chaînes de caractères.

Listing 3.8 – Programme saisie_affichage_chaine_shell.php

```
<?php
// Définit la présentation des dates, valeurs numériques
// au format local (français)
setlocale(LC_ALL, '') ;
// --- Saisie et affichage d'une chaîne ---
echo "\n";
echo "Entrez une chaîne (ex : azerty) : ";
fscanf(STDIN, "%s", $chaine);
echo "echo   : La valeur saisie avec fscanf() est : " . $chaine . "\n"
;
printf("printf : La valeur saisie est : |%-10s|   <== cadrage à
gauche\n",$chaine) ;
echo "Entrez une chaîne (ex : azerty) : ";
$chaine=fgets(STDIN);
$chaine=trim($chaine);  // On retire le caractère fin de ligne
                        // de la variable chaine
echo "echo   : La valeur saisie avec fgets() est : " . $chaine . "\n"
;
fputs(STDOUT,"fputs : La valeur saisie avec fgets() est : " . $chaine
. "\n") ;
?>
```

Pour un réel géré en chaîne de caractères

Le programme `saisie_affichage_reel_chaine_shell.php` lit un nombre réel en chaîne de caractères puis le convertit en numérique, ce qui permet la saisie d'un réel au format français (virgule décimale) comme : 3,2.

Listing 3.9 – Programme saisie_affichage_reel_chaine_shell.php

```
<?php
// Dates, et Valeurs numériques au format local (français)
setlocale(LC_ALL, '') ;
// Saisie et affichage d'un réel au format chaine
echo "Entrez un réel en français (ex : 2,8) : ";
fscanf(STDIN, "%s", $chaine2);
// Conversion en type réel
$chaine3 = str_replace(",",".",$chaine2) ;
$reel3   = (float) $chaine3              ;
echo "echo:La valeur réelle est:".$reel3."\n"        ;
printf("printf:Valeur réelle est:|%+6.2f|\n",$reel3) ;
?>
```

Voici son exécution :

Listing 3.10 – Exécution de saisie_affichage_reel_chaine_shell.php

```
Entrez un réel en français (ex : 2,8) : 3,2
echo:La valeur réelle est:3,2
printf:Valeur réelle est:| +3,20|
```

3.3 Dans un environnement web

Les entrées/sorties dans un environnement web utilisent principalement les formulaires ou les arguments de l'URL.

Nous présentons ces deux possibilités.

Les formulaires

Nous reprenons l'exemple présenté au chapitre 1 (voir listing 1.3).

Rappel

Un formulaire HTML saisit des informations qui sont généralement au format texte. Le fichier `formulaire_post.html` saisit le nom, le prénom et l'âge d'une personne. La figure 3.1 présente cette page HTML.

Figure 3.1 –Affichage de formulaire_post.html.

La sélection du bouton « Valider » transmet les données au programme indiqué dans le champ *action* du formulaire *via* la méthode POST ou GET.

La méthode POST

Avec la méthode POST les informations passées après validation sont *invisibles* dans le champ URL du navigateur. Voici le fichier `formulaire_post.html` qui implémente cette méthode :

Listing 3.11 – Fichier formulaire_post.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Saisie d'une personne</title>
 </head>
 <body>
  <form action="formulaire_post.php" method="post">
   Nom : <input type="text" name="nom" size="20" /><br/>
   Pr&eacute;nom : <input type="text" name="prenom" size="30" /><br/>
   Age : <input type="text" name="age" size="10" /><br/><br/>
   <input type="submit" value="Valider" />
   <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Les syntaxes qui balisent le formulaire sont :

```
<form action="formulaire_post.php" methode="post">
</form>
```

Le bouton « Valider » envoie les données à traiter *via* la méthode POST au programme PHP `formulaire_post.php` que voici :

Listing 3.12 – Programme formulaire_post.php

```
<!DOCTYPE html>
<html>
    <head>
     <meta charset="utf-8" />
     <title>Traitement d'une personne</title>
    </head>
    <body>
    <!-- Début du programme PHP -->
    <?php
      // == On récupère les données ==
      $nom    = $_POST['nom']   ;
      $prenom = $_POST['prenom'];
      $age    = $_POST['age']   ;
      // == On traite les données ==
      // -- Conversion en majuscules --
      $nom    = strtoupper($nom)   ;
      $prenom = strtoupper($prenom);
      // -- Conversion en entier --
      $age    = intval($age)       ;
      // == On affiche les données ==
      echo 'Nom : '.$nom.'<br/>';
      echo 'Pr&eacute;nom : '.$prenom.'<br/>';
      echo 'Age : '.$age.'<br/>';
    ?>
    <!-- Fin du programme PHP -->
    </body>
</html>
```

Ce programme effectue les actions suivantes :

1. Récupération des données, *via* la variable tableau `$_POST[ ]`, sur chaque élément `$_POST['nom']`, `$_POST['prenom']`, `$_POST['age']` , et affectation des variables `$nom`, `$prenom`, `$age`.
2. Conversion du nom et du prénom en majuscules, et de l'âge en entier.
3. Affichage des données traitées, *via* l'instruction `echo`.

La figure 3.2 présente le résultat de l'exécution de ce programme sur les données saisies dans le formulaire.

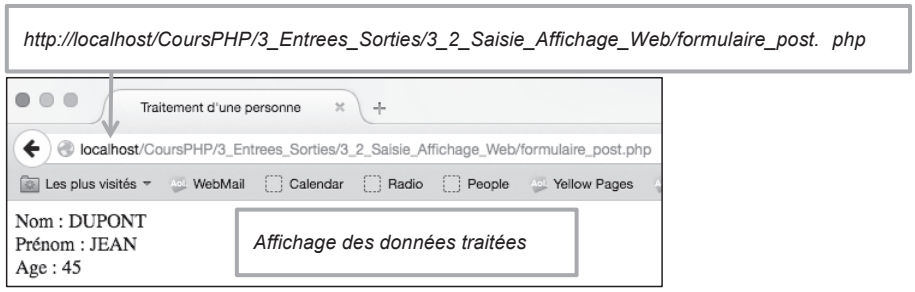

Figure 3.2 – Exécution de formulaire_post.php.

La méthode GET

Avec la méthode GET, les informations passées à la validation sont *visibles* dans le champ URL du navigateur. Le fichier `formulaire_get.html` qui implémente cette méthode est identique à `formulaire_post.html`. Seule la ligne d'en-tête du formulaire change. Le listing 3.13 présente la ligne d'en-tête et de fin du formulaire, et les modifications. Les autres lignes sont remplacées par des « … ».

Listing 3.13 – Fichier formulaire_get.html

```
...
<form action="formulaire_get.php" methode="get">
...
</form>
...
```

L'écran de saisie est identique à celui de la figure 3.1. Le bouton « Valider » envoie les données à traiter *via* la méthode GET au programme PHP `formulaire_get.php` identique au programme `formulaire_post.php`. Seules les trois lignes récupérant les données transmises sont modifiées. Le listing 3.14 présente les principales lignes, les autres lignes sont remplacées par des « … ».

Listing 3.14 – Programme formulaire_get.php

```
<!DOCTYPE html>
<html>
    <head>
     <meta charset="utf-8" />
     <title>Traitement d'une personne</title>
    </head>
    <body>
    <!-- Début du programme PHP -->
    <?php
      // == On récupère les données ==
      $nom    = $_GET['nom']   ;
      $prenom = $_GET['prenom'];
```

```
        $age    = $_GET['age']   ;
        // == On traite les données ==
        ...
    ?>
    <!-- Fin du programme PHP -->
    </body>
</html>
```

Ce programme récupère les données *via* le tableau `$_GET[ ]`, sur chaque élément `$_GET['nom']`, `$_GET['prenom']`, `$_GET ['age']`, et les affecte aux variables `$nom`, `$prenom`, `$age`. Les autres traitements restent inchangés.

L'affichage dans le navigateur du résultat de l'exécution du programme PHP montre que les données transmises au programme PHP apparaissent dans l'URL.

```
http://localhost/CoursPHP/3_Entrees_Sorties/3_2_Saisie_Affichage_Web/
formulaire_get.php?nom=Dupont&prenom=Jean&age=45
```

Le nom du programme PHP est suivi par le caractère « **?** », puis des informations ayant la forme **nom_de_la_variable=valeur**, séparées par le caractère « **&** », comme cela est présenté à la figure 3.3.

Figure 3.3 – Exécution de formulaire_get.php.

Avec la méthode GET, les données transmises au programme PHP sont visibles dans l'URL, ce qui constitue une faille de sécurité. L'utilisateur peut injecter des nouvelles valeurs au programme PHP en modifiant « à la main » les données de l'URL puis en rechargeant la page, contournant ainsi les éventuels contrôles définis sur la page précédente du formulaire HTML.

Mixte des méthodes GET et POST

Les arguments peuvent être passés à la fois en POST et en GET. Il faut indiquer `method="post"`, et passer les variables à transmettre en GET directement dans le champ `action`. Le fichier `formulaire_post_get.html` implémente cette méthode. La variable `categorie` est passée en GET avec la valeur `Elève`.

Listing 3.15 – Fichier formulaire_post_get.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Saisie d'une personne</title>
 </head>
 <body>
  <form action="formulaire_post_get.php?categorie=Elève"
method="post">
   Nom : <input type="text" name="nom" size="20" /><br/>
   Pr&eacute;nom : <input type="text" name="prenom" size="30" /><br/>
   Age : <input type="text" name="age" size="10" /><br/><br/>
   <input type="submit" value="Valider" />
   <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

Le bouton « Valider » envoie les données à traiter *via* les méthodes POST et GET au programme `formulaire_post_get.php`. Les lignes identiques aux programmes précédents sont remplacées par des « ... ».

Listing 3.16 – Programme formulaire_post_get.php

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>Traitement d'une personne</title>
 </head>
 <body>
 <?php
  // == On récupère les données passées par GET ==
  $categorie=$_GET['categorie'];
  // == On récupère les données passées par POST ==
  $nom    = $_POST['nom']   ;
  $prenom = $_POST['prenom'];
  $age    = $_POST['age']   ;
  // == On traite les données ==
  ...
  // == On affiche les données  ==
  echo 'Nom : '.$nom.'<br/>';
  echo 'Pr&eacute;nom : '.$prenom.'<br/>';
  echo 'Age : '.$age.'<br/>';
  echo 'Catégorie : '.$categorie.'<br/>';
 ?>
 </body>
</html>
```

La figure 3.4 montre que la donnée transmise *via* GET est visible dans l'URL.

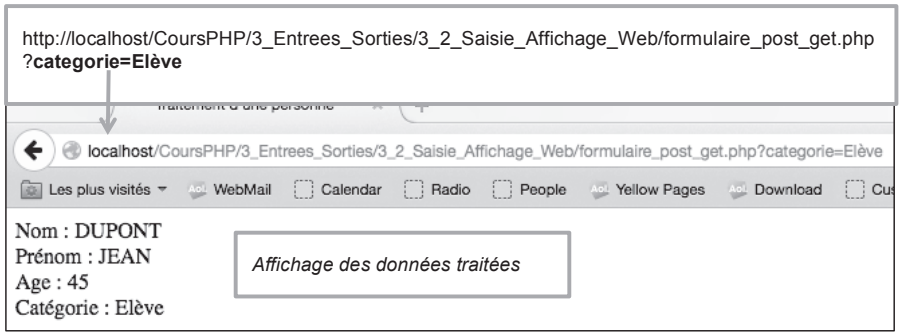

Figure 3.4 – Exécution de formulaire_post_get.php.

Remarque

Ces programmes utilisent la balise `<meta charset="utf-8" />` pour coder les caractères accentués directement en UTF8.

Les arguments de l'URL

Le formulaire en méthode GET montre que l'on peut passer des arguments dans la syntaxe de l'URL, soit manuellement dans le champ URL, soit *via* des syntaxes particulières dans les fichiers HTML ou PHP.

La modification manuelle

Il est possible de réécrire manuellement l'URL dans l'en-tête du navigateur, et ainsi d'appeler le `formulaire_get.php`, avec d'autres valeurs.

Avec la saisie des informations « Dupont », « Jean », « 45 » dans le formulaire et après sa validation, l'URL suivante apparaît dans l'en-tête du navigateur :

```
http://localhost/CoursPHP/3_Entrees_Sorties/3_2_Saisie_Affichage_Web/
formulaire_get.php?nom=Dupont&prenom=Jean&age=45
```

Il suffit alors de remplacer les valeurs des variables `nom`, `prenom` et `age` directement dans l'URL. Par exemple :

```
http://localhost/CoursPHP/3_Entrees_Sorties/3_2_Saisie_Affichage_Web/
formulaire_get.php?nom=Superman&prenom=Clark&age=102
```

Une simple validation envoie ces valeurs au programme PHP.

La balise HTML href

La balise HTML `href` crée un lien qui peut pointer vers un programme PHP. La page HTML `balise_href.html` envoie des données, directement au programme `formulaire_get.php`, sans passer par une saisie :

Listing 3.17 – Fichier balise_href.html

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>Envoi des données d'une personne</title>
 </head>
 <body>
  <p><a href="formulaire_get.php?nom=Dupont&prenom=Jean&age=45">Cliquez
sur ce lien pour envoyer les données</a></p>
 </body>
</html>
```

La figure 3.5 présente cette page HTML. Le résultat obtenu est identique à la figure 3.3.

Figure 3.5 – Appel d'un programme PHP via href.

L'instruction PHP header

L'instruction PHP header, effectue la redirection d'URL. Contrairement à `href` en HTML, aucune action de l'utilisateur n'est requise. Voici le programme `instruction_header.php`.

Listing 3.18 – Programme instruction_header.php

```
<?php
  // Redirection du visiteur vers la page du formulaire_get
  header('Location: formulaire_get.php?nom=Dupont&prenom=Jean&age=45');
?>
```

Cette instruction ne doit pas être précédée par une balise HTML envoyée au navigateur, même pas `<html>`.

Les risques

Cette section montre que la possibilité de saisir les données directement dans l'URL comporte des risques. Cela peut permettre à l'utilisateur de saisir des balises HTML, ou des requêtes SQL à la place des données attendues, et ainsi de pirater le site par la méthode d'*injection de code*.

Il est préférable de cacher les variables *via* la méthode POST par exemple, ou de passer les données entre les pages *via* des *variables de session*. Enfin, il faut contrôler les valeurs reçues par le programme PHP, avant de les traiter, et éviter tout risque de piratage par injection de code.