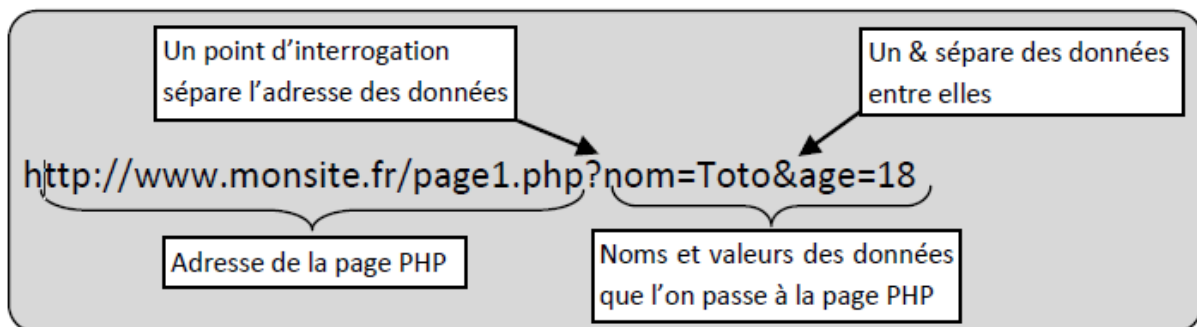




## Transmission de données via l'URL

Une URL (Uniform Resource Location) est une adresse qui permet d'accéder à une page web. On peut ajouter des paramètres (que l'on veut faire passer à la page) à la fin de l'adresse : - Les paramètres sont séparés de l'adresse par un point d'interrogation ? - Les paramètres transmis sont de la forme nom du paramètre = valeur du paramètre - les paramètres sont séparés entre eux par le symbole &

Exemple :



Remarque 1 : Une URL est limitée à 256 caractères, on ne peut donc pas transmettre énormément de données.

Remarque 2 : Une URL est très facilement modifiable par son utilisateur, il faudra donc vérifier la validité des données transmises (cf chap3.)

Créer un lien avec paramètres

Rappel : Un lien est réalisé grâce à la balise HTML `<a>...</a>` et à son attribut 'href' qui a pour valeur l'URL du lien. Pour passer les paramètres au lien, il suffit de donner à l'attribut 'href' la valeur de l'URL (avec la partie adresse mais aussi la partie donnée).

Récupérer les paramètres de l'URL :

La page à qui est adressée les paramètres va automatiquement créer un array associatif du nom de `$_GET`. C'est un array associatif dont les clés auront les noms des données que l'on a passées dans l'URL. De plus, à chaque clé sera associée la valeur de la donnée portant le même nom.

Ainsi, dans l'exemple précédent, `page1.php` va créer un tableau associatif `$_GET` avec 2 éléments :

- `$_GET['nom']` qui a pour valeur : Toto
- et `$_GET['age']` qui a pour valeur : 18

Maintenant que l'on sait où sont les données passées dans l'URL, on peut les exploiter.  
Ex : on écrit le code suivant sur la page correspondant au lien (`page1.php` en l'occurrence).

```
<p>Bonjour, tu t'appelles <?php echo $_GET['nom'];> et tu as <?php echo $_GET['age'];></p>
```

Ce code affichera : Bonjour, tu t'appelles Toto et tu as 18



## Transmission de données via des formulaires

Les formulaires sont les principaux moyens de récupérer des informations sur un site web. La création des différents formulaires est expliquée en détail dans le chapitre consacré aux formulaires.

Pour rappel :

- Un formulaire se met entre les balises `<form>` et `</form>`
- L'attribut 'action' de la balise `<form>` indique la page de destination des données.
- L'attribut 'méthod' de la balise `<form>` indique par quel moyen les données sont envoyées à la page de destination.
- L'attribut 'method' de la balise `<form>` peut prendre 2 valeurs :
  - Si `method="get"` : Les données sont envoyées via l'URL et on peut y accéder grâce au array associatif `$_GET` (dont nous avons vu le fonctionnement au chapitre 1)
  - Si `method="post"` : Les données sont envoyées via le canal normal et on peut y accéder grâce au array associatif `$_POST` (qui fonctionne suivant le même principe que `$_GET`).

Remarque : On utilise presque toujours la méthode post pour l'envoi des données d'un formulaire.

`$_POST[' valeur de l'attribut name de la balise <input> ']` (si `method="get"`, remplacer `$_POST` par `$_GET`)

Balise définissant la petite zone de texte

Cette variable contiendra ce qui a été saisi dans la petite zone de texte.

Exemple :

```
<form method="post" action="traitement.php">
  <label for="pays" >Rentrer votre pays </label>
  <input id ="pays" name="pays" value="France"/>
</form>
```

→

Dans `traitement.php`, une variable `$_POST['pays']` sera générée et contiendra ce qui a été saisi dans la petite zone de texte  
 Donc, si on a rentré France dans la petite zone de texte, nous aurons dans la page `traitement.php` : `$_POST['pays']='France'`

## Vérifier les données transmises

Tester la présence d'un paramètre :

Avant d'utiliser une variable qui n'a pas été définie dans la page courante, il faut vérifier qu'on l'a bien reçue. En effet si on fait appel à une variable alors qu'elle n'existe pas, le programme plantera lamentablement. La fonction `isset` permet de tester l'existence d'une variable. Elle prend comme paramètre le nom de la variable dont on veut vérifier l'existence et renvoie un booléen égal à 'true' si la variable existe (et égal à 'false' si elle n'existe pas).



## Exemple :

```
<?php
    if(isset($_GET['nom']))
        echo'bonjour ' . $_GET['nom'];
    else echo'Veuillez vous identifier.';
?>
```

Ce code test l'existence de la variable `$_GET['nom']` qui est censé être arrivée via l'URL. Si cette variable existe on dit bonjour au visiteur sinon on lui demande de s'identifier.

## Vérifier la conformité d'une valeur :

Lorsque que l'on récupère une variable venant de l'extérieur, elle n'a peut être pas le type que l'on attend. Par exemple, dans un formulaire où l'on demande l'âge, rien n'empêche de répondre par du texte. Cela risque d'être problématique lors du traitement de cette variable. Pour remédier à cela, on peut effectuer du transtypage, c'est-à-dire que l'on va convertir la variable reçue dans le type que l'on souhaite. Pour modifier le type d'une variable, on écrit devant la variable et entre parenthèses le nom du nouveau type (int, bool, double, string, array...).

## Exemple :

```
<?php
$_GET['age']=(int)$_GET['age'];
?>
```

Si `$_GET['age']=18.5` avant le transtypage, `$_GET['age']=18` après le transtypage  
Si `$_GET['age']='dix'` avant le transtypage, `$_GET['age']=0` après le transtypage  
→ la conversion d'un texte en entier ne peut pas être logique.

## Visibilité des variables

Une variable n'est pas forcément visible à tous les points d'un programme. Par exemple, une variable déclarée dans le programme principal n'est pas accessible à une fonction si elle n'a pas été passée en paramètre à la fonction. Il y a 3 niveaux de visibilité des variables :

- Les variables superglobales : Ce sont des arrays générés automatiquement par PHP (comme `$_GET` et `$_POST`). Elles sont visibles de partout dans le programme (même à l'intérieur d'une fonction). A une exception près, leur nom commence par un underscore (`_`) et elles sont écrites en majuscule.
- Les variables globales : Ce sont toutes les variables, tableaux et constantes que nous créons dans le programme principal. (Elles ne sont pas visibles dans les fonctions si on ne les passe pas en paramètre).
- Les variables locales : Ce sont les variables déclarées dans les fonctions. Elles ne sont accessibles que dans la fonction, le programme ne peut pas agir sur elles, d'ailleurs, elles sont automatiquement supprimées de la mémoire à la fin de la fonction. Attention, ceci n'est valable que pour les variables déclarées normalement dans la fonction (cf remarque ci-dessous).

Remarque : Portée des variables déclarées dans une fonction. Dans une fonction, un variable déclarée : - normalement (sans rien devant son nom) est par défaut une variable locale. Elle n'est donc visible que dans la fonction et est supprimée à la fin de l'exécution de la fonction. - avec le mot clé 'global' devant son nom, est considérée comme global. Elle est donc visible de partout dans le programme et n'est pas supprimée à la fin de l'exécution de la fonction. - avec le mot clé 'static' devant son nom, n'est accessible qu'à l'intérieur de la fonction mais elle n'est pas supprimée à la fin de son exécution. Elle sera donc à nouveau disponible lors du prochain appel de la fonction.



## Transmission de données via les sessions

- Grâce aux variables super globales `$_GET` et `$_POST`, on peut transmettre des données à une page, mais celles-ci ne seront accessibles que de cette page là. Dès que l'on va quitter cette page les données seront oubliées. Si on veut qu'elles soient accessibles de toutes les pages, il faut utiliser un système de session.
- Une session est en fait un fichier stocké sur le serveur et dans lequel on sauvegarde des données.
- Les informations de session du serveur sont accessibles au client grâce à un numéro hexadécimal unique (I.D. de session) attribué au client lors de la création de la session.
- Une fois qu'une session est générée, on peut y enregistrer toutes les données que l'on désire. La gestion des données de sessions se fait à l'aide de l'array super global `$_SESSION`.
- Une session a une durée de vie limitée, elle est détruite automatiquement après un certain temps d'inactivité (en général le timeout est d'une vingtaine de minutes).
- Une page fait appel au système de session grâce à la fonction `session_start()`. Celle-ci ouvre la session existante ou si aucune session n'est présente, en crée une et lui attribue un ID de session. ATTENTION, la fonction `session_start` doit être appelée avant toute écriture de code HTML.
- Une session est supprimée à l'aide de la fonction `session_destroy()`. Celle-ci est automatiquement appelée à la fin du timeout.

## Exemple d'utilisation de sessions :

- Dans les sites de ventes en ligne, lorsqu'on met des articles dans le panier et que l'on veut garder en mémoire le contenu du panier d'une page à l'autre.
- Lorsque qu'un site demande une authentification par login et mot de passe, on peut les enregistrer dans des variables de sessions afin de ne pas avoir à les demander à chaque page.



## Exemple de programmation :

### Index.php

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>test sessions</title>
  </head>

  <body>

    <form method="post" action="page1.php">
      <label for="nom">quel est ton nom ?</label>
      <input type="text" name="nom"><br><br>
      <input type="submit" value="Envoyer">

    </body>
  </html>
```

quel est ton nom ?

Envoyer

### Page1.php

```
<?php session_start();?>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>page 1</title>
  </head>

  <body>

  <?php
if(isset($_POST['nom']))
{
$_SESSION['nom']=$_POST['nom'];
echo "bonjour ".$_SESSION['nom'].", maintenant je me souviendrai de ton nom sur toutes les pages de mon site<br>";
echo "<a href='\"page2.php\">clic ici pour vérifier</a>";
}
else echo "Identification <a href='\"index.php\">ici</a>";
?>
  </body>
</html>
```

bonjour SNIR, maintenant je me souviendrai de ton nom sur toutes les pages de mon site  
[clic ici pour vérifier](#)



## Page2.php

```
<?php session_start();?>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>page 1</title>
  </head>

  <body>

<?php
if(isset($_SESSION['nom']))
{
echo "bonjour ".$_SESSION['nom'].", Tu vois je n'ai pas oublié ton nom<br>";
}
?>
  </body>
</html>
```

bonjour SNIR, Tu vois je n'ai pas oublié ton nom

## Transmission de données via les cookies

### Présentation des cookies :

Comme les sessions, les cookies sont des fichiers qui permettent de stocker des données qui seront consultables sur toutes les pages d'un site. Les cookies sont toutes fois moins évolués que les sessions :

- Ce sont des fichiers textes ne dépassant pas 4ko et ne contenant qu'une seule information à la fois.
- Ils sont stockés sur l'ordinateur du client (et non sur le serveur, ce qui était le cas pour les sessions). Chaque utilisateur de l'ordinateur peut les lire ou les modifier, il ne faut donc pas y mettre des données sensibles.
- Ils sont gérés par les navigateurs qui peuvent les refuser.
- On ne peut pas avoir plus de 20 cookies par nom de domaine. • Comme les sessions, ils ont une durée de vie limitée.

### Génération d'un cookie :

La création d'un cookie repose sur l'envoi d'entêtes HTTP au navigateur du client au moyen de la fonction `setcookie()`. Cela sous-entend qu'il faudra l'appeler avant tout envoi de données au navigateur, il ne faut donc jamais placer de code HTML avant la fonction `setcookie()`.



La fonction `setcookie()` peut prendre jusqu'à 7 paramètres. Seul le premier est obligatoire car il définit le nom du cookie. Elle renvoie un booléen : `true` en cas de succès et `false` si échec.

Signature de la fonction `setcookies` :

```
bool setcookie ( string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false ]]]]] ] )
```

Paramètre	Explications
<code>name</code>	Nom du cookie
<code>value</code>	Valeur du cookie
<code>expire</code>	Date d'expiration du cookie au format timestamp UNIX, c'est à dire un nombre de secondes écoulées depuis le 01 janvier 1970. On fixe cette valeur à l'aide de la fonction <code>time()</code> qui retourne le nombre de seconde depuis cette date (on peut aussi utiliser la fonction <code>mktime()</code> . <b>Exemple</b> : Si on veut que le cookie expire dans 1 mois, il faut écrire : <code>time()+30*24*3600</code> Si on ne spécifie pas ce paramètre, le cookie expire à la fin de la session.
<code>path</code>	Chemin sur le serveur dans lequel le cookie sera valide. Si la valeur <code>'/'</code> est spécifiée alors le cookie sera visible sur tout le domaine. Si <code>'/examples/'</code> est précisé alors le cookie sera visible dans le répertoire <code>/examples/</code> et tous ses sous-dossiers. La valeur par défaut est le répertoire courant où le cookie a été défini.
<code>domain</code>	Domaine sur lequel le cookie sera valable. Si la valeur est <code>'.domaine.com'</code> alors le cookie sera disponible sur tout le domaine (sous-domaines compris). Si la valeur est <code>'www.domaine.com'</code> alors le cookie ne sera valable que dans le sous-domaine <code>'www'</code>
<code>secure</code>	Ce paramètre prend pour valeur un booléen qui définit si le cookie doit être utilisé dans un contexte de connexion sécurisée par protocole HTTPS
<code>httponly</code>	Ce paramètre prend pour valeur un booléen qui définit si le cookie sera accessible uniquement par le protocole HTTP. Cela signifie que le cookie ne sera pas accessible via des langages de scripts, comme Javascript. Cette configuration permet de limiter les attaques via XSS (bien qu'elle ne soit pas supportée par tous les navigateurs). Ajouté en PHP 5.2.0

En général, on utilise la fonction `setcookie()` avec seulement les 3 premiers paramètres .

Exemple de génération de cookies :

```
<?php
setcookie('nom','Toto',time()+30*24*3600);
setcookie('age',20,time()+30*24*3600);
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
  <head>
    <title>Index de mon site</title>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"/>
  </head>
  <body>
    <p>Cliquer <a href="page1.php">ici</a> pour aller à la page suivante.</p>
  </body>
</html>
```

Création de 2 cookies





## Lecture d'un cookie :

On accède aux cookies grâce à l'array superglobal `$_COOKIE`. Il suffit d'écrire le nom du cookie entre les crochets : `$_COOKIE[' nom_du_cookie ']`.

Exemple : Ici on va lire les cookies que l'on a crée dans l'exemple précédent.

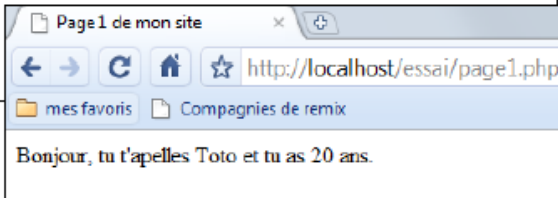
```

!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
  <head>
    <title>Page 1 de mon site</title>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"/>
  </head>

  <body>
    <?php
      echo"Bonjour, tu t'apelles ".$_COOKIE['nom']." et tu as ".$_COOKIE['age']. " ans.";
    ?>
  </body>
</html>

```

Lecture des 2 cookies créés dans l'exemple précédent.



## Suppression d'un cookie :

Pour supprimer un cookie, il suffit de générer un cookie de même nom mais sans valeur avec la fonction `setcookie()`, puis de lui appliquer la fonction `unset()`.

Exemple : Suppression du cookie 'age' de l'exemple précédent.

```

<?php
  // Suppression du cookie age
  setcookie('age');
  // Suppression de la valeur en local
  unset($_COOKIE['age']);
?>

```